FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

On Preventing Violations of Service Level Agreements in Composed Services Using Self-Adaptation

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Sozial- und Wirtschaftswissenschaften

eingereicht von

Philipp Leitner
Matrikelnummer 0225511

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Schahram Dustdar

Diese Dissertation haben begutachtet:

(Univ.Prof. Schahram Dustdar) (Prof. Fabio Casati)

Wien, 19. Oktober 2011

(Philipp Leitner)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Philipp Leitner
Kresengartengasse 10/14, 1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Philipp Leitner)

Acknowledgments

The research leading to this dissertation has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube).

Danksagung

Mit der vorliegenden Abschlussarbeit beende ich nun (endlich!) ein Wirtschaftsinformatik-Studium, an dem ich mehr als ein Drittel meines bisherigen Lebens gearbeitet habe. Dieser Abschluss kann daher zu Recht als persönlicher Meilenstein für mich angesehen werden. Zum Erreichen dieses grossen Zieles haben viele Menschen beigetragen, bei denen ich mich an dieser Stelle sehr herzlich bedanken will:

An allerersten Stelle sind das natürlich meine Eltern, die mich in meinem Studium immer finanziell und emotional unterstützt haben. Mein Erfolg ist immer auch ihr Erfolg. Eine weitere wichtige Rolle beim Entstehen dieser Arbeit hat mein Betreuer, Prof. Schahram Dustdar, gespielt. Ich bedanke mich herzlich für die Chance meine Dissertation in der Distributed Systems Group (DSG) zu schreiben, und für eine Arbeitsatmosphäre in der ich meine eigenen Forschungsideen entwickeln und verfolgen konnte.

Weiters möchte ich mich bei all meinen Kollegen in der DSG für die grossartigen Zusammenarbeit über die letzten Jahren bedanken. Im Besonderen gilt dieser Dank Florian Rosenberg, Anton Michlmayr und Waldemar Hummer, die an den meisten meiner Forschungsvorhaben beteiligt waren, und durch Zustimmung, wie auch Kritik, die Arbeit hinter dieser Dissertation geformt haben. Ebenso danke ich Prof. Fabio Casati, der dankenswerterweise die Zweitbegutachtung dieser Arbeit übernommen hat.

Der meiste Dank aber gebürt Julia Schwarz, die mich über (fast) den kompletten Verlauf der Dissertation begleitet hat, die es toleriert hat wenn ich Abende vor dem Computer verbringen musste, und die mich aufgebaut hat wenn ich mit mir selbst und meiner Arbeit nicht zufrieden war.

Abstract

Providers of composite Web services face the challenge of having to comply to Service Level Agreements (SLAs), which are agreements governing the minimum performance that customers can expect from a composite service. SLAs contain Service Level Objectives (SLOs), concrete numerical Quality of Service (QoS) objectives, which the service needs to fulfill. If objectives are violated, agreed upon consequences (usually taking the form of penalty payments) go into effect. However, fulfilling SLAs can also lead to costs for the service provider (e.g., because the composite service provider needs to use more expensive services itself, or because of the costs inherent to optimizing the composition). Therefore, it is not trivial for the provider to decide to what extend the service's SLAs should be fulfilled, or which SLAs should (temporarily) be violated for economical reasons. Even more so, these decisions should ideally be automated, to allow for fast reactions to changes in the business environment.

In this thesis, a framework for optimizing adaptations of service compositions with regards to SLA violations has been developed. The framework, dubbed PREvent (Prediction and Prevention of SLA Violations Based on Events), uses techniques from the areas of machine learning and heuristic optimization to construct models for prediction of SLA violations at runtime, and to decide which adaptation actions may be used to improve overall performance in a composition instance. An optimizer component decides, whether applying these changes makes sense economically (i.e., whether the costs of violating the SLAs are bigger than the adaptation costs). If this is the case, the respective actions are applied in an automated way. At its core, Prediction and Prevention of SLA Violations Based on Events (PREvent) is a self-optimizing system in the sense of autonomic computing, with the target of minimizing the total costs of adaptations and SLA violations for the service provider.

The outcomes of the thesis are explained and evaluated based on an illustrative case study. Results show that, using the PREvent system, providers of composite Web services are able to reduce their total costs. Furthermore, for service clients, the advantage of PREvent is that the number of SLA violations is reduced. This in turn increases client satisfaction.

Kurzfassung

Anbieter von Web Diensten (engl. Web services), die durch eine Komposition existierender Dienste implementiert werden, sind oft mit der Herausforderung konfrontiert, vertragliche Verpflichtungen (sogenannte Dienstgütevereinbarungen, engl. Service Level Agreements) technisch einhalten zu müssen. Dienstgütevereinbarungen regeln die minimale Dienstqualität, die Kunden des Dienstes für jede Benutzung erwarten dürfen. Wenn diese Vereinbarungen nicht eingehalten werden können, ist im Allgemeinen eine vereinbarte Strafzahlung (Pönale) fällig. Allerdings kann auch das Einhalten der Vereinbarung zu Kosten für den Dienstanbieter führen, zum Beispiel, weil der Anbieter auf kostpieligere Basisdienste innerhalb seiner Komposition zurückgreifen muss, um die Dienstgütevereinbarungen einzuhalten. Daher gibt es Fälle, in denen es ökonomisch sinnvoll ist, bestimmte Vereinbarungen bewusst zu verletzen, auch wenn ein Einhalten der Dienstgütevereinbarung technisch möglich gewesen wäre. Die Entscheidung, ob eine Vereinbarung eingehalten oder gebrochen werden soll ist daher nicht trivial. Zusätzlich sollte diese Entscheidung optimalerweise automatisiert getroffen werden, um ein schnelles Reagieren auf Änderungen im Geschäftsumfeld des Dienstes zu ermöglichen.

In der vorliegenden Abschlussarbeit wird ein System vorgestellt, dass optimale Laufzeitanpassung von Dienstkompositionen in Bezug auf Dienstgütevereinbarungen realisiert. Dieses System (genannt PREvent, Prediction and Prevention of SLA Violations Based on Events) nutzt Techniken des maschinellen Lernens und der heuristischen Optimierung, um Modelle zur Vorhersage von Verletzungen der Vereinbarungen zu generieren. Diese Modelle werden auch benutzt, um zur Laufzeit des Dienstes zu entscheiden, welche Anpassungen vorgenommen werden sollten, um die Qualität des Dienstes zu verbessern. Eine Optimierungskomponente entscheidet, ob diese Anpassungen auch ökonomisch sinnvoll sind, das heißt, ob die Pönale für die Verletzung der Vereinbarung höher ist als die Kosten der Anpassung. Wenn dies der Fall ist, werden die Anpassungen automatisiert vorgenommen. Allgemein gesprochen ist PREvent ein selbstoptimierendes System im Sinne des autonomen Systemmanagements (engl. autonomic computing), dessen Ziel die Minimierung der Gesamtkosten für den Dienstanbieter ist.

Die Ergebnisse der vorliegenden Abschlussarbeit werden an Hand eines beispielhaften Dienstes erörtert und evaluiert. Die Arbeit zeigt, dass Dienstanbieter mit Hilfe des PREvent Systems ihre Gesamtkosten signifikant senken können. Zusätzlich wird auch die Anzahl an Verletzungen von Dienstgütevereinbarungen reduziert, wodurch auch die Kundenzufriedenheit gesteigert werden kann.

Contents

List of Tables

List of Figures

List of Listings

Earlier Publications

This thesis is based on work published in scientific conferences, workshops, journals and books. For reasons of brevity, these core papers, which build the foundation of this thesis, are listed here once, and will generally not be explicitly referenced again. Parts of these papers are contained in verbatim. Please refer to Appendix C for a full publication list of the author of this thesis.

- Leitner, P., Hummer, W., Satzger, B., and Dustdar, S.: Stepwise and Asynchronous Runtime Optimization of Web Service Compositions (Short Paper), In *Proceedings of the 12th International Conference on Web Information System Engineering (WISE 2011)*, 2011.
- Leitner, P., Hummer, W., and Dustdar, S.: Cost-Based Optimization of Service Compositions, *IEEE Transactions on Services Computing (TSC)*, 2011. To Appear.
- Leitner, P., Wetzstein, B., Karastoyanova, D., Hummer, W., Dustdar, S. and Leymann, F.: Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution, In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'10)*, pp. 365–380, IEEE Computer Society, 2010. http://dx.doi.org/10.1007/978-3-642-17358-5_25
- Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S.: Monitoring, Prediction and Prevention of SLA Violations in Composite Services, In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 369–376, IEEE Computer Society, 2010. http://dx.doi.org/10.1109/ICWS.2010.21
- Leitner, P., Rosenberg, F., Michlmayr, A., Huber, A., and Dustdar, S.: A Mediator-Based Approach to Resolving Interface Heterogenity of Web Services, In *Emerging Web Services Technology Volume III, Whitestein Series in Software Agent Technologies and Autonomic Computing*, pp. 55–74, 2010. http://dx.doi.org/10.1007/978-3-0346-0104-7_4
- Wetzstein, B., Leitner, P., Rosenberg, F., Dustdar, S. and Leymann, F.: Identifying Influential Factors of Business Process Performance Using Dependency Analysis, *Enterprise Information Systems vol.4, nr.3*, pp. 1–8, Taylor & Francis, 2010. http://dx.doi.org/10.1080/17517575.2010.493956
- Michlmayr, A., Rosenberg, F., Leitner, P., and Dustdar, S.: End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo, In *IEEE Transactions on Services Computing (TSC) vol.3, nr.3*, pp. 193–205, IEEE Computer Society, 2010. http://dx.doi.org/10.1109/TSC.2010.20

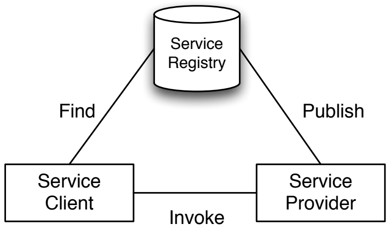
- Hummer, W., Leitner, P., Michlmayr, A., and Rosenberg, F.: VRESCo – Vienna Runtime Environment for Service-oriented Computing, In *Service Engineering – European Research Results*, pp. 299–324, Springer, 2010.
- Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S. and Leymann, F.: Runtime Prediction of Service Level Agreement Violations for Composite Services, In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09), co-located with ICSOC'09*, pp. 176-186, Springer, 2009. http://dx.doi.org/10.1007/978-3-642-16132-2_17
- Leitner, P., Rosenberg, F., and Dustdar, S.: Daios: Efficient Dynamic Web Service Invocation, *IEEE Internet Computing vol.13, nr.3*, pp. 72–80, IEEE Computer Society, 2009. http://dx.doi.org/10.1109/MIC.2009.57

CHAPTER 1

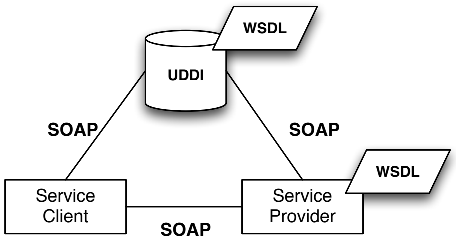
Introduction

In the last years, an ever increasing number of companies are discovering the merits of an Information Technology (IT) infrastructure based on a Service-Oriented Architecture (SOA) [51]. Most importantly, building business applications based on such a SOA, usually referred to as Service-Oriented Computing (SOC) [145], allows solution architects to reuse existing internal and external services via platform-independent protocols and interfaces defined in a machine-readable format. The idea of building higher-level business applications mostly or purely by assembling existing services in a novel way, is called service composition [48]. Services are typically discovered through a service registry, which decouples service provider and service client. In this way, the well-known SOA triangle [76] is established (see Figure 1.1a). By far the most widely used technology for implementing the SOA triangle are Web services [38], i.e., the three implementation specifications of SOAP [196] as message format, Web Service Definition Language (WSDL) [197] as service description language and Universal Description Discovery and Integration (UDDI) [173] as service registry. This is depicted in Figure 1.1b. By now, hard reality has shown that, for practical applications, only SOAP and WSDL are essential, while UDDI has failed to strike a significant impact, and has almost faded into obscurity by today. This can best be seen in the fact that Microsoft, SAP and IBM have discontinued their public UDDI registries as early as 2005 [129].

Today, service composition remains one of the main topics of research in SOA, especially when it comes to dynamic and adaptive composite services [144]. Service composition is strongly related to another topic currently taking off, the idea of Business Process Management (BPM) [175]. Strictly speaking, BPM is not a new idea, going back to work on business process reengineering in the early and mid nineties [13, 40, 69]. However, while business process reengineering was, by and large, an organisational issue (as opposed to an IT topic), SOA and service composition, along with the related development of automated workflow management [63], allowed to map business processes directly to IT in a way that has not been possible before. In this way, business processes went from abstract entities that existed mostly in the minds of managers (and in the organisational documentations produced by consultants) to concrete programs that could be deployed, executed and monitored. Hence, service composition and composition lan-



(a) The Triangle of SOA



(b) Web Services Basic Standards

Figure 1.1: SOA Basics

guages like the Web Service Business Process Execution Language (WS-BPEL) are more than just a programming environment for SOC. Much more, they are tools to formalize concepts that make sense on business level (processes and business activities) and map them to executable artifacts in the IT infrastructure (service compositions and Web services).

1.1 Problem Statement

Through these developments, service composition has also gained significant importance for day-to-day business operations for many SOA-based companies. In the wake of these developments, monitoring and quality assurance of service compositions has gained more prominence, especially if these compositions interface to external business partners. Building on top of the existing idea of Web service Quality of Service (QoS) [121], providers and service customers now often agree upon contracts regulating the minimal quality of the provided composite service. These contracts are called Service Level Agreements (SLAs) [96]. Concrete objectives in SLAs are called Service Level Objectives (SLOs).

For providers of service compositions, it is important to minimize cases of SLA violations for two reasons. Firstly, SLOs are usually associated with financial penalties, i.e., if a provider is unable to provide the promised service quality some monetary penalties have to be paid. Secondly, frequent SLA violations drastically reduce the customer satisfaction of the users of the service, resulting in a loss of customers. For these reasons, providers generally monitor the performance of their processes and composite services, often in an event-driven way [163]. This is one aspect of the important area of Business Activity Monitoring (BAM) [175]. If monitoring reveals performance problems, human business analysts use techniques such as data mining [187] or top-down reasoning [19] to identify the reasons for these problems, and manually change the composition to improve its quality. This approach is certainly useful, but it can only remove causes of violations in an *ex post* way, i.e., after SLA violations have already happened. Evidently, using monitoring alone it is not possible to prevent violations *ex ante*, before they have happened. To allow for more timely prevention of violations, one needs the possibility to predict violations before they have happened, decide which actions can be carried out, and apply these actions. Deciding on which actions to apply is particularly challenging, as one needs some knowledge about the impact that a given action will have on the SLOs of a composition.

Furthermore, this approach needs to be fully automated, since monitoring, prediction and adaptation needs to be done continuously (unlike the more traditional *ex post* approach, which is done offline and can hence rely on human intuition and decision making).

However, preventing SLA violations is, in general, not for free for the service provider. For instance, some alternative services usable in a composition may provide faster response times (thereby improving the response time of the composite service, and reducing the probability of violating response time related SLOs), but those services are often more expensive than slower ones. Therefore, there is an apparent tradeoff between preventing SLA violations and the inherent costs of doing so. In literature, this tradeoff is currently not considered sufficiently. Instead, researchers often assume that the ultimate goal of service providers is to minimize SLA violations no matter what, ignoring the often significant costs of doing so (e.g., [4, 25, 95, 134]). Arguably, this does not reflect the actual incentives of the service provider very well. Instead, one should not consider violations and adaptation in isolation, but take the costs of adaptation into account when deciding on which actions to take in order to prevent violations (or, in some cases, deciding to not prevent a violation at all). Specifically, there can be cases where no economically feasible action exists to prevent a given violation, i.e., all actions that could potentially prevent a violation are more expensive than the violation itself. In this case the optimal action for the service provider is to ignore this violation.

1.2 Research Questions

The two problems outlined in Section 1.1 motivate the research conducted as part of this thesis work. More concretely, the following questions are researched in this work.

Research Question I:
How can SLA violations in service compositions be prevented in an ex ante way?

As discussed in Section 1.1, *ex ante* prevention of SLA violations requires means to predict violations before they have happened, decide on concrete actions and apply them in a fully automated way. In order to select the best actions to apply, one needs means to model the impact of an action on the SLOs of a composition. There is some existing work on partial aspects of these problems in isolation (these are discussed in Chapter 6), but no end-to-end approach exists, which fully integrates prediction, selection and adaptation into a general framework for automated prevention of SLA violations in service compositions.

Research Question II:
How can the costs of adaptation be considered in the action selection process of ex ante prevention of SLA violations?

The impact of actions on SLOs is not the only important factor when selecting the right actions to prevent predicted violations. Another important factor, that is generally ignored by existing frameworks for dynamic adaptation, is the cost of adaptation, i.e., how much more expensive it is to execute the adapted service composition as compared to the original one. In order to capture the real decision making process of providers of service compositions, one needs to take these costs into account when deciding on actions to prevent SLA violations, and whether a given violation should be prevented at all. No existing research work has modelled the action selection problem in a way that takes these costs into account.

1.3 Scientific Contributions

Guided by the research questions stated in Section 1.2, the following contributions to the state of the art in service composition research have been made.

Contribution I:
A machine learning based framework for predicting SLA violations before they have happened

Timely prediction of (potential) violations is a precondition for automatically preventing them. However, prediction of violations also has merits on its own right. For instance, even if no means are available to automatically trigger adaptations, a human responsible can still be informed about potentially troubling composition instances in advance, and take respective actions. Of course, this is only possible for longer-running compositions. As part of the Prediction and Prevention of SLA Violations Based on Events (PREvent) framework (which has been used to implement the technical contributions of this thesis), a machine learning based prediction approach has been implemented. Details are presented in Section 4.3. Contribution I has originally been presented in [104].

Contribution II:
A framework for automatically adapting service compositions at runtime, both by exchanging service bindings and by adapting the composition structure

Prediction of SLA violations becomes more powerful if means exist to automatically adapt a service composition. In this thesis, adaptation is considered on multiple levels, including exchanging single services in the composition (with the same or different service interfaces) and changing the composition structure. As with the prediction approach, these contributions have been implemented as part of the PREvent framework. This contribution is based on earlier work, which has been carried out as part of the Vienna Runtime Environment for Service-Oriented Computing (VRESCo) [129, 130] and Dynamic And Asynchronous Invocation of Services (DAIOS) [110] projects. Details on automated adaptation of service compositions are discussed in Section 4.5. Contribution II has originally been presented in [109–112].

Contribution III:
A formal model of the decision making process of finding the best actions to prevent violations as one-dimensional optimization problem, which incorporates not only the benefits, but also the costs of adaptation

As explained in Section 1.1, the costs of adaptation are currently not considered by most researchers when adapting service compositions at runtime. In this thesis, an optimization problem is presented, which better reflects the incentives of service providers. This model is discussed in Section 4.4. Contribution III has originally been presented in [105].

Contribution IV:
A set of concrete deterministic and meta-heuristic algorithms to solve the aforementioned optimization problem (Contribution III) efficiently at runtime, and a stepwise optimization approach which is applicable to shorter-running compositions

As part of the PREvent framework, the optimization problem produced as Contribution III has been solved using a variety of different algorithms. More concretely, the problem has been solved with variants of Branch-and-Bound, local search and evolutionary computing. Note that this list of algorithms does not make any claim of completeness. Evidently, any other optimization algorithm can be applied to the model of Contribution III as well. Additionally, a novel stepwise, constructive approach to optimization has been introduced, which is especially helpful if the service composition to optimize is relatively short-running and requires timely decision making. The algorithms and the stepwise optimization procedure are discussed in Section 4.4. Contribution IV has originally been presented in [105, 106].

Contribution V:
A case study based end-to-end evaluation, which quantifies all aforementioned contributions (Contributions I – IV) and shows their benefits as compared to the state of the art

In order to validate the ideas discussed in Contributions I to IV, quantitative and qualitative evaluations have been carried out. For simplicity, a unified case study (ACMEBOT) has been used to describe and evaluate all contributions of this thesis. This case study is introduced in Section 3. The evaluation is discussed in Chapter 5. The basis of Contribution V has originally been presented as individual evaluations along with the individual contributions [104–106, 109, 110, 112].

The PREvent framework largely follows the vision of autonomic computing [97], i.e., the architecture of the system implemented as part of this thesis follows the seminal steps of "Monitoring", "Analysis", "Planning" and "Execution" (MAPE loop).

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 provides background information on important concepts and techniques used in the remainder of this thesis. In detail, the ideas of service composition (Section 2.1), Quality-of-Service (Section 2.2), Service Level Agreements (Section 2.3), autonomic computing (Section 2.4) and machine learning (Section 2.5) are discussed.

- Chapter 3 introduces the ACMEBOT case study, which is used as illustrative example in the thesis.

- Chapter 4 contains the most important part of the thesis, Contributions I to IV. This chapter is structured along the MAPE loop of autonomic computing, i.e., after an overview over the end-to-end system (Section 4.1), the components "Monitoring" (monitoring of service compositions, Section 4.2), "Analysis" (prediction of violations, Section 4.3), "Planning" (solving the decision making problem, Section 4.4) and "Execution" (applying adaptations, Section 4.5) are explained in detail.

- Chapter 5 contains Contribution V, the quantitative and qualitative evaluation of the remaining contributions of the thesis.

- Chapter 6 explains work related to one or more parts of the thesis. Most focus is put on existing work that either directly contributed to some part of this thesis, or which is related to more than one of the research contributions.

- Chapter 7 concludes the thesis, and provides an outlook on future research directions opened up by this work.

CHAPTER 2

Background

In this chapter, some basic concepts, which are used in the remaining thesis, are presented. First of all, the idea of service composition is explained. Service compositions are the main subject discussed in this thesis. Afterwards, some QoS metrics, as well as their definition, are outlined. Based on these metrics, the concepts of SLAs, of SLOs, and of SLA violations are introduced. These concepts form the primary motivation of the thesis. Sucessively, the ideas of autonomous computing are covered, which are the primary means of how SLA violations are prevented in this thesis. Finally, the chapter ends with a brief discussion of machine learning and the concrete machine learning techniques of Artificial Neural Networks (ANNs) and decision trees. Machine learning is the basic technique that has been used for predicting SLA violations in this work.

2.1 Service Composition

One of the basic claims of Web services is their composability [76] into value-added structures, so-called service compositions. Service composition is often cited as one of the key research topics in SOC [144]. Generally, two types of composition can be distinguished [33, 148]:

- Service orchestration refers to compositions where one central controller "orchestrates" (steers) the execution logics. The control flow is centralized in a single composition engine. Typically, service orchestrations are intra-organizational. They reflect business processes within an organization, although some of the services used in the orchestration may well be external to the organization. These are the compositions most commonly seen in practice, and many tools and languages exist to model orchestrations. The most relevant languages today are the WS-BPEL [140] (although WS-BPEL is, to a limited extend, also able to describe service choreographies) and Windows Workflow Foundation (WWF) [166]. Historical examples include the Web Service Flow Language (WSFL) and XLANG, both of which strongly influenced the development of WS-BPEL.

- Service choreography is the more complex case of composition across organizational boundaries. In a choreography, no central controller exists. Instead, every organization participating in the choreography has their own composition engine, and control over the execution is passed between those engines using defined interfaces. No entity has knowledge of the whole choreography. Instead, every partner knows its own internal flow and the interfaces of the entities it interacts with. Because of these properties, choreography has proven hard to implement in practice and has gained little attention in industry so far. However, some languages exist in research to model chroeographies. The most important one today is the Web Service Choreography Description Language (WS-CDL) [194].

The work described in this thesis deals solely with issues of service orchestration. Hence, in the remainder, the terms composition and orchestration are used interchangeably. The thesis prototype has been implemented for the WWF service orchestration language. However, in principle, all research results discussed are also applicable for WS-BPEL (even if the implementation might prove to be more difficult in some cases). In the following, brief introductions to both WS-BPEL and WWF are given.

Web Service Business Process Execution Language (WS-BPEL)

WS-BPEL is the composition language most commonly used today, to the extent that oftentimes the terms "service composition" and "BPEL process" are used synonymously. WS-BPEL is essentially a block-structured special-purpose (programming) language based on Extensible Markup Language (XML), with the main feature that Web service invocations are considered first-class citizens. WS-BPEL process definitions are often authored in a graphical notation, which is then transformed into the XML representation executed by WS-BPEL engines. WS-BPEL is designed to be semantically close to the models of business processes, as created by business analysts and domain experts. Hence, a vivid research community deals with the automated or semi-automated translation of typical business process languages, such as the Business Process Modelling Notation (BPMN) [1], to WS-BPEL [142].

In WS-BPEL, partner services are represented using so-called partner links. Partner links are defined via WSDL interfaces. Hence, WS-BPEL, as a language, is strongly tied to WSDL, which makes the integration of services which are not using WSDL (e.g., RESTful Web services [155]) relatively hard [146]. As an XML-based language, WS-BPEL is inherently programming language neutral. However, at the moment, the WS-BPEL ecosystem is strongly tied to the Java programming language, as the most important WS-BPEL engines (e.g., Apache ODE[1] or the ActiveVOS engine[2]) are written in Java and work best in Java environments.

Windows Workflow Foundation (WWF)

Contrary to WS-BPEL, WWF is used exclusively in .NET environments, as it is deeply rooted in the .NET family of tools. WWF is a graphical language, which is compiled either directly to C#

[1]http://ode.apache.org/

[2]http://www.activevos.com/products/activevos/overview

code (up to .NET 3) or Extensible Application Markup Language (XAML) in more recent versions. In this way, WWF is much closer to a regular programming language than WS-BPEL, as one can execute C# code directly, trigger compositions directly from .NET applications and integrate compositions into larger regular applications. Since compositions are compiled to code, the composition engine can be efficient and simple. WWF compositions come in two flavors. Firstly, sequential workflows implement a block-structured language with features and activities very similar to WS-BPEL. Activities in such a composition are generally executed one after the other (with the exception of parallel blocks, which are of course possible), and activity transitions are triggered by finished activity executions. Secondly, WWF allows to defined so-called event-driven workflows. These are very different from sequential workflows, to the extent that one can argue that they implement an entirely different language. Event-driven workflows implement state machines. In every state, one or more activities are executing as long as no event is triggered, which changes the state. These types of workflows are rarely used to implement service compositions. Hence, in the remainder of this thesis, the term WWF composition always refers to sequential workflows. WWF compositions are usually developed graphically. An example of a service composition modelled using the graphical developer tool included in Microsoft Visual Studio is given in Figure 2.1. This example implements a small part of the case study presented in Chapter 3.

Note that sequential workflows offer a default feature set very similar to that of WS-BPEL (branching, looping, invoking Web services, throwing and handling faults, parallelism, receiving external input via Web service interfaces). However, WWF provides strong built-in support for easy extension, which most WS-BPEL tools are missing. For instance, it is very easy to create new custom activities, which allow for previously unsupported features, e.g., invocation of RESTful services. Furthermore, WWF provides an API for runtime changes of service compositions[3]. In this thesis work, both features have tremendously eased prototype implementation. However, as WWF sequential workflows and WS-BPEL are conceptually very similar, similar tooling could arguably also be build for WS-BPEL, even if significantly more implementation effort would be necessary.

2.2 Quality of Service (QoS)

Service descriptions include metadata about services. At least three different groups of metadata can be differentiated:

- *Functional descriptions* are the most common metadata, and define the functionality that a service provides. Simple functional descriptions can be published using WSDL [197]. More complex service descriptions are often specified using semantic Web service [120] technology, for instance Web Service Modeling Ontology (WSMO) [157] or Semantically Annotated WSDL (SAWSDL) [101]. However, there are also approaches which do not rely on semantics to provide more powerful functional descriptions [160].

[3]http://msdn.microsoft.com/en-us/library/ms734569(v=vs.85).aspx

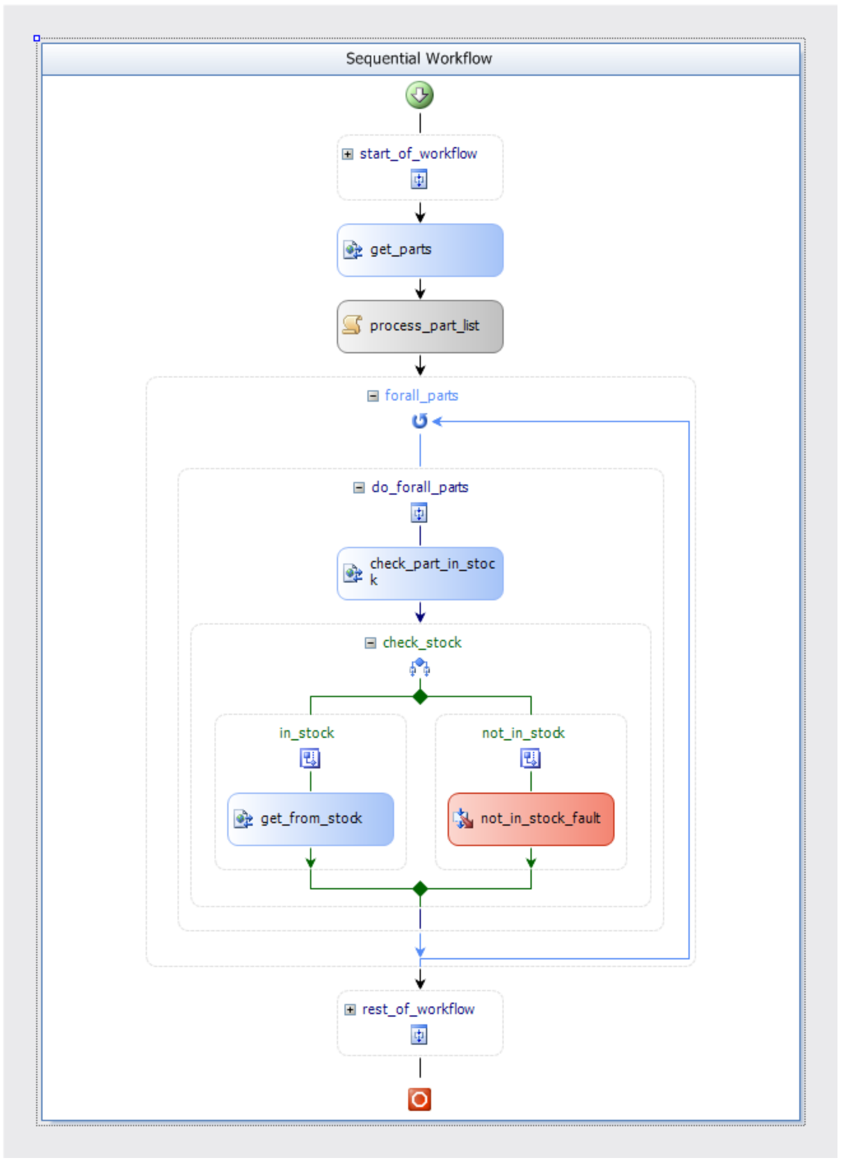


Figure 2.1: WWF Example Process

- *Protocol descriptions* cover the dynamic aspects of service description. These are only relevant for stateful Web services, where service invocations have to be issued following a defined protocol or Message Exchange Pattern (MEP). These "usage protocols" for stateful services can be specified using languages such as BPEL Light [115] or the Service Protocol Language (SEPL) [77].

- Finally, QoS [121] aspects denote the non-functional properties of services. Hence, QoS is often seen as a discriminator between functionally equivalent services.

The abstract concept of QoS can be measured in virtually infinite different dimensions. Often-used dimensions to measure QoS are availability, response time, failure rate (reliability) and security. However, other research papers have identified a plethora of additional metrics. For instance, in [162] the dimensions wrapping time (time to unwrap the request XML structure), execution time (time necessary for the actual service invocation, excluding networking and message serialization issues) and network latency are proposed. Others consider trust and reputation as qualities that a service can exhibit [168, 184], and that can be used to differentiate between services.

QoS Monitoring

One aspect of QoS research drawing considerable interest is QoS monitoring [128, 141, 162, 209]. Generally speaking, the research community has gravitated towards four standard ways of monitoring QoS:

- Server-side monitoring essentially assumes that the service itself reports QoS data, which is usually not reliable or trusted. Such monitoring approaches often need to be combined with other means to make sure that service providers do not have incentives to report inaccurate quality information [128].
- Client-side monitoring uses either test invocations or measures the invocation time of real invocations to find out about the service quality as experienced by the client [74, 162].
- Event-based monitoring is neither on client- nor server-side. Instead, event traces, as produced by the middleware, for instance the composition engine or an Enterprise Service Bus (ESB), are analyzed [12, 163, 186, 209].
- For more fuzzy or "human-based" dimensions (e.g., trust), feedback from previous service users is used [88, 108, 168]. This approach is related to client-side monitoring, with the main difference that in this case feedback from others is the basis of QoS calculation.

In this thesis, QoS is mainly used as the foundation of SLAs. Hence, QoS attributes which are often used to define SLOs are of particular interest. These include response time, availability and reliability. Monitoring of these QoS dimensions is generally done using an event-based approach, as discussed in Section 4.2.

2.3 Service Level Agreements (SLAs)

In a way, Web service SLAs [96] are a formalization and contractual arrangement of the concept of QoS. Instead of assuming that services provide the highest quality they can on a best-effort basis, SLAs fix the minimally promised quality in various dimensions. SLAs are often seen as legally binding contracts between one or more service clients and a service provider. SLAs are mainly a collection of SLOs. An SLO is an observable quality dimension of a service. Evidently, there is a strong relationship between SLOs and QoS, and, indeed, frequently SLOs are defined on top of QoS characteristics. For instance, often-used SLOs are the response time and availability of the service. Additionally, SLAs define penalties for non-achievement (violation) of SLOs. Penalties are often monetary consequences, which are expected to press service providers to achieve their target values. Both, penalties and target values, can be different for every SLA in which a an SLO is used. At runtime, concrete values for SLOs can be monitored. Based on the type of SLO (see below), this measured value can be generated either per composition instance or per aggregation interval. For clarity, the domain model of SLAs, as they are understood in this thesis, is depicted in Figure 2.2.

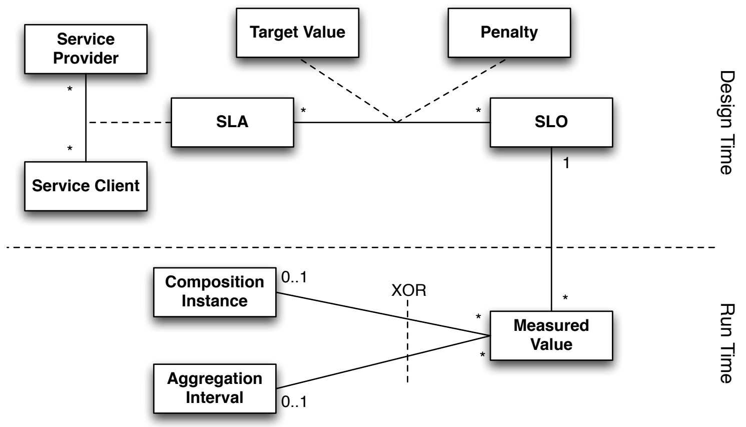


Figure 2.2: SLA Model

Some different languages have been proposed to model SLAs, including Web Service Level Agreement (WSLA) [39, 96], WS-Agreement [3] and SLAng [167]. These models do not differ so much in their expressiveness, but more in the environment they live in. For instance, WSLA specifies a monitoring and accounting infrastructure along with the basic language [39]. It is important to note that the work in this thesis is agnostic with regard to the used SLA language, as long as it fits the basic model specified in Figure 2.2.

Types of SLOs

Just like QoS dimensions, SLOs come in different flavors. In this thesis, two distinctions are of relevance.

- Firstly, one can differentiate between nominal and continuous SLOs. For nominal SLOs, the measured value of an objective can be one of a finite number of potential values. For these SLOs, the target value is a subset of the set of potential values. Metric SLOs, which are more prevalent, can take an infinite number of values. Target values are defined as thresholds on the metric. Examples of both types of SLOs are given as part of the case study in Chapter 3.

- Secondly, one can distinguish SLOs on composition instance level and aggregated SLOs. This was already hinted at in Figure 2.2. For composition instance level SLOs, a decision of whether an SLA violation has happened can be made for every single composition instance individually. Aggregated SLOs are defined over an aggregation period, for instance a number of composition instances or a time interval. Decisions can be made only looking at the whole aggregation period, i.e., usually numerous composition instances. Unless stated otherwise, all work in this thesis is applicable for composition instance level SLOs only. A generalization of the presented approach to aggregated SLOs is part of ongoing work.

SLA Monitoring

As SLAs are contractual agreements, monitoring of whether these contracts are fulfilled or violated becomes an essential feature. Evidently, QoS monitoring as described above is related to SLA monitoring, but given the financial aspect of SLA monitoring, both, pure client- and server-side monitoring, is not feasible. Instead, most practical solutions rely on a (presumably impartial) monitoring and accounting entity [96]. Some modern SOA infrastructures, such as VRESCo [129, 130], provide SLA monitoring of client-service interactions as built-in feature [128]. These systems use the event-based monitoring approach discussed above, usually integrated with the methods of Complex Event Processing (CEP) for event correlation.

In the remainder of this thesis, it is assumed that SLA monitoring is provided as a service by the used SOA infrastructure. As stated above, this is the case for VRESCo, which has been used as basis for the thesis prototype.

SLA Negotiation

So far, only the execution of an SLA has been discussed, not the inital step of even arriving at an agreement. This step is usually refered to as SLA negotiation [42, 201]. In essence, SLA negotiation is not very different from any other business negotiation. However, very often, and especially so in Grid computing scenarios, SLA negotiation is to a large degree an automatized process [201]. In automated SLA negotiation, SLOs, target values and penalties are agreed upon via predefined negotiation algorithms. Of course, this only shifts the negotiation problem to an earlier phase, where the automated negotation algorithm needs to be agreed upon (and different algorithms might favor different parties). Hence, research work on SLA meta-negotiation [21, 22] tackles the automated negotiation of negotiation protocols.

In this thesis the problem of SLA negotiation is excluded. It is assumed that service provider and service clients have already established concrete agreements, which are accepted by all parties. This has the advantage that the approach of this thesis is not bound to any concrete way of SLA negotiation and management approach. For instance, the approach of this thesis is also applicable if no SLA negotiation in the technical sense has happened, and SLAs are simply legal documents negotiated by human lawyers.

2.4 Autonomic Computing

The vision of autonomic computing [97] is that systems are able to manage themselves autonomically, that is, without a human operator instructing the system directly. Human operators interact with the system through high-level policies, which focus more on the goals of the system (what states should the system be in?) and less on the way to reach these goals (what should be done in which states?). In this regard, autonomic computing is similar to the idea of self-adaptive systems [164], even if autonomic computing generally has a less broad scope than general self-adaptiveness [150, 164].

Self-* Properties

In literature, a large number of different properties of autonomic systems have been proposed (often referred to as the self-* properties of autononomic computing). The ones which were included in IBM's original autonomic computing initiative were self-configuration (systems adapt autonomically to changing environments), self-healing (failing systems are able to autonomically return into a non-failing state), self-optimization (systems monitor and adapt their parametrization autonomously, to improve performance in operator-defined dimensions) and self-protection (systems are able to anticipate and protect from attacks autonomously) [61]. In this thesis, most focus is put on self-optimization. The main parameter that is optimized is the total costs for the service provider. Other existing research work discusses self-healing [4, 11, 67] in service-based environments. Self-configuration and self-protection are less frequently discussed in this area, even though some research results from distributed components [35] may be applicable for service-based systems as well.

The MAPE Loop of Autonomic Computing

Self-* properties are generally thought to be achieved using a four-step feedback loop [97] (see Figure 2.3):

1. Firstly, the *managed element*, i.e., the system that is to be managed autonomically, is *monitored*. Monitoring is implemented using *sensors*, whose main task is to sense and report the current state of the managed element. Usually, no processing as such happens in the sensors themselves, even though the monitoring component is sometimes thought to be able to handle simple transformations of the sensed data (for instance, simple aggregation).

2. Secondly, the sensed data is processed, interpreted and *analyzed*. In this step, the raw data produced by the monitoring step is made sense of, enriched with context and other necessary additional information, and brought into a format that the third component, the planner, understands. The concrete semantics of the planning step are case-dependent. In the context of this thesis, predictions of SLA violations are generated using machine learning techniques.

3. Thirdly, the analyzed data is used for action *planning*. In this step, a concrete plan is constructed how the sensed state can be transformed into one that is more desireable. Oftentimes, planning is implemented using techniques from the areas of artifical intelligence [190] or optimization. In this thesis combinatorial optimization [143] of different plans is used to implement decision making in the planner.

4. Finally, the constructed plan needs to be *executed*. Oftentimes, *actuators* are used to actually re-configure or adapt the managed element. However, as part of this thesis, no explicit actuators have been utilized. Instead, adaptation has been implemented mostly based on the means provided by the VRESCo SOA infrastructure and the used WWF workflow engine.

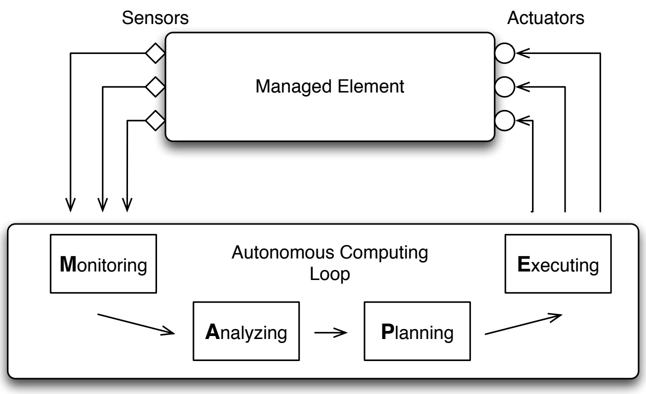


Figure 2.3: MAPE Loop (Adapted from [97])

These four steps are generally referred to as the MAPE (**M**onitoring, **A**nalysis, **P**lanning and **E**xecution) loop of autonomic computing. Chapter 4, which describes the research contribution of this thesis, follows the structure of the MAPE loop and explains how all four steps have been implemented.

2.5 Machine Learning

Machine learning is a branch of the wider area of artifical intelligence research. Generally speaking, machine learning deals with automated learning of relationships or behavior from data. Machine learning is often applied to the following three groups of problems:

- *Classification* is the problem of assigning a data item to one of a number of classes. The classes are known in advance, and the number of classes is finite. Esimating the concrete value of a nominal SLO value in a running composition instance is in essence a classification problem.
- *Clustering* is related to classification, in that data items are assigned classes. However, unlike in classification, the classes are unknown in advance, and are also learned. Nevertheless, the number of classes is still finite.
- This is not the case for *regression*. Regression does not produce an assignment of a class as output, but a numerical value on a continuous scale. Hence, the number of different results that regression can produce is infinite. Estimating the value for a continuous SLO in advance is a regression problem.

Machine learning comprises various different approaches, which solve different problems, use different algorithms and which are based on different preconditions. The most important of these approaches are ANNs [72, 86] (mostly used for regression), decision trees [151] (mostly classification), Support Vector Machines (SVMs) [37] (both classification and regression), Bayes networks [60] (classification) and cluster analysis [94] (clustering). The WEKA machine learning toolkit[4] implements a collection of popular and well-known algorithms for all of these approaches. WEKA is open source software and can easily be integrated into Java applications using an Application Programming Interface (API). In the thesis prototype, WEKA has been used to implement machine learning, i.e., no novel machine learning algorithms as such have been devised as part of this thesis. Instead, the decision was to not re-invent the wheel and focus on the application of exising, battle-proven algorithms to novel problems (SLA management).

In the following, some more details on ANNs and decision trees are presented, since these approaches form the basis of the prediction approach for continuous and nominal SLOs respectively.

Artificial Neural Network (ANN)

ANNs are a machine learning technique inspired by the inner workings of the human brain. Therefore, ANNs belong to the larger group of bio-inspired computing techniques [57]. Basically, ANNs consist of one or more layers of nodes (neurons) and directed connections between neurons. Oftentimes, ANNs consist of an input layer, output layer and one or many intermediary layers. In every layer, every node is connected with all nodes in the next layer. This ANN architecture is called feed-forward multilayer perceptron [72], an example with one intermediate layer and three neutrons per layer is visualized in Figure 2.4.

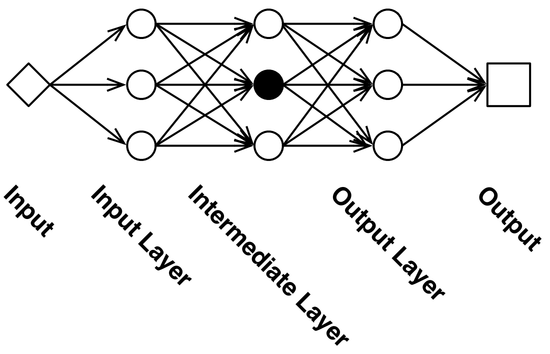


Figure 2.4: Example Feed-Forward Multilayer Perceptron

In this graph model, nodes are processors which typically sum all input they receive from nodes from the previous layer weighted using the edge weights, add some activiation bias, and run the result of this sum through an activitation function. The activation function decides if the node should fire, based on the input received, the weights of these inputs and the activation bias. The activation function is an indicator function, i.e., the output is always either 0 (not fired) or

[4]http://www.cs.waikato.ac.nz/ml/weka/

1 (fired). This neuron model is visualized in Figure 2.5 for the black neuron in Figure 2.4. In Figure 2.5, $w(k)$ are edge weights, $b(k)$ is the activation bias, and ϕ is the activation function.

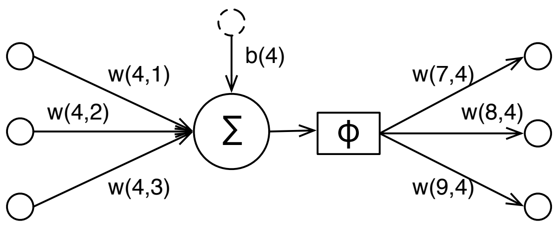


Figure 2.5: Activation of a Neuron (Adapted from [72])

In mathematical terms, the output of a neuron can be described as in Equation 2.1, with y_k being the output of the neuron, and x_1 to x_m the output of the neurons of the previous layer.

$$y_k = \phi(\sum_{j=1}^{m}(w(k,j)x_j) + b(k)) \tag{2.1}$$

In this model, the node weights $w(k, j)$ and the activation bias $b(k)$ are central. Essentially, all knowledge learned by the ANN is contained in these weights. Evidently, weights need to be trained prior to using an ANN. Mostly, training of multilayer perceptrons is done using input-output examples [72], i.e., using exemplary data, where the outcomes of learning data is already known. Training starts with random weights, and is continued in many iterations until a set of weights is found which maximizes the correlation between the outcome of the ANN and the actual outcome of the examples. Different algorithms are used to adapt the weights in every iteration, for instance back propagation [86]. Multilayer perceptrons are popular because they can (approximately) represent any relationship between input data and outcome (unlike simpler neural network techniques such as the perceptron, which cannot distinguish data that is not separable by a hyperplane [72]).

In this thesis, feed-forward multilayer perceptrons are used for regression, as necessary for the prediction of continuous SLOs. Note that the usage of ANNs was a design decision. In principle, any other regression algorithm (for instance SVMs) could be used as well. Indeed, switching to using SVM or any other regression algorithm implemented in the WEKA framework is just a configuration-level step in the PREvent prototype.

Decision Trees

Decision trees [151] are an approach primarily used for classification of data, even though some work on extending trees to regression problems exist [114]. The decision tree is a directed, acyclic graph with one root node and any number of leave nodes (nodes with an out-degree of 0). Every leaf node represents a classification to one of the classes, every other node represents a decision. The decision has as many possible outcomes as the out-degree of the decision node, since every out-edge represents one outcome of the decision. If data has to be classified using the decision tree, one just reads the tree from the root node downwards, always continuing with the edge indicated by the outcome of every decision evaluated against the data, until a leaf is reached. This leaf is the result of the classification. A primitive example of a decision tree is depicted in Figure 2.6. In the figure, only two decision nodes exist (*a?* and *b?*), and two possible classifications (*x* and *y*).

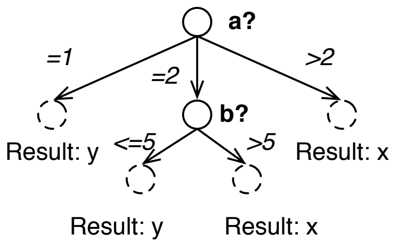


Figure 2.6: Decision Tree

Depending on the concrete variation of decision trees, different algorithms are used to learn trees from data. The most important decision tree algorithm is probably C4.5 [83, 152]. C4.5 uses a divide-and-conquer algorithm to construct trees. The basic idea is to add decision nodes to the tree until every leaf node contains only instances of one class. Decision nodes are selected based on the "purity" [192] of the split they provide, i.e., decisions, which separate the instances more clearly, are preferred. In every step, the decision that improves the purity the most (i.e., which provide the largest information gain) is selected. The metric to measure the purity of a node is the entropy, defined as in Equation 2.2 (from [192]), with n being the number of possible outcomes of the decision, and p_i being a fraction representing the number of instances with outcome i divided by the number of total instances in this node. $p_1, p_2, \ldots, p_n$ sum up to 1. The entropy is in $[0 : 1]$, with higher values being preferred.

$$entropy(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} -p_i \log_2 p_i \tag{2.2}$$

As stated above, this construction algorithm is in principle used until all data is correctly classified. However, oftentimes stopping earlier is better, to prevent overfitting and to produce smaller, more readable trees. This is called the minimum description length principle [153]. The process of either stopping the decision tree construction early or throwing away part of an already constructed tree is called pruning [131]. In C4.5, a process called subtree raising is used for pruning [152]. Further discussion of this process is out of scope here.

In this thesis, decision trees have been used twofold. Firstly, decision trees form the basis for prediction of nominal SLOs. In this sense, decision trees are used as an ANN counterpart for classification. In this function, decision trees can also easily be exchanged for any other classification algorithm. Secondly, decision trees have also been used to implement dependency analysis (see Section 4.2). Here, the tree representaion of knowledge inherent to decision trees is one of the core features of the approach, hence decision trees cannot be easily exchanged for another classification approach.

CHAPTER 3

Case Study

In the remainder of this thesis, the case of a reseller of built-on-demand heavy-duty industry robots (ACMEBOT) will be used. The business model of ACMEBOT is as follows. Customers request a quote (request for quotation, RFQ) for a specific robot in a specific configuration. ACMEBOT plans the steps necessary for the requested configuration, checks if necessary parts are not in the internal warehouse (these parts need to be ordered from external suppliers), and generates an offer. The customer can then either cancel the process or place the order. If the order is placed, ACMEBOT starts by getting all missing parts from external suppliers and waits for these parts to arrive. As soon as all parts are available, the product is scheduled for assembling. After assembling is finished, the product is subject to a quality control step, and, if successful, shipped to the customer. In parallel to the shipping of the product, the customer is billed and an invoice is sent. ACMEBOT's core business process is depicted in Figure 3.1 (in BPMN [1] notation).

For reasons of brevity, this thesis concentrates on the two roles "Customer" and "ACMEBOT Assembly Service" in the figure, even though ACMEBOT interacts with many different external partners (e.g., suppliers of parts, shippers, credit card companies) to implement the described functionality. ACMEBOT's IT heavily uses the notion of SOA, hence, the order process is implemented as a service composition, i.e., activities in the process are mapped to one or more invocations of (Web) services.

Generally, customers of ACMEBOT are larger business partners, typically resellers (i.e., ACMEBOT does not interact with end customers directly). For each reseller, there are pre-established SLAs, which typically consist of a subset of a number of often-seen SLOs. Table 3.1 provides a (non-comprehensive) list of these common SLOs for this process. Note that these can be of quantitative (SLOs #1 to #4) or of qualitative (SLO #5) nature.

All SLOs are associated with target values and penalties for violating these targets, which are both SLA-specific (i.e., even if the same objective is used in many SLAs, the target value does not need to be the same each time). In any case, ACMEBOT has a strong interest in complying to these SLOs, as long as the costs of doing so do not exceed the benefit. The manufacturer may apply a number of runtime adaptations to the process. Some example adaptation actions

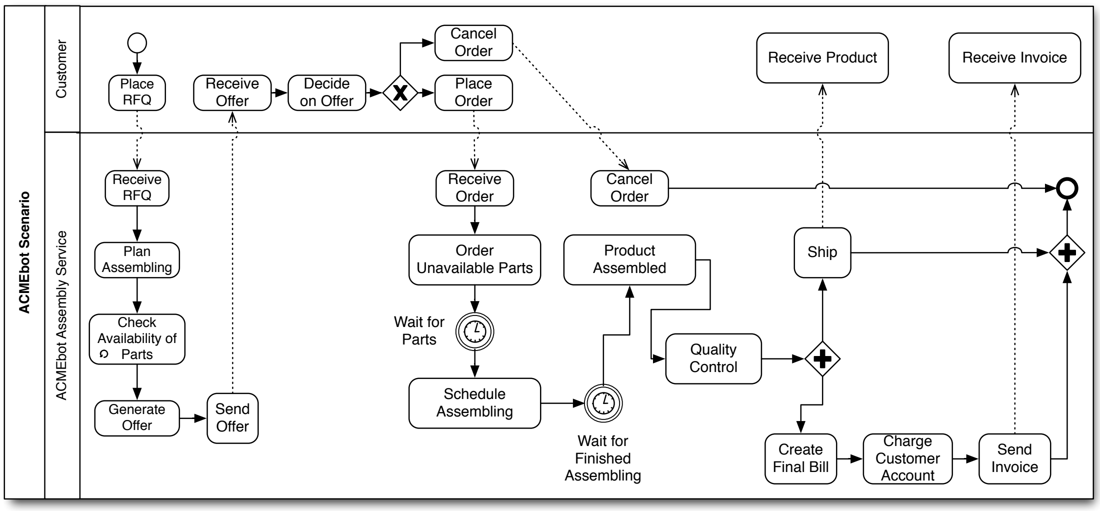


Figure 3.1: Motivating Scenario

are sketched in Table 3.2. The columns **+** and **-** refer to SLOs in Table 3.1, and indicate that the respective action has a positive (**+**) or negative (**-**) impact on this SLO. Note that these actions and impacts are just of exemplary nature, that is, while for some business cases outsourcing may reduce costs and increase the process duration (and error rate), this does not necessarily hold for all processes. Additionally, applying these actions generally also has some associated costs, which need to be taken into account (for instance, express shipping is more expensive than regular shipping). As one can see, for the manufacturer, there is a tradeoff between the three

#	SLO Name	Description	Target	Penalty
1	**Time to Offer**	Time between receiving the RFQ and responding with an offer (in working days).	$<= 2$	Implicit costs - customer will choose a different manufacturer if offer is not received in time.
2	**Order Fulfillment Time**	Time between receiving the order and finishing the process (in working days).	$<= 5$	Manufacturer grants 5% discount per 1 day delay, 20% max discount, not additive with SLO#3.
3	**Process Lead Time**	Time between initializing the process and finishing it (excluding activities at customer side) in working days.	$<= 6$	Manufacturer grants 5% discount per 1 day delay, 20% max discount, not additive with SLO#2.
4	**Cost Compliance**	Cost overrun with regard to the offer in % of the offer.	$<= 5$	Manufacturer cannot charge more than the offer plus 5%.
5	**Product as Specified**	Product is exactly as specified.	n/a	If wrong product is delivered, manufacturer needs to produce and ship the specified product within 7 working days and grant a 5% discount.

Table 3.1: Service Level Objectives

dimensions duration, costs and quality, which is well-known in many fields of engineering.

As the manufacturer business process is implemented as a service composition, applying these adaptations essentially boils down to adapting the service composition. This can be done by either adapting the data flow of the composition (Action #1 and Action #4), by rebinding activities to different services (Action #2), or by applying more or less complex adaptations to the structure of the composition itself (all others).

#	Adaptation Action	Implemented As	+	-
1	Use faster shipping option, e.g., express shipping.	Change the `shipping option` parameter in invocation of shipping service	#2, #3	#4
2	Use external supplier who delivers faster (but is presumably more expensive)	Bind to different supplier service in the `Order Unavailable Parts` activity	#2, #3	#4
3	Order more parts instead of producing them in-house.	Change structure of composition to order parts that are actually not unavailable	#2, #3	#4
4	Generate offer with higher priority.	Change the `priority` parameter in invocation of the offer service	#1, #3	-
5	Outsource assembling and quality control.	Change structure of composition to reflect outsourcing	#4	#2, #3, #5
6	Skip quality assurance	Skip quality assurance activity in the composition	#2, #3, #4	#5
7	Add an additional quality assurance step.	Modify composition structure and add a new activity	#5	#2, #3, #4

Table 3.2: Possible Adaptation Actions

In the remainder of this thesis, it will be shown (1) how violations of SLOs such as the ones sketched in Table 3.1 can be predicted at runtime, (2) how the most promising adaptations from a set of potential actions such as the ones in Table 3.2 can be selected, and (3) how these adaptations can be executed in practical service composition scenarios. The ACMEBOT case study will be used as illustrative example in the following discussions, and will form the basis of the experimental results that are presented in Chapter 5.

CHAPTER 4

Cost-Based Optimization and Adaptation of Composite Services

In this chapter, the main contributions of this thesis are presented. The descriptions in this chapter use the case study discussed in Chapter 3 for practically explaining ideas. Structurally, the chapter closely follows the well-known autonomic computing loop [97] of *Monitoring*, *Analysis* (referred to as *Prediction* in this thesis), *Planning* (*Cost-Based Optimization* in this thesis) and *Execution* (*Adaptation*). After presenting a high-level overview, a detailled look on all research that has been carried out for each of the MAPE components is given.

4.1 Overview

All research work carried out in this thesis has been done within the PREvent framework. Generally, PREvent is a closed loop system [164] for self-optimizing service compositions. PREvent is based on the SOA runtime environment VRESCo [129, 130]. VRESCo is a .NET[1] based prototype, which provides a registry for Web services metadata and QoS information (including QoS monitoring), as well as base services for dynamic service compositions. Even more important in the context of this thesis is the strong support for eventing in VRESCo, as discussed in [127]. The most current public version of VRESCo is available online[2]. Furthermore, the interested reader is invited to visit the VRESCo project web page[3].

The overall architecture of PREvent is sketched in Figure 4.1. As can be seen in the middle tier of the figure (Prevention View), there are four different phases of adaptation.

1. The first phase includes the monitoring of runtime data, for which the *Composition Monitor* component is responsible.

[1] http://www.microsoft.com/net/
[2] http://sourceforge.net/projects/vresco/
[3] http://www.infosys.tuwien.ac.at/prototype/VRESCo/

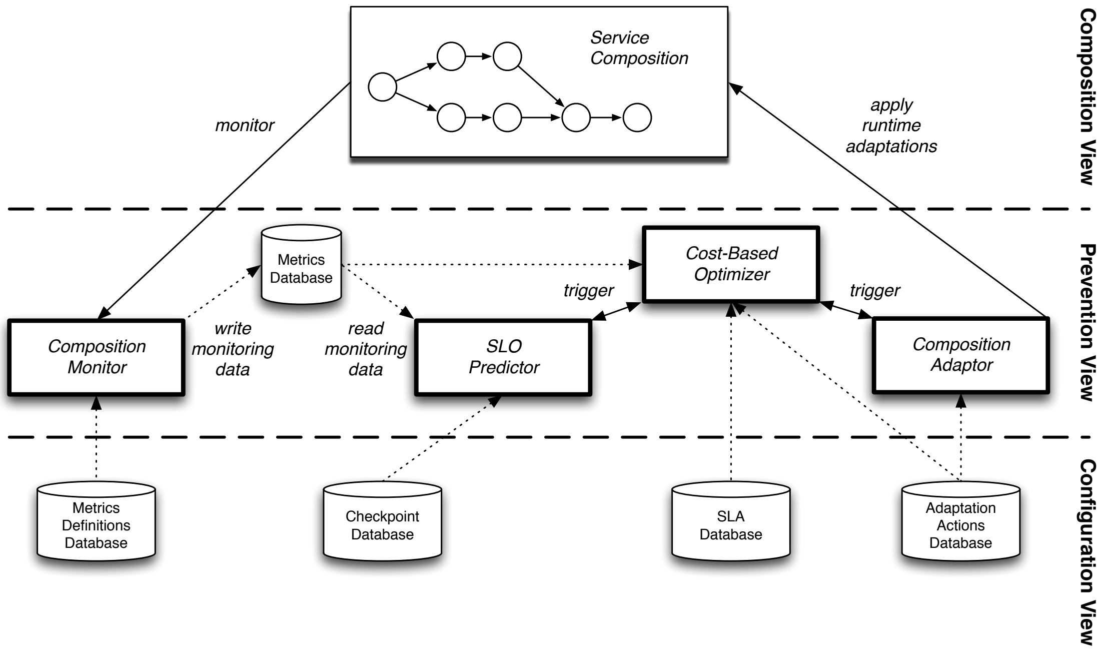


Figure 4.1: Overall Architecture of the PREvent Framework

2. The second phase comprises the prediction of violations, which is handled by the *SLO Predictor* component.

3. The third phase is the selection of the most promising adaptations from a candidate set of possible actions using the *Cost-Based Optimizer* component. Cost-based optimization selects the actions which minimize the total costs for the service provider, i.e., the sum of the costs of applying all actions plus the remaining penalty costs.

4. The fourth phase comprises the application of the actions selected by cost-based optimization, carried out by the *Composition Adaptor* component.

Composition Monitor and SLO Predictor are connected via the *Metrics Database*, which stores all collected runtime data. The Composition Monitor is configured using the *Metrics Definitions Database*. The SLO Predictor needs access to the *Checkpoint Database*, which specifies at which points in the execution of the composition prediction, optimization and adaptation should take place. The Cost-Based Optimizer needs the information contained in the *Adaptation Actions Database* (a list of all available adaptation actions) and *SLA Database*, as well as access to the metrics database, in order to find the most suitable actions to apply. Obviously, the Composition Adaptor component needs access to the Adaptation Actions Database as well. All information stored in the bottom tier of the figure (Configuration View), as well as the service composition itself (Composition View), are considered inputs to the approach. Specifically, the process of SLA negotiation [42] or meta-negotiation [21] is not covered explicitly in this thesis. The same is true for identification of possible adaptation actions and modelling the impact of adaptations. Generally, human domain experts are more than able to derive these actions using domain knowledge and modern tooling, e.g., Business Intelligence (BI) techniques.

4.2 Monitoring of Service Compositions

The Composition Monitor component is responsible for collecting all runtime information necessary for the prediction of SLA violations. In the MAPE loop, the Composition Monitor evidently implements the **M**onitoring phase. It takes a list of metrics to monitor as input, and writes all monitoring data to the Metrics Database.

Types of Metrics

Generally two different types of metrics are monitorable: *composition metrics* are defined based on one or a series of lifecycle events of the service composition, as produced by the VRESCo event engine (e.g., shipping activity has started, shipping activity has ended); *external metrics* contain data which stem from outside of the composition (e.g., the order date). The latter allow to seamlessly integrate any kind of external information, which may influence SLA conformance.

For composition metrics, this thesis uses the terminology and distinction established by [188]. In this terminology, two types of composition metrics are distinguished. One the one hand, metrics can be typical QoS metrics, such as the response times of Web services. On the other hand, and more interestingly, metrics can also be Process Performance Metrics (PPMs). PPMs are domain-specific metrics, which are often defined recursively, and which make business sense in the domain of the service composition. PPMs are often the basis for defining the Key Performance Indicators (KPIs) and SLOs of a service composition or business process. Examples of PPMs include simple metrics, such as the customer that has ordered a product, or the number of parts that are not in stock for a given order. The SLOs described in Table 3.1 are further examples of PPMs

At any point in the execution of a service composition, every QoS and PPM metric can be in one of three distinct states:

- Metrics can be available as **facts**. Facts represent data which is already known at this time. Generally speaking, this includes all metrics which can already be monitored at this point in the execution. Typical examples of facts are the QoS of services that have already been invoked.

- Metrics can be **unknowns**. Unknowns are the logical opposites of facts, in that they represent data which is entirely unknown at this point in the composition instance. Oftentimes, PPMs related to parts of the composition that have not yet been executed fall into this category. For instance, before starting the assembling process, the assembling time will be unknown.

- As a kind of middle ground between facts and unknowns, metrics can also be **estimates**. Estimates represent data which is not yet available, but can be estimated in some way. This is often the case for QoS data, since techniques such as QoS monitoring [128] can be used to create an approximation of e.g., the response time of a service before it is actually invoked. As PPMs are generally domain-specific, it is often harder to estimate PPMs in advance. However, in many cases, this is possible as well. For instance, given the address of the customer and the selected shipping option, it is possible for domain experts to estimate the shipping time before even starting the shipping process.

For PREvent, facts and estimates are particularly relevant, as they form the input that is used to generate predictions of SLA violations.

Event-Based Monitoring in PREvent

The general PREvent monitoring approach is sketched in Figure 4.2. External metrics (as opposed to composition metrics) are provided by an external data provider, e.g., a Web service or an external database. For these metrics, the Composition Monitor does little more than retrieve the value from the external provider. Specifically, we envision that our approach can be easily integrated with the current trend towards providing external data as services [78]. Monitoring of composition metrics is based on means provided by the VRESCo system, specifically by the VRESCo event engine [127]. This engine implements CEP for service-based systems, which, besides other features, includes means to register for all kinds of lifecycle events, such as events that indicate that a service has been bound, invoked or failed. As event payload, these events contain all information necessary to monitor even domain-specific PPMs. Further information on the VRESCo event model can be found in [126, 127].

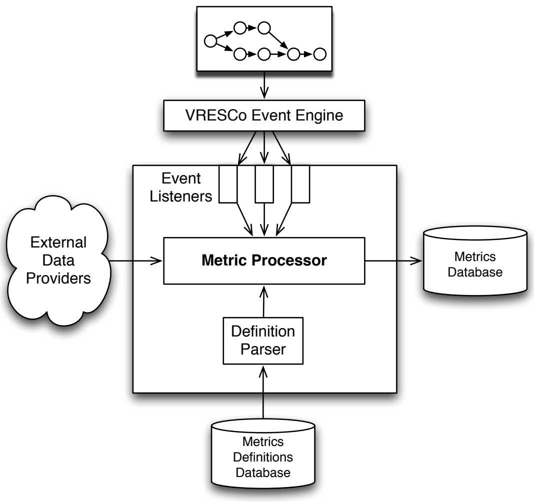


Figure 4.2: Composition Monitoring

The core of PREvents Composition Monitor is the *Metric Processor*. On startup, the processor gets a parsed list of all metrics to monitor and their definitions from the *Definition Parser* component (which retrives this list from the Metrics Definitions Database). Metric definitions are provided in an XML-based format. For every composition metric, the processor registers exactly one event listener in the VRESCo Event Engine.

Metric Definition

Metrics are specified in XML as a mandatory event query, an optional message property path, an optional postprocessing script, and finally an optional aggregation function. From these definitions, the metric processor can calculate metric values using the following procedure:

1. Firstly, metric calculation is triggered whenever a new event is received, which matches the event query. This event is the basis for metric calculation. For instance, for the PPM metric "Payment Preference of Customer", a listener is generated, which delivers the event issued after the activity "Get Payment Preferences" is finished.

2. As payload, this event contains the necessary data. The Metric Processor retrieves the raw metric value from a given message property path (e.g., `GetPaymentPrefsResult/Preference`). The message property path tells PREvent how to extract information from the event body.

3. Next, a postprocessing script is executed, if one is specified. Postprocessing scripts are specified using the CSScript[4] scripting language directly within the XML definition of the metric.

4. Finally, if the CEP event stream returns more than one event, aggregation functions (average, sum, minimum, maximum, count) are used to indicate how to aggregate multiple metric values into a single final monitoring result. For instance, if the event query relates to the invocation time of a service, and the service is invoked multiple times in the composition, the `avg` function can be used to calculate the average invocation time.

```
<fact name="average_exec_time_order_externally"
  type="integer"
  eql=
    "select
       _before.WorkflowId as id,
       (_after.WFTimestamp - _before.WFTimestamp) as data
     from
       BeforeWFInvokeEvent _before,
       AfterWFInvokeEvent _after
     where
       _before.WorkflowId = _after.WorkflowId
       and _before.ActivityName = 'order_externally'
       and _after.ActivityName = 'order_externally'
       and _after.BeforeReferer=_before.WFId"
  aggregation="avg"/>
```

Listing 4.1: Aggregated Metric Definition

Listing 4.1 gives a sample metric definition, where the PPM "average_exec_time_order_externally" is defined as the arithmetic mean time between the begin and end of the activity "order_externally". As this activity is executed multiple times in the service composition, the CEP

[4] http://www.csscript.net/

event stream defined by the query (in "eql") may contain many events, which are aggregated using the aggregation function "avg". Event streams are defined using the Esper Query Language (EQL[5], nowadays referred to as EPL, the Esper Processing Language). The final result of the calculation is of type integer.

```
<fact name="total_nr_of_items"
  type="integer"
  eql=
    "select
       _event as msg, _event.WorkflowId as id
     from
       AfterWFInvokeEvent _event
     where
       _event.ActivityName = 'get_parts_list'"
  messagePath="GetPartListResult/Parts"
  script="return (input as string).Split(';').Length;"
/>
```

Listing 4.2: Metric With Postprocessing

An example of a metric with postprocessing is given in Listing 4.2. The list of items is accessible in the event returned by the EQL query. The needed information can be retrieved for the event property "GetPartListResult" and, recursively, the property "Parts". There, the part list is stored as a simple string (a semicolon-separated list), so a simple CSScript (basically plain C# code) is applied to calculate the number of items in the list.

Dependency Analysis

The main interface of the Composition Monitor to the SLO Predictor is the Metrics Database. Whenever the metric processor measures a metric at runtime (both composition and external metrics), it saves the value to the Metrics Database (identified by the identifier of the current composition instance), where the value can be looked up by the SLO Predictor. However, the monitoring data collected can also be used for other purposes. One example usage of runtime monitoring data is to statically analyze the performance of service compositions using Dependency Analysis, which is briefly presented here for illustrative purposes.

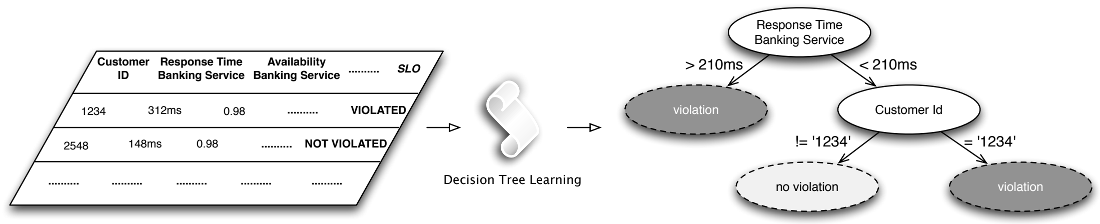


Figure 4.3: Dependency Analysis Based on Monitored Metrics

Dependency analysis uses historical composition data to determine the most important factors that dictate whether a process instance is going to violate its SLAs. The output of dependency analysis is an easily visualizable model of the internal dependencies of the composition.

[5] http://esper.sourceforge.net/esper-0.7.5/doc/reference/en/html/EQL.html

It identifies the most important factors of influence of process performance. In this thesis, decision trees [192] have been used as dependency model. These tree models are referred to as *dependency trees*, because they represent the main dependencies of the business process on QoS and PPM metrics, i.e., the metrics which contribute "most often" to the failure or success of a composition instance in respect to its SLAs.

The general approach of Dependency Analysis is shown in Figure 4.3. The main motivation for using decision trees is that they are easy to depict graphically. This is important if results should be presented to non-experts. An additional benefit is that many well-researched algorithms to construct decision trees from data exist, such as the C4.5 [83, 152] or the alternate decision tree, ADTree [59], algorithms. For tree learning, 10-fold cross validation is used, which is a standard technique in data mining to avoid having to split the input data into training, test and validation sets. Additionally, cross validation allows to estimate the classification error of the decision tree. The classification error enables the business analyst to measure the quality of the dependency tree, i.e., how exact the tree represents the actual structure of real-world dependencies.

4.3 Prediction of Service Level Objectives

In the PREvent approach, the SLO Predictor is the main consumer of the monitoring data collected by the Composition Monitor. In the MAPE loop, the SLO Predictor implements the **A**nalysis phase, in the sense that the SLO Predictor takes the collected metrics and uses machine learning algorithms to analyse the raw data for evidence of upcoming SLO violations in running composition instances. In this section, the core ideas of the machine learning based prediction approach used in PREvent are presented. Generally, this approach is based on the idea of predicting concrete SLO values based on whatever monitoring information is already available at a concrete point in the execution of a composite service. This point in the execution of the composition is referred to as checkpoint. Predicted SLO values can then be compared against the target values as specified in the SLAs, and, hence, violations can be forecast before they have actually happened.

Prediction Approach Overview

The overall architecture of the prediction approach is depicted in Figure 4.4. The most important concept used is that the user defines checkpoints in the service composition, which indicate points in the execution where a prediction should be carried out (indicated by a gray activity in the figure). For every SLO that needs to be predicted, every checkpoint is associated with exactly one Prediction Model (but the same model can be used if the same SLO is used in multiple SLAs). The Prediction Model uses a function, taking as input all facts which are already available, and, if applicable, a number of estimates of not yet known facts, and produces an estimation of the SLO value as output. Facts are retrieved from the Metrics Database. Checkpoint Predictors are generated based on the definition of checkpoints, as stored in the Checkpoint Database. Estimators are separate, even if usually small, software entities, which provide estimations for missing metrics. As it is not advisable to instantiate new Estimators for every prediction, the Estimators are managed within an Estimator Pool, and can hence be shared between different predictions and different Checkpoint Predictors. Management of this Estimator Pool (e.g., instantiating new Estimators, removing unused Estimators, instantiating more instances of overloaded Estimators) is one of the tasks of the Predictor Manager. The other task of this component is the monitoring of Checkpoint Predictors. Monitoring includes storing predictions of SLOs and comparing them with the respective actual results, in order to evaluate the performance of predictions. If necessary, the Predicor Manager re-trains underperforming Checkpoint Predictors or Prediction Models. In a PREvent system, there is only one Predictor Manager, which manages many Checkpoint Predictors. Finally, predictions are forwarded to a Prediction Graphical User Interface (GUI) for displaying to a human user. Additionally, predictions are sent to the Cost-Based Optimizer component, which has the task of deciding whether to apply adaptation actions to the running instance.

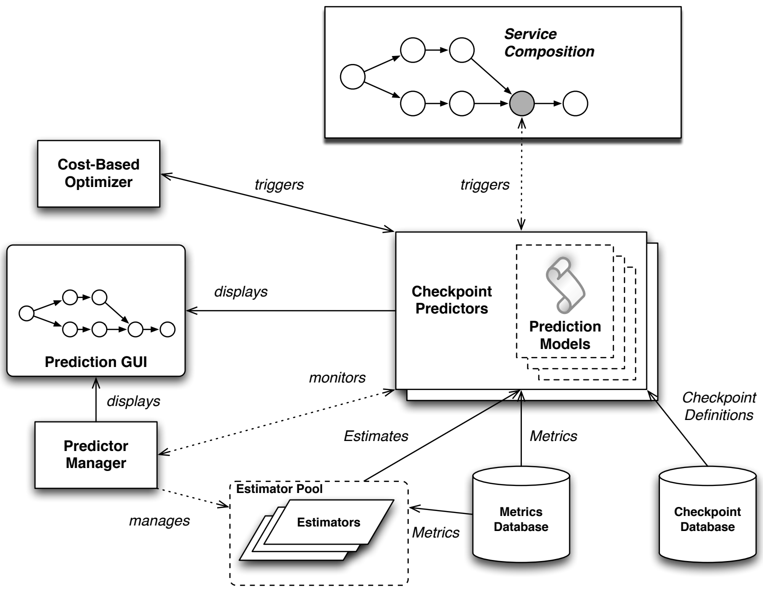


Figure 4.4: Prediction Architecture Overview

In PREvent, the prediction function used in the Prediction Models is generated using machine learning techniques. For continuous SLOs (e.g., order fulfillment time in the ACMEBOT case), multi-layer neural networks [72]) are used as prediction function. For nominal SLOs (e.g., product as specified) C4.5 decision trees [152] are used. This is similar to earlier work presented in [186]. In both cases, these machine learning functions are trained from existing monitoring information about previous runs of the composition. Therefore, PREvent needs to be bootstrapped with a reasonably large number of recorded historic executions. The amount of data necessary is case-specific, since it vastly depends on the complexity of the composition. We generally use the Training Data Correlation as a metric for evaluating the quality of a freshly trained model (see below for a definition), however, a detailed discussion of this is out of scope of this thesis.

Note that, in principle, it would also be possible to use prediction models, which predict violations directly (instead of SLO values). This approach has not been followed in this thesis for two reasons. Firstly, in the larger context of the PREvent framework, the predicted SLO values have significance, as they are directly used as input for cost-based optimization (see Section 4.4). Secondly, the PREvent framework is open for various types of penalty functions, which will be discussed in more detail below. For many penalty functions, the SLO value is relevant for determining the penalty that the provider has to pay, i.e., the SLO value indicates not only whether a violation is likely to occur, but also how expensive this violation is going to be.

Checkpoint Definition

At design-time, one core issue for using PREvent is the definition of suitable checkpoints in the composition model. For every checkpoint, the following input needs to be provided.

1. The hook, which defines the concrete point in the execution that triggers the prediction.
2. A list of available facts from the metrics database.
3. A list of estimates, and the estimator component as well as the parameters used to retrieve or calculate them.
4. The retraining strategy, which governs at which times a rebuilding of the prediction model should be triggered by the Predictor Manager.
5. Optionally, some parameterizations of the machine learning techniques used to build the prediction models for all SLOs.

With these inputs, the checkpoint can be deployed using the predictor manager, and an initial model is built from previous monitoring data. If no or too little historical data is available, the checkpoint is suspended by the predictor manager until enough training data have been collected. After the initial model is built, the continuous optimization of the predictor is governed by the predictor manager, according to the retraining strategy. In the following, these essential inputs are discussed in more depth.

Hooks

Hooks can be inserted either before or after any activity in the service composition (typically, a Web service invocation). Generally, there is a tradeoff to take into account here, as early predictions are usually more helpful (in that they usually allow for more adaptation actions if violations are predicted), but also less accurate, as less metrics are available. In this thesis, this tradeoff is referred to as Quality / Timeliness Tradeoff.

Figure 4.5 depicts a (simplified) example based on the ACMEBOT case, and shows two possible checkpoints. In C_1, the only facts available are the ones given as input to the composition (such as a customer identifier, or the ordered product). Some other facts (mainly QoS metrics) can already be estimated, however, other important information is simply not known

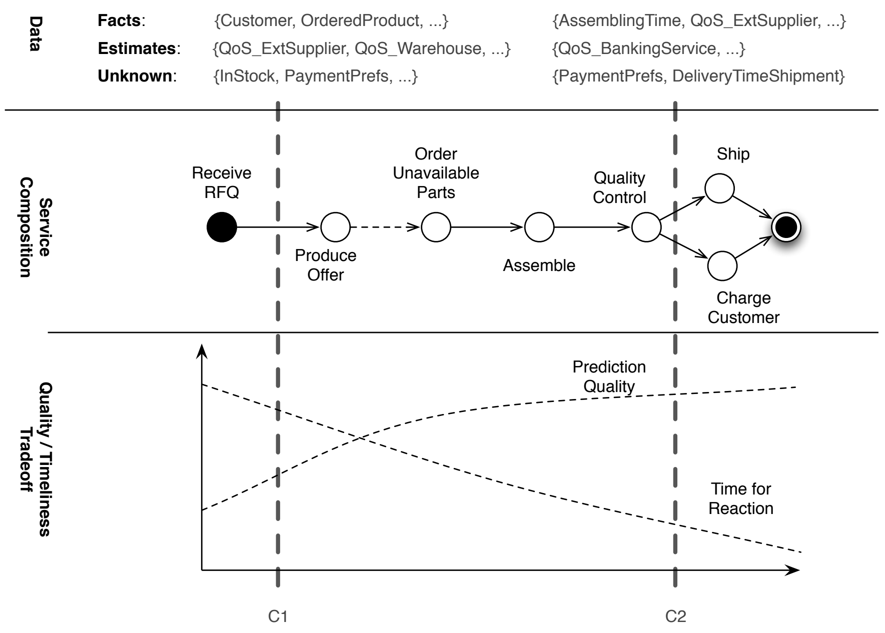

Data
Facts: {Customer, OrderedProduct, ...} {AssemblingTime, QoS_ExtSupplier, ...}
Estimates: {QoS_ExtSupplier, QoS_Warehouse, ...} {QoS_BankingService, ...}
Unknown: {InStock, PaymentPrefs, ...} {PaymentPrefs, DeliveryTimeShipment}
Service Composition
Receive RFQ
Produce Offer
Order Unavailable Parts
Assemble
Quality Control
Ship
Charge Customer
Quality / Timeliness Tradeoff
Prediction Quality
Time for Reaction
C1
C2

Figure 4.5: Possible Checkpoints in the ACMEBOT Process

yet. Therefore, the prediction cannot be very accurate, i.e., the prediction error is usually high. In checkpoint C_2, on the other hand, most of the composition's important raw data is already available as facts, allowing for good predictions. However, compared to C_1, the possibilities for adaptation are limited, as only the payment and shipping steps are left to adapt. Finding good checkpoints at which the prediction is reasonably accurate and still timely enough to react to problems demands for some domain knowledge about influential factors of composition performance.

Factors of influence are rarely obvious, even to domain experts. Hence, a process has been devised based on dependency analysis (as discussed before), which can be used by business analysts to identify factors of influence. This process is summarized here.

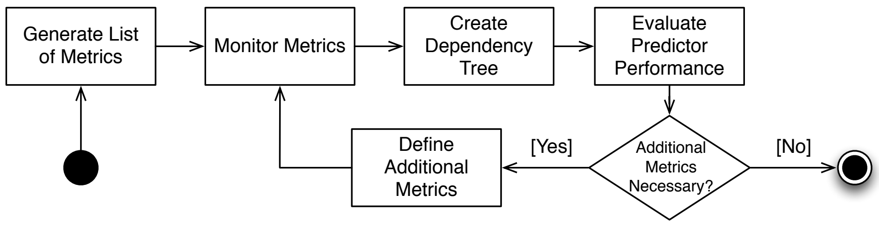

Generate List of Metrics
Monitor Metrics
Create Dependency Tree
Evaluate Predictor Performance
Additional Metrics Necessary?
[Yes]
[No]
Define Additional Metrics

Figure 4.6: Factors of Influence Analysis Process

The process for identifying factors of influence is a semi-automated process. It relies on the domain knowledge of a human business analyst, but supports her with automation and knowledge discovery tools to ease repetitive tasks. The high-level process is sketched in Figure 4.6.

As a first step, the business analyst needs to define an (initial) list of potential factors of influence. These include both, PPMs, which need to be defined manually, and typical QoS metrics, which can be automatically generated (e.g., for every used service response time and availability metrics are generated). For every potential factor of influence, a monitor is defined or generated, which specifies how this metric can be measured from a running instance. Secondly, a data set containing these factors needs to be generated, either by simulating the composition in a Web service test environment (e.g., Genesis2 [89]) or by monitoring real executions with monitoring of all potential factors of influence enabled. Using this data set, a dependency tree can be generated, as discussed above and in [186, 187]. The dependency tree is essentially a decision tree, containing the factors that best explain SLO violations in the composition. The third step is then to use these factors to try and train a prediction model from the identified factors of influence. If this prediction model has a sufficiently high training data correlation against the measured data set, these factors can be used as input to the prediction model. If the correlation is not sufficient, the business analyst needs to identify the reason for the lacking performance. Generally, the analyst will define additional potential factors of influence, and repeat from the second step.

Facts and Estimates

Facts represent all important information that has already been monitored and collected in a checkpoint. This includes both, QoS and PPM data. Note that the relationship between facts and the predicted SLO values does not need to be known. For instance, a domain specialist can include PPMs such as user identifiers or ordered items, even if she is uncertain about the relevance for the SLO. However, dependency analysis can again be used to identify the most important facts for a checkpoint.

In addition to measured data (facts), the domain specialist should also define estimates. In Figure 4.5, in C_1 the response time of the warehouse service is not yet known, however, it can (for instance) be estimated using QoS monitoring [128, 162]. In PREvent, the QoS monitoring facilities of VRESCo can be used. Estimating instance data is oftentimes domain-specific, hence PREvent is extensible in that more specific estimators can be integrated seamlessly via Estimator components. Estimates are linked to facts, in the sense that they have to represent an estimation of a fact which will be monitorable at a later point. Estimates are not mandatory for the approach, but they help improve prediction quality. However, one should keep in mind that only reasonably accurate estimates are helpful for prediction. If an estimate is oftentimes very far off the mark, it might be better to leave it out of prediction altogether. One reasonable metric for discovering bad estimates is to compare the training data correlation and the prediction error at runtime (see below). If the training data correlation is high (indicating a well-trained predictor), but the prediction error is high as well, then one plausible explanation for this mismatch are bad estimates. This is because estimates are at training time replaced for the facts they estimate. Hence, training is always done as if all estimates are entirely accurate.

In general, the PREvent framework does not incude any particular implementations for the generation of estimates, as this is usually domain-dependent and specific for the metric to estimate. The only estimator implementation, which is part of PREvent, is an estimator component for estimating the QoS characteristics of Web services based on historical QoS, as tracked in the VRESCo SOA runtime environment. A sample configuration of this component is depicted in

```
<estimatorClass
  class="DefaultEstimators.VrescoQoSEstimator, SLOPrediction"/>
<argument value="ResponseTime"/>
<argument value="CustomerService"/>
<argument value="avg"/>
<argument value="5"/>
```

Listing 4.3: QoS Estimator Component Configuration

Listing 4.3, and consists of four parameters. The first parameter is the name of the QoS dimension to estimate (e.g., “ResponseTime” or “WrappingTime”), the second parameter is the name of the service, as stored in VRESCo, the third parameter is an aggregation function (the possible values are “avg”, “mean”, “min”, “max”), and the fourth parameter is the number of QoS measurements to take into account (i.e., in the example the average of the last five measurements is returned).

Retraining Strategy

Generally, Prediction Models need to be rebuilt by the Predictor Manager whenever enough new information is available to significantly improve the model. The retraining strategy is used to define at which times the Predictor Manager should check whether rebuilding the prediction model is necessary. Table 4.1 summarizes all retraining strategies available, and gives examples. The custom strategy is defined using regular programming language code, all other strategies are pre-implemented in the PREvent prototype and can be used and configured without any additional code.

Strategy	Retrains ...	Example
periodic	... in fixed intervals	*every 24 hours*
instance-based	... whenever a fixed number of new composition instances have been fully monitored since the last training	*every 250 instances*
on demand	... on user trigger	n/a
on error	... if the mean prediction error (see below) exceeds a given threshold	*if* $\bar{e} > T$
custom	... if a user-defined condition applies	*whenever more than 10 orders from customer 12345 have been received*

Table 4.1: Predictor Retraining Strategies

Prediction Model Parameterization

A domain specialist can also define the machine learning technique that should be used to build the underlying prediction models. This is done by specifying an algorithm and the respective parameterization for the WEKA toolkit [68]. WEKA is internally used by the PREvent prototype implementation. In this way the prediction quality can be tuned by a machine learning savvy user, however, a useful default configuration is also provided, which can be used out of the box.

This default configuration is expected to deliver good, even if not necessarily optimal, prediction results without further parameterization. Listing 4.4 shows a simple example of a parametrized machine learning model, where an alternating decision tree (ADTree [59]) is used instead of C4.5. 20 boosting iterations are used to train the ADTree.

```
weka.classifiers.trees.ADTree -B20
```

Listing 4.4: Example Parametrization for AD Decision Tree

XML-Based Configuration

All these definitions are again specified in an XML-based format, similar to the metrics definitions described in Section 4.2. This proprietary language is referred to as Checkpoint Definition Language (CPDL). An excerpt of a CPDL file for the ACMEBOT case is provided in Listing 4.5.

In the listing, a checkpoint, which is hooked before the execution of the invoke activity "getPaymentPrefs", is defined. A multilayer perceptron is used as prediction model. As custom parameters, a learning rate of 0.4 and 1000 epochs of training have been specified. The checkpoint will be retrained periodically every 5 hours, and will predict the SLO `ORDER_FULFILLMENT_LEAD_TIME`. Afterwards, a number of available facts and estimates are specified. For estimates, an estimator class is given as a full qualified C# class name, which implements the actual prediction. Additionally, a number of arguments can be given to the estimator class, a discussed above. Finally, for every estimate a link to the estimated fact needs to be specified. Facts are defined as references of metrics in the Metrics Database, which are monitored by the Composition Monitor component.

```
<cpdl:checkpoints
  xmlns:cpdl="http://www.infosys.tuwien.ac.at/2009/cpdl">

  <checkpoint
    name="beforeGetPaymentPrefs"
    activityName="getPaymentPrefs" breakBefore="true"
    predictor="weka.classifiers.functions.MultilayerPerceptron -L 0.4 -N 1000">

    <update type="periodically" value="5"/>

    <class fact="ORDER_FULFILLMENT_LEAD_TIME"/>

    <fact fact="RESPONSE_TIME_WAREHOUSE"/>
    <fact fact="ORDER_INSTOCK"/>
    <!-- ... more facts ... -->
    <estimate name="getPaymentPrefsResponseTime" type="integer">
      <estimatorClass
        class="DefaultEstimators.VrescoQoSEstimator, SLOPrediction"/>
      <argument value="ResponseTime"/>
      <argument value="CustomerService"/>
      <argument value="Average"/>
      <argument value="5"/>
      <estimatedField fact="RESPONSE_TIME_GETPAYMENTPREFS"/>
    </estimate>
    <!-- ... more estimates ... -->
  </checkpoint>
  <!-- ... more checkpoints ... -->
<checkpoints>
```

Listing 4.5: Checkpoint Definition in XML Notation

Run-Time Prediction

At runtime, the prediction process is triggered by lifecycle events emitted by the VRESCo event engine, which have already been used to monitor composition metrics (cp. Section 4.2). When checkpoints are deployed, the hook information is used to register respective event listeners. For instance, for a checkpoint with the hook "before GetPaymentPrefs", as in Listing 4.5, a listener for `ServiceInvokedEvents` of the activity "GetPaymentPrefs" is generated. Please refer to [127] for details on available events in VRESCo. In Figure 4.7 the sequence of actions triggered by such a hook event is shown.

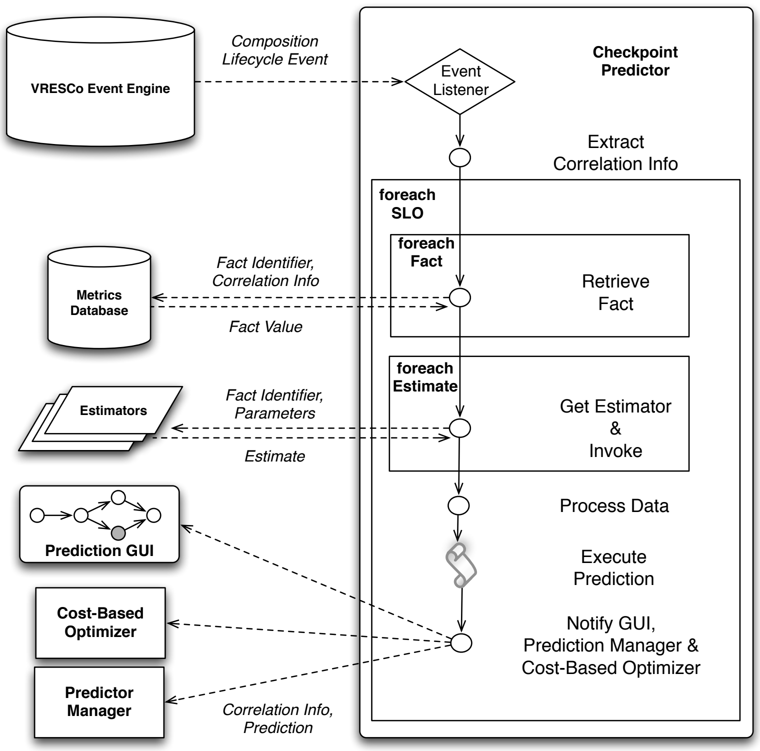


Figure 4.7: Runtime View On Checkpoint Predictors

After being triggered by a VRESCo event, the checkpoint predictor first extracts some necessary correlation information from the received event. This includes the composition instance identifier, as assigned by the composition engine, the instance start time (i.e., the time when the instance was created) and the timestamp of the event. This information is necessary to be able to retrieve the correct metrics from the Metrics Database (e.g., in order to find the correct fact "CustomerNumber" for the correct composition instance, the instance identifier needs to be known). Now, the Checkpoint Predictor can generate predictions for each SLO. Firstly, all already available facts are retrieved from the Metrics Database. When all facts have been gathered, the predictor also collects the missing estimates. For this, for every estimate the predictor contacts the Estimator Pool, retrieves an instance of the respective estimator component, and invokes it (passing all necessary parameters as specified in the checkpoint definition). The gathered facts and estimates are then converted into the format expected by the prediction model (in the case of the PREvent prototype, this is the WEKA Attribute-Relation File Format ARFF[6]), and, if necessary, some data cleaning and postprocessing is done. Afterwards, the actual prediction is carried out by passing the gathered input to the Prediction Model responsible for this SLO, producing either a numerical or non-numerical estimation. This prediction can then be forwarded to all components interested in new predictions (most importantly the Cost-Based Optimizer).

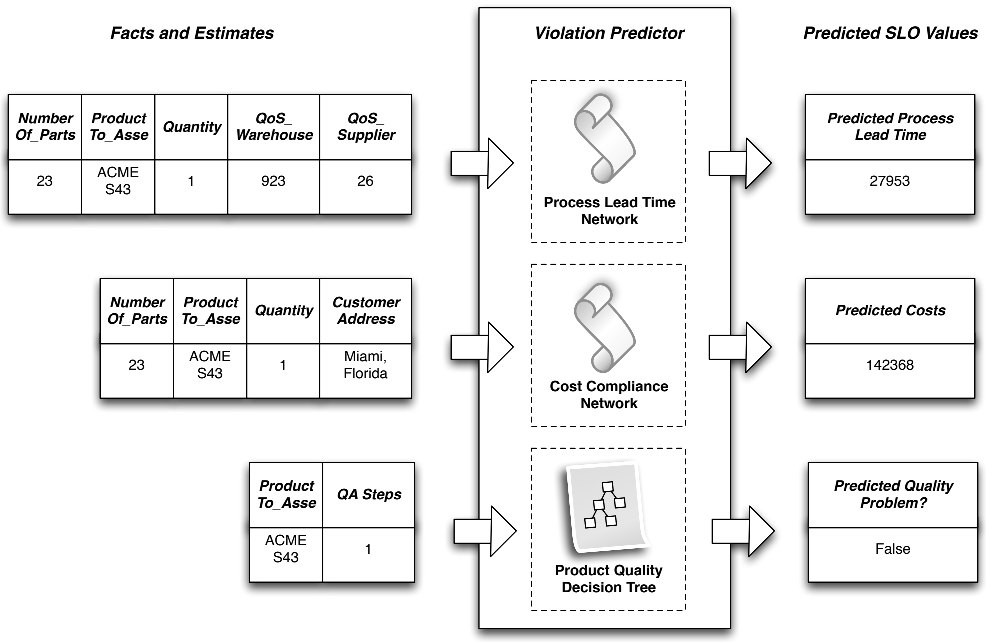


Figure 4.8: Black-Box Prediction Models

Evidently, the "intelligence" that actually implements the prediction of the SLO values is contained in the Prediction Model. As stated above, the Prediction Model is a black-box function which takes a list of numeric and nominal values as input, and produces a numeric (for quantitative SLOs) or non-numeric (for qualitative SLOs) output (Figure 4.8).

Evaluation of Predictors

Quality management of predictors is one of the main tasks of the Predictor Manager. Quality management includes continually supervising how predictions compare to the actual SLO values, once the instance is finished. Generally, three different quality metrics are used to measure the quality of predictions in checkpoints.

[6]http://www.cs.waikato.ac.nz/~ml/weka/arff.html

$$corr = \frac{cov(P,H)}{\sigma_P \sigma_H} \tag{4.1}$$

The first metric, *Training Data Correlation*, is defined in Equation 4.1. Training Data Correlation is a standard machine learning approach to evaluating regression models. In Equation 4.1, H is the training set of historical SLO values, P are the predictions that would have been generated for these historical instances, and $cov(P,H)$ is the statistical covariance of P and H. Furthermore, σ_P and σ_H denote the statistical standard deviations of H and P, respectively. In PREvent, $corr$ is used mainly to evaluate freshly generated Prediction Models, when no actual runtime predictions have yet been carried out. However, note that this metric is inherently overconfident, as during training all estimates are replaced for the facts that they estimate (i.e., the training is done as if all estimates were perfect). However, a low training data correlation is an indication that important facts are still unknown in the checkpoint, i.e., that the checkpoint may be too early in the composition to do much good.

$$\bar{e} = \frac{\sum_{i=0}^{n} |m_i - p_i|}{n} \tag{4.2}$$

The most important quality metric in PREvent is the *Mean Prediction Error* $\bar{e}$, as defined in Equation 4.2. Simply said, $\bar{e}$ is the average (Manhatten) difference between predicted and monitored values. In Equation 4.2, n is the total number of predictions, p_i is a predicted value, and m_i is the measured value to prediction p_i (that is, every prediction is compared to the value that has been measured after this instance was finished).

$$\sigma_{\bar{e}} = \sqrt{\frac{\sum_{i=0}^{n} (e_i - \bar{e})^2)}{n}} \tag{4.3}$$

Finally, the *Prediction Error Standard Deviation* (denoted here as $\sigma_{\bar{e}}$), as defined in Equation 4.3, is used to describe the variability of $\bar{e}$ (i.e., high $\sigma_{\bar{e}}$ essentially means that the actual error for an instance can be much lower or higher than $\bar{e}$). In Equation 4.3, e_i is the actual prediction error for an instance $(m_i - p_i)$.

These three error metrics are mainly used to evaluate the reliability of predictions produced by the SLO Predictors. Additionally, the "on error" retraining strategy (see Table 4.1) triggers on $\bar{e}$ exceeding a certain threshold.

WEKA-Based Implementation

As stated before, PREvent makes use of the WEKA machine learning toolkit for building Prediction Models. For this, the WEKA Java API[7] has been used. As PREvent is implemented in C#/.NET, a Java-based wrapper has been built as a RESTful Web service to enable access to WEKA from C#. No application code of WEKA has been modified for this. Usage of WEKA in PREvent has the advantage that business specialists, especially those with in-depth knowledge of machine learning technology, are not bound to using one algorithm for building Prediction Models. Instead, any of the plethora of algorithms implemented by the vivid WEKA community

[7] http://weka.wikispaces.com/Use+WEKA+in+your+Java+code

can be used. This allows to use advanced techniques for tuning the performance of predictors, such as bagging [23], boosting [58] or stacking [193].

4.4 Cost-Based Optimization

For this thesis, the most important task of the SLO Predictor is to trigger the Cost-Based Optimizer. In the MAPE loop, this component implements the **P**lanning. The Cost-Based Optimizer takes predictions of SLOs and monitored metrics as input. If the predictions indicate that no violation will occur, the component does nothing. However, if violations are predicted, the optimizer selects the most cost-effective adaptation actions to prevent the predicted violations. However, note that this does not mean that every violation will surely be prevented. If there is no cost-efficient way (or simply no way at all) to prevent a violation, the Cost-Based Optimizer might still decide that the best course of action for the service provider is to bite the bullet and accept one or more violations.

Optimization Problem Formulation

In this section, the problem of selecting the most cost-effective adaptation actions to prevent one or more predicted SLA violations in a service composition, is formalized. In the following, an interaction of the service composition with a given client, who has a given SLA with the composition provider, is considered. Let I be the set of all possible composition instances of this client, and let $i \in I$ be concrete instances that can be monitored using the Composition Monitor. Furthermore, let $S = \{s_1, s_2, \ldots s_k\}$ be the set of SLOs defined in the SLA. As part of the SLO definition, a penalty function is associated with all SLOs in S. Collectively, these functions are referred to as $P = \{p_{s1}, p_{s2}, \ldots p_{sk}\}$. Penalty functions define the costs for the provider based on a measured SLO value, i.e., they are functions defined as $p_s : \mathbb{R} \rightarrow \mathbb{R}, s \in S$. Similarly, the measured value of an SLO m_s is a function $m_s : I \rightarrow [0:1]$. For simplicity, SLO values are normalized to the interval $[0:1]$, in order to make them easier to compare. Putting it all together, penalty functions are defined for a given SLO s and instance i as $p_s^i \overset{\text{def}}{=} p_s(m_s(i))$.

Penalty functions can take many different shapes. Some examples of typical penalty functions are given in Figure 4.9. Even though this is not a complete list, these functions still incorporate the most common types of penalties:

- *Constant penalty* - a constant payment needs to be made if a given threshold value is surpassed.
- *Staged penalty* - similar to a constant penalty, but with different levels of penalty.
- *Linear penalty* - the penalty is linearly increasing with the degree of violation.
- *Linear penalty with cap* - the penalty is linearly increasing up to a maximum value.
- *Asymptotic penalty* - the penalty asymptotically converges against a maximum penalty.
- *Polynomially increasing penalty* - the penalty increases quickly with the degree of violation.

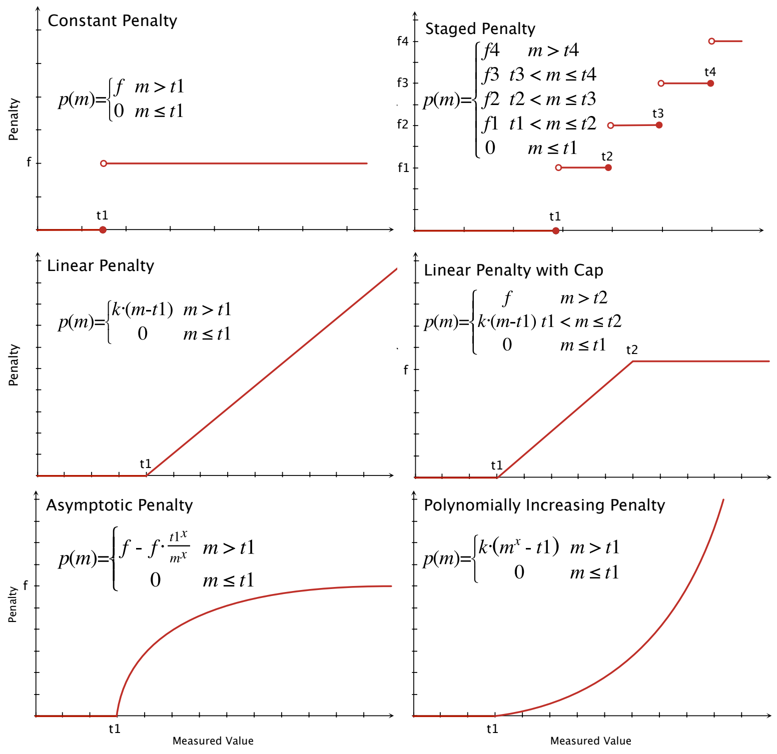


Figure 4.9: Typical SLA Penalty Functions

The two latter alternatives are included mostly for reasons of completeness, and have little practical relevance. In the remainder of this thesis, focus is set on the first four variants. Note that, even though these functions apparently span many different types of mathematical functions, they share two essential characteristics. Firstly, penalty functions are always monotonically increasing, i.e., $\forall p_s \in P : \forall x_1, x_2 \in \mathbb{R} : (x_1 < x_2) \implies (p_s(x_1) \leq p_s(x_2))$. This is evident, as the penalty for a higher degree of violation should never be smaller than the penalty for a lesser violation. Secondly, penalty functions always have a point discontinuity in a special violation threshold point (t_1). Before (and including) t_1 the penalty is generally 0 (no violation has occured), and beyond this point, a positive penalty needs to be paid ($\forall s \in S : \forall x \in \mathbb{R} : (x \leq t_1 \iff p_s(x) = 0) \wedge (x > t_1 \iff p_s(x) > 0)$). This also means that penalty functions are generally discontinuous. Furthermore, this property signifies that there is no incentive for the service provider to apply further adaptation and improve an SLO value below t_1, as all further improvements do not further reduce his costs (they are already 0 for this SLO).

To prevent violations, a number of possible adaptations can be applied to an instance i. These actions are defined as $A = \{a_1, a_2, \ldots a_l\}$, and $A^* \in \mathcal{P}(A)$ ($\mathcal{P}(A)$ denotes the powerset of A) is one concrete subset of adaptation actions, for instance the adaptations that are selected to be applied. All adaptations have some associated costs, defined as a cost function $c : A \to \mathbb{R}$. Cost functions are assumed to be constant, that is, cross-pricing models for services [200], which would lead to non-constant costs of adaptation, are not taken into account here. Furthermore, adaptation actions have, if applied, some defined impact on the composition instance i. Hence, the transformation of i to a modified instance i' is defined using the $\circ$ operator, which is a function $\circ : I \times \mathcal{P}(A) \to I$. This impact is captured by the impact model, which has to be specified as part of the action definition (see Section 4.5).

Selecting the most cost-effective adaptation actions means finding the A^* that minimize the total costs for the service provider. The total costs TC are defined in Equation 4.4 as the sum of the costs of SLA violations after adaptation (VC) and the costs of adaptation (AC).

$$TC : \mathcal{P}(A) \to \mathbb{R}, \quad TC(A^*) = VC(i \circ A^*) + AC(A^*) \tag{4.4}$$

AC is the sum of the costs of all applied adaptation actions (Equation 4.5).

$$AC : \mathcal{P}(A) \to \mathbb{R}, \quad AC(A^*) = \sum_{a_x \in A^*} c(a_x) \tag{4.5}$$

VC is defined as the sum of all penalty functions applied to an instance (Equation 4.6).

$$VC : I \to \mathbb{R}, \quad VC(i) = \sum_{s_x \in S} p^i_{sx} \tag{4.6}$$

Obviously, the goal of the service provider is to minimize TC. Hence, the optimization objective becomes finding the A^* that minimizes TC for a given instance i (Equation 4.7).

$$TC(A^*) = \sum_{s_x \in S} p^i_{sx} + \sum_{a_x \in A^*} c(a_x) \to min! \tag{4.7}$$

Note that it is easy to calculate AC for any given A^*, but, at runtime, when this optimization problem has to be solved, VC is unknown (it is impossible to know for sure, which SLOs will be violated when the instance is finished, with or without adaptation). However, the SLO Predictor component provides means to estimate SLOs based on monitoring data. Hence, it can be assumed that there are estimation functions $e_s : I \to \mathbb{R}, s \in S$ available for each SLO, which estimate the concrete penalty values in advance with a reasonably small estimation error ϵ ($\forall s \in S, i \in I : |e^i_s - p^i_s| < \epsilon$). Replacing VC with its estimation using e_s leads to Equation 4.8, which can be solved.

$$TC(A^*) \approx \sum_{s_x \in S} e^i_{sx} + \sum_{a_x \in A^*} c(a_x) \to min! \tag{4.8}$$

However, not all combinations of adaptation actions are legal. Some adaptation actions are mutually exclusive (e.g., `use Shipping Service DHL` and `use Shipping Service UPS`), while others depend on each other. For simplicity, these additional constraints are captured using a penalty term $v : \mathcal{P}(A) \to \mathbb{N}$. The definition of v is shown in Equation 4.9. Section 4.5 below will explain how these constraints are defined as part of the action definition.

$$v(A^*) = \begin{cases} \infty & A^* \text{ contains constraint violation} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Incorporating this penalty term into Equation 4.8 leads to the final target function (Equation 4.10).

$$TC(A^*) \approx v(A^*) + \sum_{s_x \in S} e^i_{sx} + \sum_{a_x \in A^*} c(a_x) \rightarrow min! \tag{4.10}$$

All necessary information to evaluate Equation 4.10 is available in a checkpoint at optimization time, for any set of actions A^*. However, finding the A^* that minimizes $TC(A^*)$ is still far from trivial, as Equation 4.10 is discrete and cannot be optimized using analytical means.

Algorithms

In the following, algorithms to find a optimal or near-optimal solution for this optimization problem are discussed. These algorithms form the basis of the Cost-Based Optimizer component. For all following discussions of algorithms, a binary encoding is used to represent solutions. Every solution A^* is represented as a binary vector, and an adaptation action with index j is applied iff the solution vector is 1 at index j. For example, the solution vector 00110100 encodes that the third, fourth and sixth adaptation action should be applied. Evidently, $2^{|A|}$ different solutions exist for each optimization problem, where $|A|$ is the number of possible adaptation actions (even if some combinations contain constraint violations and have a value of ∞, as per Equation 4.10). For solutions that are still being constructed, a third symbol is allowed, "$*$", representing an action which is still undecided (also referred to as "alive"). Solutions which contain at least one alive action are called "partial", and solutions which do not contain any alive actions are "complete". Therefore, the vector $001101 * 0$ is a partial solution, where the last-but-one action is alive.

Branch-and-Bound

Branch-and-Bound [103] is a very general deterministic algorithm for solving optimization problems. The high-level idea of this algorithm is to enumerate the solution space in a "smart" way, so that at least some sub-optimal solutions can be identified and discarded prematurely, i.e., before they have been fully constructed and evaluated.

The general branch-and-bound algorithm as used in PREvent is depicted in Listing 4.6. The algorithm is straight-forward and easy to understand. What is most important is the implementation of Line 13, the rules for pruning the search tree (i.e., for prematurely discarding solutions). In PREvent, partial solutions are pruned in two cases:

1. Solutions are pruned if a partial solution already contains at least one constraint violation.

2. Solutions are pruned if the partial solution already prevents all violations (the penalty function p^i_s is 0 for all SLOs s) without applying any more actions.

Case (1) is trivial, as the target function value for all solutions in such a sub-tree will always be ∞. Case (2) lends itself to more discussion. Remember the assumption that every action has non-negative costs, and that penalty functions are defined as non-negative. Therefore, it

```
 1 # name: bab
 2 # input: partial solution p,
 3 #        next alive action index i,
 4 #        target function v
 5 # output: optimal complete solution
 6
 7 bab(p,i):
 8   # recursion break condition
 9   if(p is complete solution)
10     return p
11
12   # check if this sub−tree can be pruned
13   if(p is pruneable)
14        forall alive_actions(p) as j
15           set p(j) = 0
16     return p
17
18   # investigate solution sub−tree with p(i)=0
19   set p(i) = 0
20   s1 = bab(p,i+1)
21
22   # investigate solution sub−tree with p(i)=1
23   set p(i) = 1
24   s2 = bab(p,i+1)
25
26   # return better solution from both subtrees
27   if(v(s1) <= v(s2))
28     return s1
29   else
30     return s2
```

Listing 4.6: Branch-and-Bound Algorithm

is assured that for any solution where all penalty functions are 0, the additional application of more actions can never improve the target function value. Hence these partial solutions cannot be improved by applying more actions, and the remaining solution sub-tree can be pruned.

Note that in Listing 4.6, the algorithm simply iterates over all actions in the order they appeared in the solution vector (see Lines 18 and 22). There is no specific order in which actions are investigated, In general, this approach is suboptimal. Even though the order in which actions are investigated has no impact on the quality of the final solution (the algorithm is deterministic, i.e., it will always find the global optimum eventually), the order may have an impact on the number of solutions that can be pruned. This is illustrated in Figure 4.10. Assume the following simple scenario: there is only one SLO, and 3 possible adaptations. Only adaptation 3 is able to prevent the violation of the SLO. Actions 1 and 2 have costs but no relevant influence. There are no conflicts between actions. Hence, the optimal solution vector is 001. In Figure 4.10(a), actions are investigated strictly in order of their position in the solution vector. Since the only "useful" action is investigated last, the whole solution tree is extended without any pruning (the worst case, equivalent to full enumeration). Now, in Figure 4.10(b), actions are investigated in reverse order (from back to front). Now, the "useful" action is investigated first, and a large part of this solution tree can be pruned according to pruning case (2).

Therefore, it can be concluded that it is beneficial to investigate actions in a specific order that

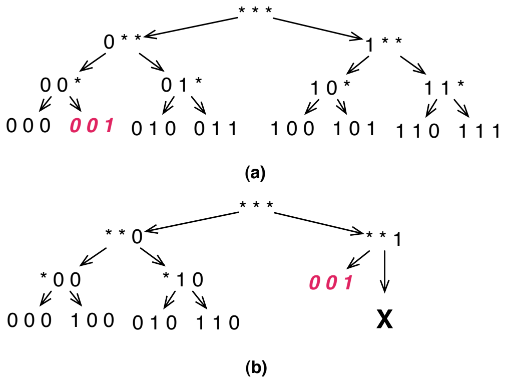


Figure 4.10: Pruning of Solution Trees

maximizes the number of solutions that can be pruned. Two possible criteria for this ordering are discussed in this thesis:

- *Impact-Based Sorting* - actions with higher impact on the SLOs should be investigated first.
- *Utility-Based Sorting* - actions with higher utility (relation between impact and costs) should be investigated first.

Assuming there is a defined set of historical process instances available, an estimation of impact and utility of each action can be calculated. The same definitions as in Section 4.4 are used. Additionally, consider the following definitions. The set of available historical process instances is referred to as $H = \{h_1, h_2, \dots h_q\}$, with $H \subseteq I$. The number of historical instances is referred to as $q = |H|$. Now, an estimation of the overall impact of an adaptation action a on a SLO s can be calculate as $\Delta_{a,s}$ (Equation 4.11). Simply put, the impact is the arithmetic mean of the difference between SLO value with and without applying the adaptation to each historical instance.

$$\Delta_{a,s} = \sum_{h \in H} \frac{m_s(h) - m_s(h \circ \{a\})}{q} \tag{4.11}$$

Note that, as defined in Section 4.4, penalty functions are monotonically increasing. Hence, higher impact values are generally good. However, the impact value may also be negative (i.e., $m_s(h) < m_s(h \circ \{a\})$). In this case, the action has a negative impact on one of the SLOs, which is reasonble and realistic. For instance, an adaptation which reduces the SLO Order Fulfillment Time can very well have a negative impact on the SLO Cost Compliance at the same time. Based on $\Delta_{a,s}$, it is now possible to define the total impact of each action (Equation 4.12).

$$\Delta_a = \sum_{s \in S} \Delta_{a,s} \tag{4.12}$$

Finally, u_a is defined as the utility of an action (Equation 4.13).

$$u_a = \frac{\Delta_a}{c(\{a\})} \tag{4.13}$$

u_a is the sum of the impact values of this action on all SLOs in relation to the costs of this action. A high utility means that, on average, this action is cheap for its usefulness in preventing violations. It is now possible to improve the branch-and-bound algorithm trivially. Instead of investigating the actions in the order they are specified in the solution vector, they are investigated either in the order of their impact Δ_a (impact-based sorting), or in order of their utility u_a (utility-based sorting).

Local Optimization

While the Branch-and-Bound algorithm discussed above has the advantage of always finding the optimal set of actions for any composition instance, the execution time of the algorithm increases exponentially with the number of available actions. Even though the execution time can be reduced using impact- or utility-based sorting of actions, the computational complexity still remains exponential. Hence there is an evident need to find strong heuristics [206], i.e., non-deterministic algorithms that find "good" (even if not necessarily optimal) solutions in polynomial time.

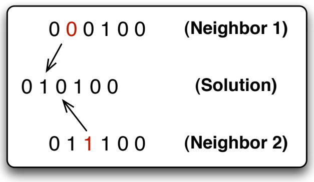


Figure 4.11: Neighborhood for Local Optimization

A simple heuristic that is often used with very good results is Local Search. Local Optimization is a meta-heuristic [81], i.e., final solutions are constructed by iteratively improving a start solution. The general idea of Local Optimization is as follows. In each iteration, the algorithm searches a specified *neighborhood* for better solutions than the current one. If at least one such solution is found, the algorithm progresses to the next iteration with one of the better solutions (typically the best one in the neighborhood, equivalent to steepest descent). If no better solution can be found, the algorithm has converged to a local optimum and is terminated. Usually, this procedure is repeated multiple times with different starting solutions (since, obviously, different starting solutions can lead to different local optima). This kind of algorithm typically depends on the definition of both a suitable neighborhood and a senseful selection of starting solutions. We use the following neighborhood definition: a complete solution vector is in the neighborhood of an original vector if the two vectors have a Hamming distance [70] of 1, i.e., if they differ in exactly one bit. This neighborhood definition is visualized in Figure 4.11. According to this definition, every solution vector has a neighborhood of size $|A|$. Additionally, note that two different solutions in the same neighborhood have a Hamming distance of 2.

For selecting the start solutions, two different approaches are used. The first and primitive one is to select n start solutions with m random bits set to 1. Alternatively, an algorithm commonly referred to as Greedy Randomized Adaptive Search Procedure (GRASP) [55] can be used. GRASP is essentially a variation of Local Optimization, where the start solutions are constructed using a greedy heuristic. The idea is that GRASP can converge to a better solution than a simple Local Optimization, because the GRASP start solutions are already better to begin with than random start solutions. However, some attention needs to be paid to using a greedy construction heuristic that actually generates start solutions, which are both of reasonable quality and at the same time widely spread over the search space. This is evident, as many very similar start solutions (which all converge to the same local optimium during Local Optimization) do not improve the overall solution quality at all, while one distinctly different solution might.

```
# name: grasp_init
# input: number of start solutions n,
         RCS max size r
# output: set of start solutions

grasp_init(n, r):
  G = {} // empty set of start solutions
  repeat n times:
    pa = empty_partial_solution
    while( VC(pa) > 0 ):
      rcs = construct_rcs(pa, r)
      if( empty(rcs) )
        break
      a = random(rcs)
      pa(a) = 1
    add(G, pa)
  return G
```

Listing 4.7: GRASP Construction Heuristic

The used GRASP construction heuristic that is sketched in Listing 4.7. The algorithm constructs n solutions by stepwise addition of actions selected randomly from a Restricted Candidate Set (RCS). The heuristic is based on similar concepts that have already been used in the discussion of Branch-and-Bound. The general idea is to stop adding actions if either no more SLOs are violated or no senseful actions are available anymore (the RCS is empty), and to prefer adding actions which have a high utility value (u_a). Hence, in every step the RCS consists of the r (defined as the maximum size of the RCS) actions with highest non-negative u_a, which have not yet been added and which do not lead to a conflict.

Genetic Algorithm

As an alternative to locality-based heuristics (Local Optimization, GRASP), this thesis also discusses a solution based on the concept of evolutionary computation [87]. More precisely, Genetic Algorithms (GAs) [66, 189] are used as a more complex, but potentially also more powerful heuristic to generate good solutions to the cost-based optimization problem. The overall idea of GA is to mimic the processes of evolution in biology, specifically natural selection of the

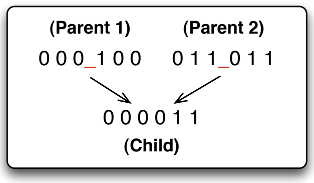


Figure 4.12: One-Point Crossover

fittest individuals, crossover, and mutation. Therefore, GA works on a population of solutions instead of a single one. The term "fitness" is used to describe solutions with a good (low) target function value.

The overall GA algorithm is as follows. As a first step, a random start population is generated. In the PREvent implementation, the same primitive construction scheme as discussed above for Local Optimization is used., i.e., in every start solution m random actions are applied. Every following iteration of the algorithm (referred to as generations) essentially follows a three-step pattern.

Firstly, a set of solutions is *selected* from the population to "survive" into the next generation. In PREvent, the fittest solution (i.e., the one with the lowest target function value) is selected deterministically (this principle is often referred to as elitism [16]), while all remaining slots in the next generation population are selected using tournament selection [65]. In tournament selection, t random solutions from the last generation are put into a tournament. The fittest solution of the tournament is selected into the next generation. The parameter t steers the selection pressure: low t increases the time that the population takes to converge against a solution, but high t increases the danger of converging against a local optimum instead of the global one.

Secondly, *crossover* is used to produce new solutions based on the selected ones from the last generation. The main challenge of implementing a strong crossover mechanism is to ensure that the crossover product of two fit solutions is also likely to be fit. Given the binary vector representation that is used to encode solutions, a simple one-point crossover scheme has been devised. For this, a random crossover point cp is chosen from $[1 : |A| - 1]$. To construct a new child the solution vector of the first solution is copied from the start until cp, and the vector of the second solution from $cp + 1$ to the end. This crossover scheme is illustrated in Figure 4.12.

This simple procedure ensures that characteristics of both original solutions are preserved. However, because of the random selection of cp, it is possible that the child solution has a conflict, even if this was not the case for any of the parents. In this case, one of the conflicting actions is removed at random.

Finally, the principle of *mutation* is used to introduce entirely new features into the population. The need for mutation can be illustrated easily. Assume that a given action a is not applied in any solution in the population. Using one-point crossover as discussed above it is not possible to create any solution that uses a. Hence, some additional randomization is introduced. After crossover, every bit in every solution in the population is randomly flipped with a very small probability. This means that most solutions in the population are not mutated, but sometimes new actions are applied, which are not the product of crossover.

Name	Description	Default
Population Size	Number of solutions in every generation	150
Selection Pressure	Number of solutions to select for tournament selection	2
Crossover Probability	Probability per solution that crossover is applied	0.8
Mutation Probability	Probability per bit that mutation is applied	0.02
Break Condition	Condition for stopping the algorithm	No improvement in 20 generations

Table 4.2: GA Configuration Parameters

GAs are notorious for having many parameters to fine-tune the performance of the optimization. For illustrative purposes, the parametrization options available in the PREvent GA are summarized in Table 4.2, including some values that have been found to provide useful default parameters if applied to the cost-based optimization problem. Evidently, further customization is possible, for instance, by using a different selection scheme. Other possibilities include fitness-proportional selection, ranking selection [65] or the more exotic fitness-uniform selection [80]. Investigating these further possibilities in more detail was not part of this thesis work.

Unfortunately, the "canonical" GA presented here takes a significant amount of time to converge against a solution, as the solution space is searched solely through the (rather unguided and strongly randomized) means of crossover and mutation. One possibility to improve this aspect is to combine the canonical GA with Local Optimization. This leads to an adapted algorithm, which is sketched in Figure 4.13. In literature, such combinations of GA and Local Optimization are often referred to as Memetic Algorithm (MA) [154].

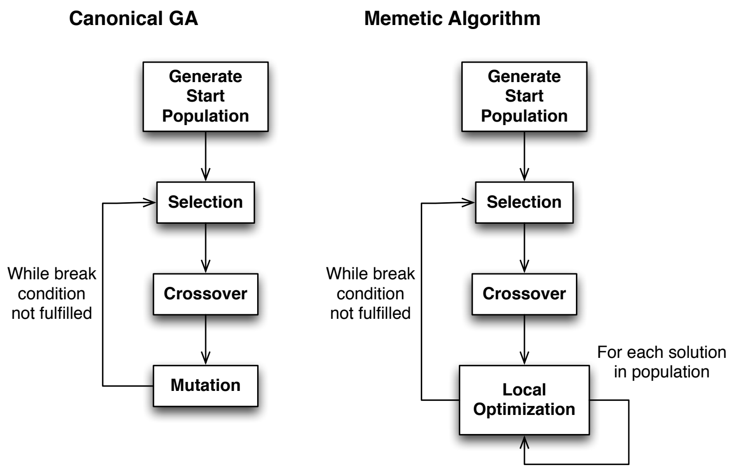

Canonical GA
Memetic Algorithm
Generate Start Population
Selection
Crossover
Mutation
While break condition not fulfilled
Generate Start Population
Selection
Crossover
Local Optimization
While break condition not fulfilled
For each solution in population

Figure 4.13: Memetic Algorithm

The main changes of MA (as compared to GA) are as follows. Firstly, a new *Local Optimization* operator is introduced after crossover. This operator applies the Local Optimization algorithm as discussed above to each solution in the generation, basically reducing the population to a set of locally optimal solutions. Secondly, the mutation operator is removed from the algorithm (technically speaking, the mutation probability parameter is set to 0). The main reason is that, given that all solutions in the population are already locally optimal, randomly mutating one bit in a solution can only lead to a worse solution. In theory, it is possible that multiple bits in a single solution are mutated at the same time, and that these mutations lead to an improvement, but this corner case is very unlikely in practice. Furthermore, the main motivation for having mutation in the first place was that it is the only way of introducing new actions in the canonical GA. This is no longer the case, as Local Optimization can do the same thing.

Generally, MA is slower than GA, as more solutions are evaluated in each generation (evidently, MA executes one Local Optimization for every solution in each generation). However, the algorithm potentially converges against a very good solution in a small number of generations. Hence, it can be argued that in practice MA improves on the canonical form most of the time for the cost-based optimization problem.

Timeliness Issues of Optimization

Even using fast meta-heuristic algorithms, optimization is still quite time-consuming. Therefore, in order not to delay the execution of the composition, it is desirable to execute the optimization asynchronously, i.e., in parallel to the service composition. However, in this case timing aspects of the optimization and the composition need to be kept in mind.

Timely Optimization and Stale Results

Figure 4.14 showcases the timing of asynchronous optimization. For an instance of the composition, an optimization is triggered at time t_0 (e.g., because SLO estimation mechanisms have predicted that this instance is going to violate its SLAs). An optimization algorithm, e.g., GA, starts searching the solution space. Meanwhile, the service composition continues executing. Firstly, assume that at time t_1 the optimization has converged and delivers the result that two adaptation actions have to be applied. For both actions the optimization was timely and the result is useful. However, assuming that the algorithm takes more time and delivers its result at time t_2 instead, the activity "Invoke Planning" (`P`) has already been executed, and part of the result (the decision to adapt `P`) came too late. For each adaptation action used in this thesis (as will be discussed in more detail in Section 4.5), it is possible to define the affected region in the service composition, i.e., the activities in the composition which are affected by the adaptation. For service rebinding or data manipulation actions, this is exactly one activity. For fragment-based structural adaptation, the affected region may be arbitrarily large, but is still always clearly defined. In this thesis the term "beginning of the affected region" (t_a^x) is used to indicate the time that the first activity affected by adaptation x starts to execute.

Intuitively, for every adaptation x with a defined affected region there is also a decision point t_d^x, the latest time in the execution of the composition when a decision needs to be made. Assuming that t_a^x is known, and the time that the application of the adaptation technically takes

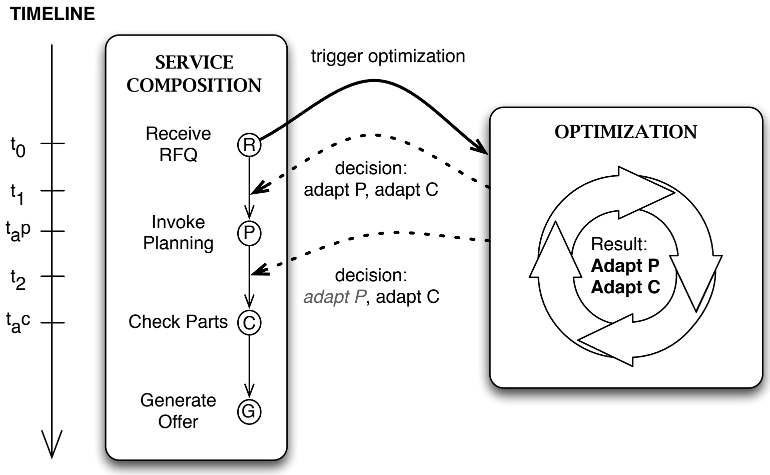


Figure 4.14: Optimization Timing

(d_x), it is possible to define the decision point of an adaptation as $t_d^x = t_a^x - d_x$. An optimization result A^* produced at time t is defined as "stale" if $\exists a \in A^* : t_d^a < t$. Stale optimization results cannot be applied in full anymore when they are available, and evidently should be avoided.

Stepwise Optimization

Two approaches can be used to handle the problem of stale results. Firstly, one can decide not to deal with the problem at all, ignoring stale results and applying only what is still possible when the result becomes available. This approach is very simple, and even in the worst case this is at least never worse than not doing optimization to begin with, even if the result may be suboptimal in the presence of stale results. Secondly, one can drop the idea of asynchronous optimization and halt the service composition while the optimization is running. This trivially prevents stale results, but severely degrades the performance of the service composition. It is well possible (if the optimization takes more time than what can be gained using adaptation, and there are execution time related SLOs) that using this approach is actually worse than not doing any optimization at all. It is easy to see that both of these ideas are not optimal. Hence, a stepwise asynchronous optimization is now proposed. This thesis considers stepwise optimization as an alternative principle to prevent stale results. The general approach is sketched in Figure 4.15.

First of all, all adaptations are sorted according to their decision points, and adaptation actions with identical decision points ($t_d^x = t_d^y$) are collected in decision sets (D_i). Let t_s^x be the decision point of a decision set D_x, defined as the decision point of all adaptations contained in the set ($\forall a \in D_i : t_d^a = t_s^i$). In the figure, two decision sets with decision points t_s^p and t_s^c exist. The first decision set contains only the action "Invoke Planning" (P), the second contains only the action "Check Parts" (C). As before, an optimization is triggered at t_0. However, results are now delivered differently. Instead of waiting for the optimization to converge, the decision procedure sketched in Listing 4.8 is triggered when t_s^p and t_s^c are passed.

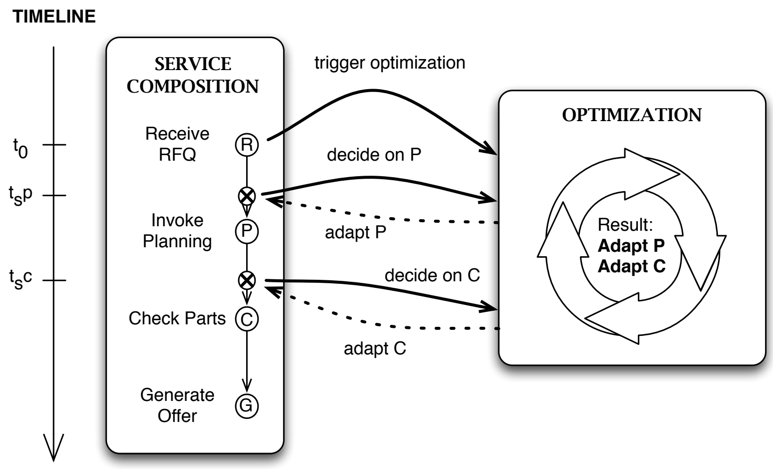


Figure 4.15: Stepwise Asynchronous Optimization

For every decision point associated with a decision set, the composition is briefly halted and decisions are made on all adaptations associated with this set. If it is decided to apply one or more adaptations, the adaptations are carried out as discussed in Section 4.5. Finally, the composition is resumed. This way, instead of producing the solution for the optimization problem in one big bang (and risking that the result arrives too late), a stepwise, constructive approach is used. This approach defers all decisions about adaptations to the latest possible time, but never later. Hence, results are never stale, and the service composition needs to be halted only briefly. Note that it is generally advantageous to wait as long as possible before making a decision (per definition, that means waiting until t_d^x), under the assumption that the quality of a decision is monotonically increasing with optimization time. This is true for most implementations of optimization algorithms, including GA (if elitism [16] is used). The decision algorithm in Figure 4.8 contains two challenges, which are briefly discussed here.

```
 1 decide(DecisionSet ds, Optimization opt):
 2
 3   suspend_process()
 4
 5   foreach(Adaptation a in ds):
 6     decision = decide_on_adaptation(a, opt)
 7     if(decision == true)
 8       apply_adaptation(a)
 9     adapt_target_function(a, decision, opt)
10
11   resume_process()
```

Listing 4.8: Decision Procedure for Stepwise Optimization

The first challenge is that a strategy needs to be defined for deciding on a single adaptation based on still ongoing optimizations (Line 6). Two strategies for implementing this come to mind.

- *Current-Optimum Based Decision* - One simple, yet promising, strategy is to base the decision on the best currently known intermediary result, i.e., decide to apply an adaptation if and only if the currently best known solution applies the adaptation. This strategy is referred to as current-optimum based decision.

- *Weighted-Voting Based Decision* - However, in the case of an early decision point (i.e., the optimization has just been started) the current-optimum based approach might be misleading. In this case it may be advantageous to use another decision strategy, which does not rely on the quality of a single solution, and instead reflects the combined knowledge of a population of different solutions (weighted-voting based decision). In weighted voting, all solutions in a population vote on whether an adaptation should be applied, and every solution has a weight equivalent to its relative solution quality as compared to the total population (i.e., better solutions with a lower solution target value have a higher weight in the voting). An adaptation is applied if the majority of solutions in the population (after weighting) votes for applying it.

The second challenge is that after deciding on an adaptation, the target function of the optimization problem needs to be modified. This is to reflect the fact that whenever a decision is made (and a given adaptation action is applied or rejected) the underlying problem of the ongoing optimization has in fact changed. If the adaptation has been applied, all solutions that do not use this action are invalid. Similarly, if the adaptation has not been applied, all solutions using the adaptation are invalid. Hence, in Line 9 of the algorithm, the optimization algorithm is suspended, additional constraint for each adaptation in the decision set are added, and the optimization is resumed. These additional constraints are easily represented using additional penalty terms v_x in the target function. If an action a_i is applied, a new penalty term v_i as in Equation 4.14 is created.

$$v_i(A^*) = \begin{cases} \infty & a_i \notin A^* \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

The penalty term for representing a rejected adaptation is defined analogously (Equation 4.15). These penalty terms are simply appended to the target function definition TC, leading to an updated target function $TC'(A^*) = TC(A^*) + v_i(A^*)$, which is then used for the ongoing optimization.

$$v_i(A^*) = \begin{cases} \infty & a_i \in A^* \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Note that, depending on the concrete optimization algorithm used, it might be necessary to re-evaluate existing solutions after adapting the target function. For instance, in GA it is necessary to re-evaluate the target function value of all members of the current population before continuing the optimization, as the previous target function value is not necessarily valid anymore.

4.5 Adaptation of Composite Services

Finally, selected adaptations have to be applied to the service composition. This is the task of the Composition Adaptor component, implementing the **E**xecution step of the MAPE loop. The Composition Adaptor relies to a large part on VRESCo [129, 130] as a service runtime environment and DAIOS [110] as dynamic Web service invoker to carry out various adaptations. In the following, a general overview over the concept of adaptation in PREvent is given. Afterwards, contributions produced as part of this thesis on each type of adaptation (service rebinding, service rebinding with interface mediation, and structural adaptation using fragment substitution) are presented.

Definition of Adaptation Actions

In PREvent, adaptation actions are defined using the elements depicted in Figure 4.16. Just like metrics and checkpoints before, adaptation actions are defined in an XML syntax. In this thesis, it is generally assumed that possible adaptation actions are well-known, i.e., no contributions have been made in the area of identifying what adaptations are actually possible. Potential adaptation actions are stored in the Adaptation Actions Database.

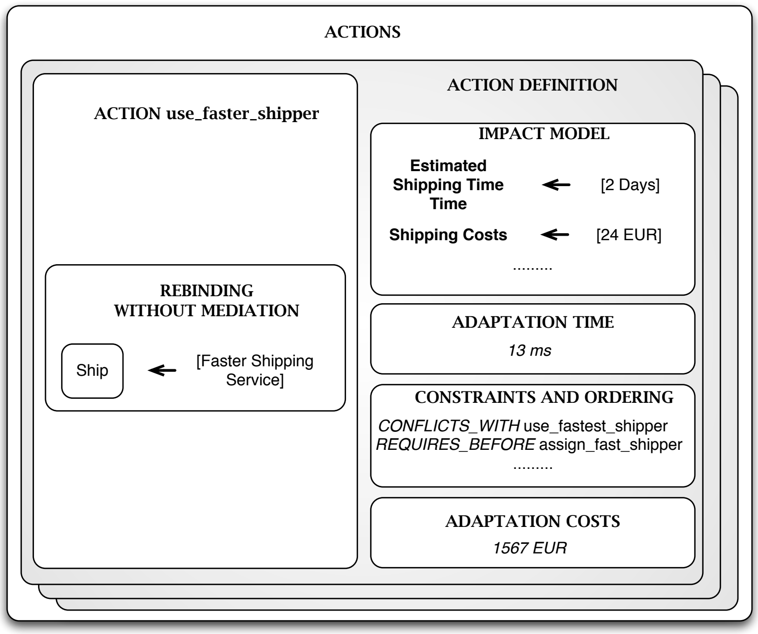

ACTIONS
ACTION use_faster_shipper
ACTION DEFINITION
IMPACT MODEL
Estimated Shipping Time Time
[2 Days]
Shipping Costs
[24 EUR]
....
REBINDING WITHOUT MEDIATION
Ship
[Faster Shipping Service]
ADAPTATION TIME
13 ms
CONSTRAINTS AND ORDERING
CONFLICTS_WITH use_fastest_shipper
REQUIRES_BEFORE assign_fast_shipper
....
ADAPTATION COSTS
1567 EUR

Figure 4.16: Definition of Adaptation Actions

As shown in Figure 4.16, any number of adaptation actions can be defined. Each action definition contains the description of the actual action (one of the types described in Section 4.5 below). In addition, the following information needs to be provided for each action.

Impact Model

The impact model contains a set of impact clauses. An impact clause defines the concrete impact on one monitorable metric caused by applying a certain adaptation action. Essentially, the clauses model updates to the data used to generate predictions. Every adaptation action can have any number of positive as well as negative impacts on any fact or estimate. A business analyst can determine the impact clauses of an adaptation action in several ways:

- If SLAs with providers of external Web services exist, these SLAs can form the basis of establishing impact clauses. For instance, if a shipping provider used in the ACMEBOT case has an established SLA guaranteeing a shipping time of 24 hours, one can assume as an impact clause `SHIPPING_TIME` $\rightarrow$ `24 hrs`.
- If the action has been applied in the past, data mining techniques [54, 192] can be used to figure out the impact of the adaptation on composition metrics. This is more suitable for complex dependencies, which are not easy to identify for humans. However, a potentially large number of executions needs to be available to identify a correct impact model in a reliable way. Additionally, this method evidently has a bootstrap problem, i.e., it is not possible to use data mining to determine the impact of an action before it has been executed the very first time.
- Finally, impact clauses can also be calculated using the methods of QoS and SLA aggregation [84, 85, 174]. This technique is most suitable to identify the impact of structural adaptation (see Section 4.5 below) on QoS metrics.

It is assumed that the impact model specifies impact clauses for all metrics which the action affects. Evidently, impact clauses do not need to be exact (very often it will realistically be impossible to statically define an exact impact model before execution), however, more exact impact models lead to better optimization, which in turn leads to a better end-to-end performance of the PREvent system.

Adaptation Time

Similar to the impact model, it is also necessary to define an estimation of the time that the technical application of the adaptation action will take (referred to as d_x of an adaptation action $x \in A$ in Section 4.4). This adaptation time is necessary to calculate the decision point for the action, which is required if stepwise optimization should be used. The adaptation time can be acquired relatively easily if the adaptation has been used before, e.g., by calculating the arithmetic mean of the adaptation time of all historical executions of the adaptation. If the action has not yet been used, a pessimistic estimation is advisable, as it is usually better to decide too early on an action than to decide too late.

Constraints and Ordering

As already discussed in Section 4.4, not all adaptation actions are necessarily independent. If actions depend on or conflict with each other, this needs to be specified as part of the action definition. Such exclusion and ordering constraints can be defined using five different order predicates (REQUIRES_BEFORE, REQUIRES_AFTER, IF_PRESENT_BEFORE, IF_PRESENT_AFTER, and CONFLICTS_WITH). REQUIRES_[BEFORE|AFTER] specifies that a given action has to be applied before or after another action (otherwise the action cannot be applied at all). IF_PRESENT_[BEFORE|AFTER] specifies that if another action is present, it has to be applied before or after this one (but the other action can also simply not be applied). Another type of ordering predicate is CONFLICTS_WITH. This predicate specifies that two actions are mutually exclusive, i.e., they cannot be applied together.

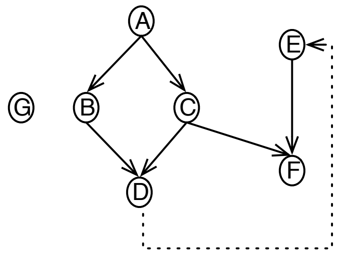


Figure 4.17: Topological Ordering of Adaptation Actions

At runtime, a forest of directed graphs is constructed from these predicates, whose nodes are actions and whose edges are "needs to be executed after" relationships. If the graphs in this forest are acyclic, there is at least one allowed order of actions, which can be consructed using topological ordering [91]. The topological order can be constructed in linear time. However, if the graphs in this forest contain at least one cycle, the definition of adaptation actions is invalid, and needs to be fixed. This is outside the scope of this thesis. An example is depicted in Figure 4.17. The action definition is legal as long as the dashed dependency is not specified. Including the dashed dependency, a cycle is constructed, and there is no valid execution order for these actions.

Adaptation Costs

As already stated in Section 4.4, it is assumed that every adaptation action has a constant, non-negative cost. For example, the cost of using a faster shipping service in ACMEBOT is the cost of using the new service minus the costs of using the original shipping service. These costs need to be well-known and, hence, are also specified as part of the action definition.

Taxonomy of Adaptation Actions

In Figure 4.18, a simple taxonomy of possible adaptation actions is given. On a primitive level, four types of actions can be distinguished (roughly in the order of the severity of their impact on the composition): data manipulation actions, service rebinding actions, structural adaptation actions, and environmental actions. As part of this thesis, work has been focused mainly on service rebinding and structural adaptation.

Data Manipulation is the most simple type of adaptation. In this case, the service composition is in fact not changed. Instead, the data flow of the composition is intercepted and adapted.

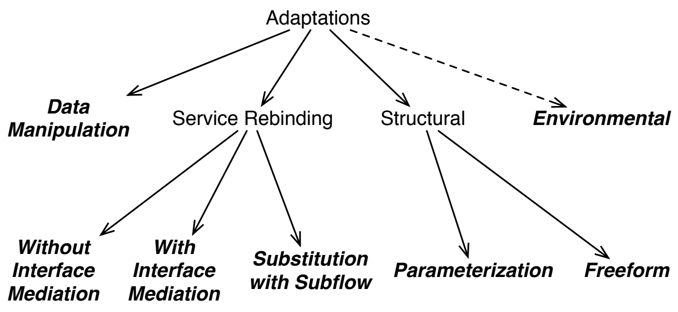


Figure 4.18: Taxonomy of Adaptation Actions

Data manipulation actions are relatively simple to implement and do not pose particular research challenges.

More complex than data manipulation is *Service Rebinding*, i.e., binding an activity in the service composition to a different Web service. Service Rebinding can be further broken down as follows:

- *Service Rebinding Without Interface Mediation* - this is the most simple case of service rebinding, where one service implementation is exchanged for another with the very same service interface. This simple rebinding case is in fact already covered in some specifications of composition languages. For instance, WS-BPEL [140] supports this using the construct of dynamic partner links[8]. The basic requirement for supporting service rebinding is that service client and service provider are not tightly coupled, and that the binding can be changed without restarting, or, even worse, re-implementing the service client. In PREvent, dynamic binding is implemented in the DAIOS dynamic Web service invocation framework.

- *Service Rebinding With Interface Mediation* - in this case, there is still a one-to-one exchange of service invocations, but the interfaces of the previously used and the new service differ. In this case, some mediation between different operations and data structures needs to take place. Possible solutions to this problem include semantic Web service technology [120] such as SAWSDL [198] or the application of transformation rules, e.g., Extensible Stylesheet Language Transformations (XSLT) transformations as often used in ESB solutions. In PREvent, mediation is based on the rich VRESCo metadata model [160]. An extension to DAIOS has been developed that uses this metadata model to enable interface mediation without the need for semantic Web technology.

- *Substitution With Subflow* - finally, one can also think of cases were a single service invocation needs to be exchanged for multiple invocations of different operations. These rebinding scenarios are the most complex ones, as they essentially change the structure of the composition itself (after adaptation, the abstract activities of the composition have

[8]http://download.oracle.com/docs/cd/E19182-01/821-0539/dynamic_addr/index.html

changed, not only the binding of activities to services, as before). For this reasons, we discuss this special case along with structural adaptations in the remainder of this thesis.

Additionally, structural adaptation can also be broken down into two separate classes:

- *Parameterization* - parameterization is a simplified version of structural adaptation. The general idea of parameterization has initially been presented in [93] for WS-BPEL processes. Essentially, the idea is that service compositions already contain a number of "variants" of the same composition, and can be configured to use a certain variant at runtime. Essentially, this boils down to enabling and disabling different parts of the composition. Variants not anticipated at composition design time cannot be covered using the parameterization approach.

- *Freeform* - actual freeform structural adaptation is much more powerful. In freeform adaptation, activities and control structures can be added, removed and modified at will. All adaptations discussed so far can (also) be expressed as freeform structural adaptations.

This is not the case for *Environmental Adaptations*. This type of adaptation considers adaptation of the execution environment, such as upgrading the hardware of the machine running the service composition. Evidently, this adaptation is on an entirely different level than all other adaptation actions that have been discussed so far. Most importantly, environmental adaptations can only be applied to all composition instances at once (it is not possible to upgrade the server for just one instance alone). Environmental adaptations are not discussed further in this thesis, However, the current trend of Cloud Computing [5, 113] enables environmental adaptations, if the service composition is hosted in a cloud. A more detailled analysis of this promising direction remains open for future research work.

```
<repairaction>
  <dataManipulationAction
    targetActivity="ship"
    message="input"
    messagePath="request/UseExpressShipping">
    <constant type="boolean">true</constant>
  </dataManipulationAction>
</repairaction>
```

Listing 4.9: Data Manipulation Action

In Listing 4.9, a simple data manipulation action is defined. The input message of the activity "ship" is intercepted, and the flag which indicates if express (e.g., overnight) shipping should be used is set to the constant value "true". More complex data manipulations can also copy or calculate values from other messages in the workflow. The message path is again used to reference body content of events, similar to the principles used in Section 4.2 for the definition of metric monitors.

Listing 4.10 displays how constraints, costs and the impact model are specified. Lines 9 to 18 show how a single impact clause (for the metric `SHIPPING_TIME`) is defined. Note that the concept of estimators (see Section 4.3 for details) has been reused to technically implement impact models. Basically, the assumption is that the value of a metric after adaptation is also an

```
<action name="USE_EXPRESS_SHIPPING">
  <repairaction>... </repairaction>
  <costs>
    <constant>1567</constant>
  </costs>
  <constraints>
    <conflictswith>USE_STD_SHIPPING</conflictswith>
  </constraints>
  <improvement>
    <improvedMetric>SHIPPING_TIME</improvedMetric>
    <improvementEstimate name="estimatedShippingTime"
      type="integer">
      <estimatorClass
        class="Estimator.ConstantEstimator"/>
        <argument value="1"/>
      <estimatedField name="SHIPPING_TIME"/>
    </improvementEstimate>
  </improvement>
</action>
```

Listing 4.10: Improvements and Constraints

estimate, and should be treated as such. In Listing 4.10, the estimation is that the shipping time will be exactly one day after adaptation. More complex estimations need to be implemented in specific estimators.

In the following, a detailed description of the most essential types of actions (service rebinding without interface mediation, service rebinding with mediation and freeform structural adaptation) within the PREvent framework is given. Data manipulation is available in PREvent, but will not be covered explicitly because of the inherent simplicity of this type of action.

Service Rebinding Without Mediation

While service rebinding without mediation does not look very challenging from the outside, it is still surprisingly hard to implement with existing tooling. For instance, in composition languages like WS-BPEL or WWF, activities are usually hardwired to concrete service endpoints. In WS-BPEL, partner links are associated with WSDL contracts. It is possible to rebind partner links to different service implementations at runtime, but the new service needs to implement exactly the same WSDL, except for the endpoint addressing. Evidently, this is infeasible in all but the most trivial cases. Realistically, even in relatively simple cases, some small differences, like differently named operations or parameters, different namespaces, or different ordering of messages will be evident. If the new service simply implements the same functionality using a RESTful service interface [56, 147], standard WS-BPEL is out of luck altogether and other means like “BPEL for REST” [146] need to be used. In WWF, activities are just as hard-wired to concrete WSDLs, disallowing any changes to the binding without going through the WWF WorkflowChanges API. Hence, as early work on this thesis, the DAIOS dynamic service invocation framework has been proposed. DAIOS forms the basis of service rebinding in PREvent.

DAIOS

The DAIOS framework has been developed with the high-level goal of providing a tool which allows for truly decoupled service clients and services. To this end, DAIOS allows to invoke services in a more dynamic manner, as compared to state-of-the-art Web service toolkits, such

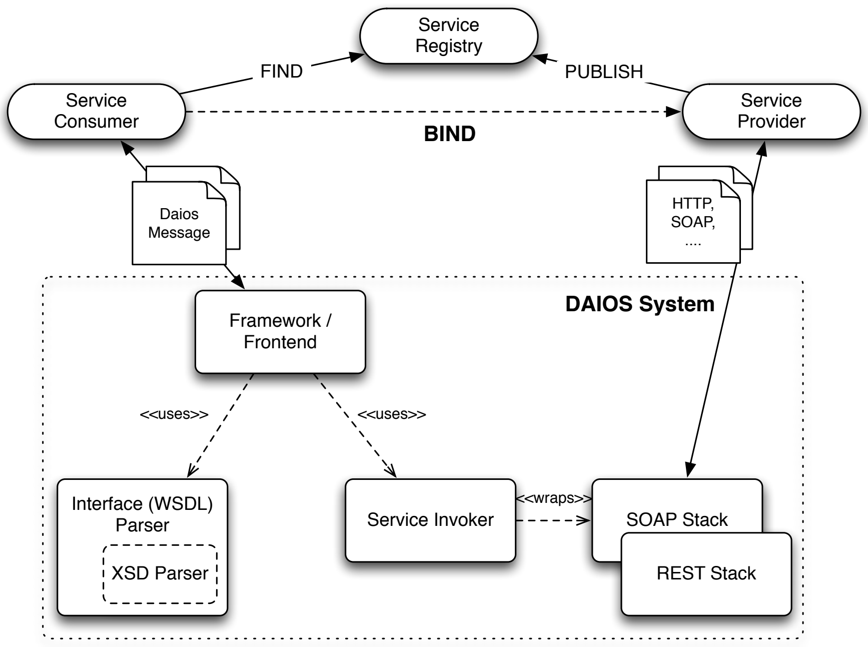


Figure 4.19: DAIOS Architecture

as Apache Axis 2[9] or Apache CXF[10]. Even though these frameworks nowdays provide Dynamic Invocation Interfaces (DIIs) to allow for more or less limited dynamic invocations, the primary modus operandi is still to generate stubs from WSDL contracts and embed these stubs deeply into client applications. Contrary, in DAIOS, the primary means of invocation is the DII. Furthermore, DAIOS is purely message-driven, as opposed to the RPC-centric Apache frameworks. Finally, DAIOS allows to abstract away from primitive interface differences, such as endpoints, operation and parameters names or parameter ordering. DAIOS even allows to replace RESTful and traditional Web services transparently.

Figure 4.19 sketches the general architecture of the DAIOS framework, and how it fits into the triangle of publish, find and bind, that loosely-coupled SOAs are based on [145]. The framework is divided internally into three functional components: the general DAIOS classes, which are at the core of the framework and orchestrate the individual other components, the interface parsing component which is responsible for preprocessing (binding) and the invoker component which conducts the actual Web service invocations using a concrete Representational State Transfer (REST) or SOAP stack. The general structure of the framework is an implementation of Composite Pattern for Stubless Web Service Invocation (CPWSI) [24]. CPWSI separates the frameworks' interface from the actual invocation backend implementation, and allows for flexibility and adaptability.

DAIOS is grounded on the notion of message exchange. Clients communicate with services

[9]http://axis.apache.org/axis2/java/core/
[10]http://cxf.apache.org/

by passing messages (so-called DAIOS messages) to them. Services return the invocation result by answering with messages. DAIOS messages are potent enough to encapsulate XML Schema complex types, but much simpler to use than working directly on XML level. Messages are unordered lists of name-value pairs, referred to as message fields. Every field has an unique name, a type and a value. Valid types are either built-in types (*simple field*), arrays of built-in types (*array field*), complex types (*complex field*) or arrays of complex types (*complex array field*). Such complex types can be constructed by nesting messages – arbitrary data structures can therefore be built easily, without the need for a static type system. This structure bears some similarity to the JavaScript Object Notation (JSON) format[11], which has gained interest in the Web 2.0 community in the last years.

Using DAIOS is generally a three-step procedure:

1. First and foremost, clients have to find a service that they want to invoke (service discovery phase). This step is external to DAIOS. The service discovery problem is mostly a registry issue and has to be handled separately. In the context of PREvent, this step is not relevant, as service selection is essentially part of the definition of adaptation actions. If used outside of PREvent, the DAIOS framework is very well integrated with VRESCo, i.e., it is rather easy to discover a service in the VRESCo service registry and directly invoke it using DAIOS. This combined usage of DAIOS and VRESCo is nicely demonstrated in [79].

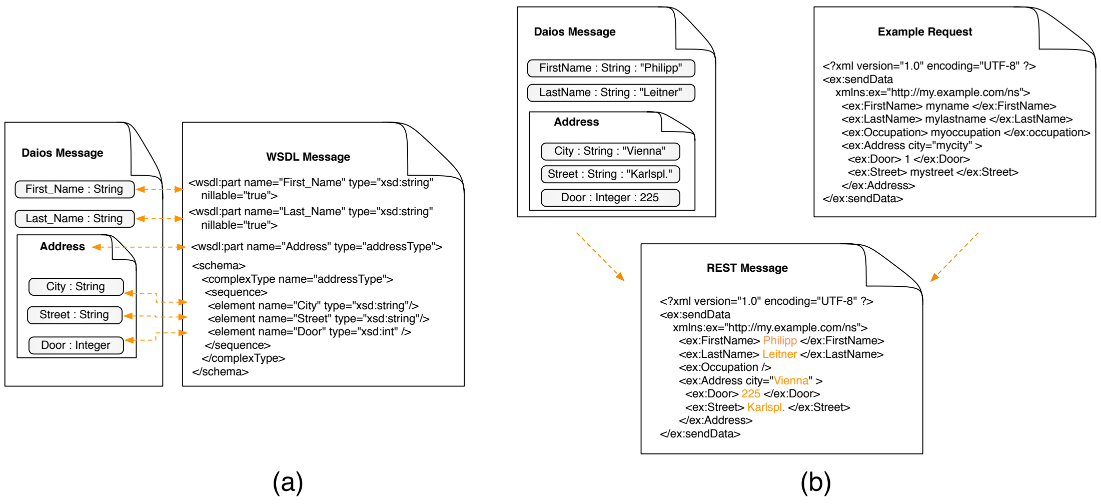
2. In the second step the service has to be bound (preprocessing phase). During this phase, the framework collects all necessary internal service information, e.g., for traditional Web services the WSDL interface will be compiled in order to obtain endpoint, operation and type information. For RESTful services, DAIOS may parse an example request (see below) to aquire necessary invocation information.

3. The final step is the actual invocation of the service (dynamic invocation phase). In this phase, the input (i.e., an input message, represented as a DAIOS message) will be converted into the encoding expected by the service (for instance a SOAP XML document for a traditional Web service, or a HTTP GET request for REST), and the invocation will be launched using a SOAP or REST service stack. When the invocation response (if any) is received by the service stack, it will be converted back into an output message and returned to the client.

Once a service is successfully bound, clients can of course issue any number of invocations without having to re-bind again. Service bindings only have to be renewed if the interface of the service changes or the client explicitly decides to release the binding for some reason.

Most of DAIOS' important processing happens in the dynamic invocation phase. For a SOAP invocation, the framework will analyze the given input and determine which WSDL input message the provided data matches best. For this, the framework relies on a specific similarity algorithm for DAIOS and WSDL messages. The general idea of this algorithm is to calculate a structural distance metric for the WSDL message and the user input, i.e., count how many parts

[11] http://www.json.org/



```
<wsdl:part name="First_Name" type="xsd:string"
   nillable="true">
<wsdl:part name="Last_Name" type="xsd:string"
   nillable="true">
<wsdl:part name="Address" type="addressType">
<schema>
  <complexType name="addressType">
   <sequence>
    <element name="City" type="xsd:string"/>
    <element name="Street" type="xsd:string"/>
    <element name="Door" type="xsd:int" />
   </sequence>
  </complexType>
</schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<ex:sendData
   xmlns:ex="http://my.example.com/ns">
     <ex:FirstName> myname </ex:FirstName>
     <ex:LastName> mylastname </ex:LastName>
     <ex:Occupation> myoccupation </ex:occupation>
     <ex:Address city="mycity" >
       <ex:Door> 1 </ex:Door>
       <ex:Street> mystreet </ex:Street>
     </ex:Address>
</ex:sendData>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<ex:sendData
   xmlns:ex="http://my.example.com/ns">
     <ex:FirstName> Philipp </ex:FirstName>
     <ex:LastName> Leitner </ex:LastName>
     <ex:Occupation />
     <ex:Address city="Vienna" >
       <ex:Door> 225 </ex:Door>
       <ex:Street> Karlspl. </ex:Street>
     </ex:Address>
</ex:sendData>
```

Figure 4.20: Matching DAIOS Inputs to Service Interfaces

in a given WSDL message have no corresponding field in the DAIOS message, whereas lower values represent a better match. If there are fields in the user message that have no corresponding field in the WSDL message, the similarity is defined to be ∞. DAIOS will choose to invoke the operation whose input messages has the best (i.e., lowest) structural distance metric to the provided data. Only if two or more input messages are equally similar to the input, the user has to specify the operation to use. If no input message is suitable at all, that is, if all input messages have a similarity metric of ∞ to the input, an error will be thrown. In that casem the provided input is simply not suitable for the chosen Web service. Otherwise, the framework will convert the input into an invocation of the chosen operation, issue the invocation, receive the result from the service and convert the result back into a message. The backend used to conduct the actual invocation is replaceable. The current version of DAIOS comes with two options of invocation backends, one which uses the Apache Axis 2 stack and one which utilizes a custom-built ("native") stack. DAIOS also puts much emphasis on client-side asynchrony. All invocations can be issued in a blocking or non-blocking fashion. More conretely, DAIOS supports the four common synchrony patterns "Blocking Invocation", "Fire and Forget Invocation", "Poll Object Invocation" and "Callback Invocation" [178, 179]. Hence, DAIOS implements an asynchronous Web service invocation framework as proposed in [207]. DAIOS' matching procedure abstracts away most of the Remote Procedure Call (RPC)-like internals of SOAP and WSDL. The client-side application does not need to know about operations, messages, endpoints or encoding. Even whether the target service is implemented as SOAP- or RESTful service is, to a certain degree, transparent to the client, even though clients need to know the endpoint address in case of RESTful services. All of these service details are handled solely by DAIOS, allowing the client application to be as generic as possible.

For RESTful invocations the procedure is slightly different. Interface descriptions for RESTful services exist (for instance Web Application Description Language (WADL) [195]), but are not commonly used. Hence, for these invocations, the user may either specify an example request instead of the WSDL interface, or give no further details on the service interface at all. If an example request is given then DAIOS will use the example as a template, which is filled with the actual input from the user at invocation time, and issue an HTTP POST request with the filled template as payload. If no information about the service interface is given (i.e., neither WSDL description nor example request) then a HTTP GET request with URL-encoded parameters will be issued.

Figure 4.20 sketches the matching of inputs to a WSDL description and an example request. Subfigure (a) shows a DAIOS message and a WSDL message in `RPC/encoded` style with a structural distance of 0 (a perfect match). If, for instance, the field "First_Name" would be removed from the DAIOS message, the structural distance would increase to 1. Subfigure (b) details how an example template is filled with user input provided as DAIOS message in case of a REST invocation.

```
 1 // create a Daios backend
 2 ServiceFrontendFactory factory =
 3     ServiceFrontendFactory.getFactory
 4       ("at.ac.tuwien.infosys.dsg.daiosPlugins."+
 5       nativeInvoker.NativeServiceInvokerFactory");
 6
 7 // preprocessing - bind service
 8 ServiceFrontend frontend = factory.createFrontend(
 9   new URL(
10     "http://vitalab.tuwien.ac.at/"+
11     "orderservice?wsdl"));
12
13 // construct input that we want
14 // to pass to the service
15 DaiosInputMessage registration
16   = new DaiosInputMessage();
17 DaiosMessage address = new DaiosMessage();
18 address.setString("City", "Vienna");
19 address.setString("Street", "Argentinierstrasse");
20 address.setInt("Door", 8);
21 registration.setComplex("Address", address);
22 registration.setString("First_Name", "Philipp");
23 registration.setString("Last_Name", "Leitner");
24
25 // dynamic invocation
26 DaiosOutputMessage response =
27   frontend.requestResponse(registration);
28
29 // retrieve result
30 String regNr = response.getString("registrationNr");
31 // ...
```

Listing 4.11: DAIOS SOAP Invocation

The message-based client API is very easy to use for the client programmer. Listing 4.11 displays the Java code necessary to invoke a WSDL-based Web service with non-trivial interface and complex data types. The message constructed in this example corresponds to the structure depicted in Figure 4.20. Note that, even though the target service uses nested data structures (registrations contain address data), DAIOS does not need any static components such as data transfer objects. All necessary service and type information is collected during the preprocessing phase (lines 8 to 11). When the actual dynamic invocation is fired (lines 26 and 27), the framework will use this information and convert the user-provided input to a concrete Web service invocation. In this example, a blocking invocation style is used, but asynchronous communication is handled widely identically. It is important to note that no SOAP or WSDL specifics, such as operation name, endpoint address, or the WSDL encoding style used, have to be specified by the client application. DAIOS abstracts from all these service internals and exposes a uniform interface, thereby enabling loose coupling between client and service.

```
String myAPIKey = ... // get an API key from Flickr

// use the native backend
ServiceFrontendFactory factory =
    ServiceFrontendFactory.getFactory
      ("at.ac.tuwien.infosys.dsg.daiosPlugins."+
      nativeInvoker.NativeServiceInvokerFactory");

// preprocessing for REST
ServiceFrontend frontend = factory.createFrontend();

// setting the EPR is mandatory for REST services
frontend.setEndpointAddress(
  new URL("http://api.flickr.com/services/rest/"));

// construct message
DaiosInputMessage in = new DaiosInputMessage();
in.setString("method",
  "flickr.interestingness.getList");
in.setString("api_key", myAPIKey);
in.setInt("per_page", 5);

// do blocking invocation
DaiosOutputMessage out =
  frontend.requestResponse(in);

// convert WS result back
// into some convenient Java format
DaiosMessage photos =  out.getComplex("photo");
// ...
```

Listing 4.12: DAIOS REST Invocation

Listing 4.12 exemplifies the invocation of a RESTful Web service. In this listing, the widely known Flickr REST API[12] is accessed and a list of hyperlinks to the most “interesting” photos is retrieved. RESTful and traditional Web services are invoked through the same interface. The code necessary to access the service is practically identical for both types of services. The main difference is that no interface definition language similar to WSDL has yet been widely accepted for REST, leading to the problem that the user is forced to specify more service details for such invocations (the endpoint address in the example).

Dynamic Binding in VRESCo

One way to use DAIOS is demonstrated by the VRESCo SOA runtime environment. The typical way of invoking a service in VRESCo is through so-called service proxies. Basically, a service proxy is a client-side abstraction of a remote service, which can be incorporated into client applications or service compositions. Service proxies come in two fashions. Fixed proxies are statically bound to a single service instance, and can invoke only this service. This is very similar to the traditional approach of generating stubs from WSDL contracts. However, more important in VRESCo are rebinding proxies. Rebinding proxies are generally not bound to a concrete service instance, but to a query on the VRESCo service registry. Actual invocations using these proxies invoke the best matching result returned by the registry. Queries are specified using the VRESCo Query Language (VQL). VQL provides a query language to access all information

[12] http://www.flickr.com/services/api/

stored in the VRESCo registry (i.e., services and service metadata, including QoS). Queries are specified in a way similar to Hibernate's Criteria queries[13]. In general, VQL queries consist of six elements:

- The *Return Type* R defines the expected data type of the query results. The return type needs to be an element of the VRESCo metadata model (see [160] for details on the VRESCo metadata model).
- The *Mandatory Criteria* C_m describe constraints which have to be fulfilled by the query (e.g., response time must be less than 500 ms).
- The *Optional Criteria* C_o add constraints which should optimally be fulfilled but are not required (e.g., service provider should be company X).
- The *Ordering* O can be used to specify the ordering of the query results (e.g., sort ascending by ID).
- The *Result Limit* L can be used to restrict the number of results (e.g., 10, or 0, which represents no limit).
- The *Querying Strategy* S finally defines how the query should be executed (e.g., exact or fuzzy matches).

The most important elements are criteria since they actually represent the constraints of the query. Criteria consist of a set of *expressions* E that are used to define common constraints such as comparison (e.g., smaller, greater, equal, etc.) and logical operators (e.g., AND, OR, NOT, etc.).

```
var query = new VQuery(typeof(ServiceRevision));

// add query criteria
query.Add(Expression.Eq("IsActive", true));
query.Add(Expression.Eq("Service.Category.Features.Name",
                        "NotifyCustomer"));
query.Match(Expression.Eq("Service.Owner.Company",
                          "CompanyX"), 1);
query.Match(Expression.Like("Tags.Property.Name",
    "STABLE", LikeMatchMode.Start), 3);
query.Match(
    Expression.Eq("QoS.Property.Name", "ResponseTime") &
    Expression.Lt("QoS.DoubleValue", 1000.0), 5);

// execute query
var querier = VRESCoClientFactory.CreateQuerier(
    "username", "password");
var results = querier.FindByQuery(query, 10,
    QueryMode.Priority) as IList<ServiceRevision>;
```

Listing 4.13: VQL Sample Query

Listing 4.13 shows an example query. As described above, queries are parameterized using the expected return type, in this case, the type `ServiceRevision` (a concrete version of a

[13]http://docs.jboss.org/hibernate/core/3.3/reference/en/html/querycriteria.html

service). In this example, two Add criteria (lines 4–6) are used to state that services have to be active and that each service has to implement the Notify_Customer feature (features are the VRESCo way of representing some abstract functionality). Additionally, three Match criteria are added in the example (lines 7–13). The first criterion expresses that services provided by *CompanyX* are preferred, while the second criterion defines that revisions should have tags starting with "STABLE" (Like expression). The third criterion specifies an optional QoS constraint on response time, which should be less than 1000 ms. The operator '&' in line 12 represents a shortcut for an And expression. All three Match criteria use priority values as third parameter to define the importance of a criterion. The query is finally executed (lines 16–19) by instantiating a querier object, and invoking the FindByQuery method. Furthermore, the result limit of the query is set in order to return only 10 results.

VQL queries, such as the one depicted in Listing 4.13, can be used to construct a rebinding proxy. Basically, a rebinding proxy needs as input only a query to select a service (typically, these queries will include QoS attributes and an ordering clause), and a strategy when the query should be re-evaluated at runtime. All available strategies are summarized in Table 4.3.

Strategy	**Proxy Reconsiders Binding...**
Fixed	...never
Periodic	...periodically
OnDemand	...on client requests
OnInvocation	...before every service invocation
OnEvent	...on event notifications

Table 4.3: Rebinding Strategies

All rebinding strategies have their advantages and disadvantages. *Fixed* proxies are used in scenarios where rebinding is not needed (e.g., because of existing contractual obligations). *Periodic* rebinding causes background queries on a regular basis, which is inefficient if invocations happen infrequently. *OnDemand* rebinding results in low overhead but has the drawback that the binding is not always up-to-date. In contrast to this, *OnInvocation* rebinding guarantees accurate bindings, but seriously degrades the service invocation time, as service bindings are checked before every invocation. Finally, *OnEvent* rebinding uses the VRESCo event engine to combine the advantages of all strategies. Therefore, clients use subscriptions for defining in which situations to rebind, which is then triggered by events. However, for this strategy to work, clients need to be able to receive event notifications.

For the WWF service composition language, a set of custom VRESCo activities have been implemented. Using these custom activities, VRESCo-enabled service invocations can be easily integrated into regular WWF compositions. In Chapter 5, the ACMEBOT case study used as foundation of the evaluation has been implemented on top of these custom VRESCo activities.

Service Rebinding With Mediation

Service rebinding with mediation is the more general case of service rebinding without mediation, as discussed above. Hence, the same problems and solutions are in principle applicable.

Of course, if interface mediation is necessary, further techniques are necessary to resolve those interface differences.

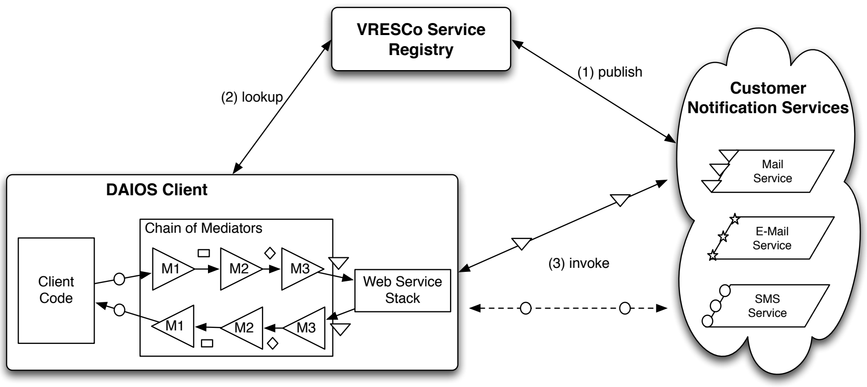


Figure 4.21: Client-Side Interface Mediation Architecture

In this section, the VRESCo Mapping Framework (VMF), which handles the mediation between different services that implement the same feature using different interfaces, is described. VMF is built as an extension of the DAIOS framework. To this end, VMF implements a VRESCo-specific interface mapping language as an interceptor of the DAIOS invoker. This technique can be easily integrated, as DAIOS internally uses the concept of a *chain of mediators* to implement the mapping of DAIOS messages to XML (see Figure 4.21). The chain of mediators implements a stepwise transformation from the original input to the format expected by the target service. VMF is implemented as an VRESCo-specific mediator, which is inserted into the chain of mediators if service rebinding with interface mediation is necessary. The custom WWF activities mentioned in Section 4.5 make use of VMF to implement interface mediation based on VRESCo.

In Figure 4.21, the client expects to invoke the “SMS Service” but is bound to the “Mail Service” instead. In the chain of mediators, the client input is now transformed from the format expected by the “SMS Service”, via two intermediary formats, into the representation needed by the “Mail Service”. VMF implements one of these transformation steps. Another mediation step is usually the conversion of DAIOS messages to XML. Further custom mediation steps can be inserted as additional interceptors by the client application. VMF is tighly integrated with the metadata model used by VRESCo. Hence, this metadata model is now briefly introduced.

VRESCo Metadata Model

VRESCo uses a custom metadata model to describe functionalities offered by Web services in an abstract way. This model has been developed as a light-weight and easier-to-use alternative to semantic Web services [120] technology. Instead of using a strong formal model and semantic Web ontologies [118], VRESCo utilizes simple strings and primitive data types to name concepts and functionality. This is made possible by abandoning the open world assumption of semantic Web services [8]. Instead, the assumption in VRESCo is that services are mapped by clients to their respective internal instance of the metadata model. Essentially, this means that clients decide which business functions services fulfill within their company.

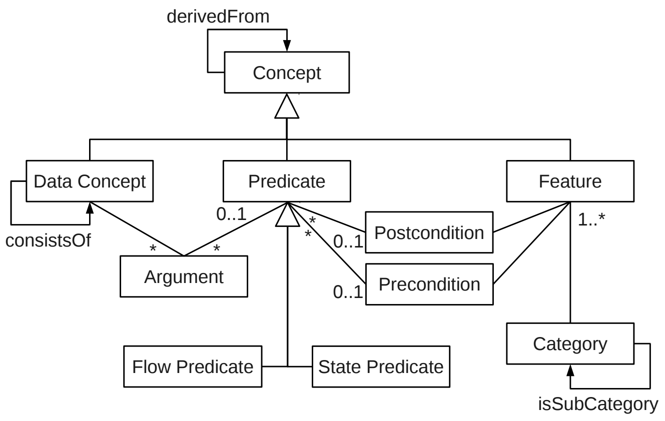


Figure 4.22: VRESCo Service Metadata Model

Figure 4.22 depicts the most important elements of the VRESCo metadata model. A category is a named entity that describes the general purpose of services and serves as an umbrella item for all other entities in that domain (e.g., the category of all services in the ACMEBOT scenario are named `ACMEBotSystem`). A category contains an arbitrary number of features, which describe concrete activities that can be carried out in the domain of the category. In the ACMEBOT case, features can include `Send Notification to Customer` or `Check Warehouse for Product`. The base element in the metadata model is the abstract concept, which is specialized by the three sub-entities data concept, predicate and feature. All concept entities are composable, i.e., a concept can be derived from another concept. Data concepts define the data types that exist in the system and are either atomic (string, number, etc.) or composed of other data concepts.

Additionally, the metadata model allows for the definition of preconditions and postconditions that need to be fulfilled when a feature is executed. Both types of conditions are associated with a number of predicates, which may each have multiple arguments. The arguments are described by an associated data concept. Two types of predicates are distinguished: flow predicates indicate constraints related to the data flow such as data required or produced by a feature (predicates `requires` and `produces`); state predicates express global constraints that need to be fulfilled.

Concrete service operations need to be mapped to features in the metadata model. With this mapping, a client defines what functionality a service fulfills within her domain. This mapping is sketched in Figure 4.23. The service model constitutes the information about concrete services managed by VRESCo. A service is available in one or more revisions. Revisions are the basis for service versioning, which is out of scope of this thesis and is discussed in [107]. Services are mapped to categories, a service operation is one concrete implementation of a feature (note that one feature may be implemented by several different operations) and operation parameters are mapped using data concepts. The mapping function between the latter is imple-

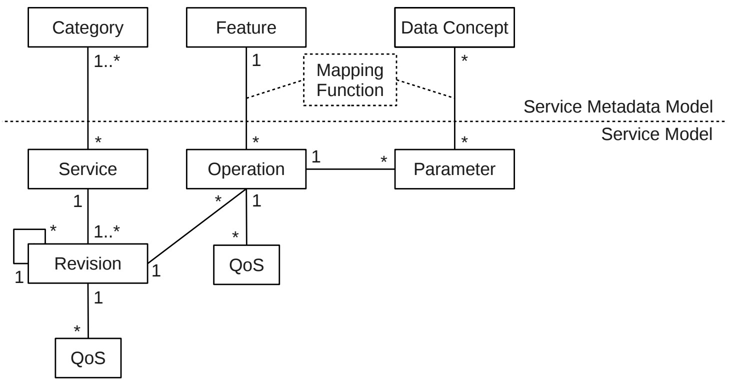


Figure 4.23: Mapping of Concrete Services to the Metadata Model

mented using VMF. In addition to the functional characteristics of services, VRESCo also saves non-functional attributes in the form of QoS attributes. A more detailed discusion of QoS data is omitted here.

VRESCo Mapping Framework (VMF)

As indicated above, as part of a service's mapping to the VRESCo metadata model, one can also specify a mapping function. These mapping functions are the basis of VMF and, hence, of the PREvent framework's approach to dealing with service rebinding with interface mediation.

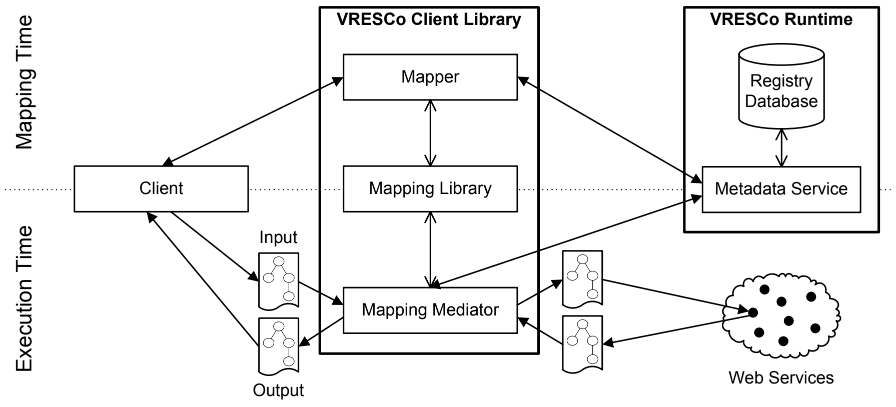


Figure 4.24: VMF Architecture

Figure 4.24 shows an overview of the architecture of the VMF mediator. Generally, VMF comprises two main components. Firstly, at mapping time, the Mapper component is used to create mapping scripts for each service. This information is stored in the VRESCo registry using the VRESCo metadata service. Secondly, at execution time, DAIOS is used as a dynamic service invocation framework. The Mediator component is used as an interceptor in DAIOS, as described above. This mediator retrieves the mapping scripts from the Metadata Service at runtime, and executes the corresponding mapping. This is done by applying all defined mapping functions sequentially, in the order they have been specified. In that sense, VMF implements an imperative, interpreted domain-specific language [122].

Functions	Description
Assign	Link one parameter to another (source and destination must have the same data type)
Constants	Define simple data type constants
Conversion	Convert simple data types to other simple data types
Array	Create arrays and access array items
String	String manipulation operations (e.g., `substring`, `concat`)
Math	Basic mathematical and logical operations (e.g., `addition`, `round`, `and`, `or`)
CSScript	Define complex mappings directly as CSScripts (basically, C# program code)

Table 4.4: VMF Mapping Functions

Mapping scripts are defined using a Mapping Library, which includes a number of predefined *Mapping Functions*. Mapping functions are the atomic building blocks, from which all mapping scripts are constructed. The provided mapping functions are summarized in Table 4.4 (grouped into 7 categories). Probably the most important function is `Assign`, which is used to map one input parameter or intermediary result to an output parameter. Functions from the *Constants* group are used to create new data directly in the mapping. All remaining mapping functions are used to transform parameters in various ways, e.g., from one data type to another, using string manipulation, or using mathematical and logical operations. Furthermore, arbitrarily complex mappings can be defined using the CS-Script language [165]. Essentially, this allows to deploy custom mapping functions by using the full power of the C# programming language. For instance, this can be used to invoke external Web services at mediation time.

A concrete mapping example is given in Figure 4.25. The feature `Notify_Customer` from the ACMEBOT case study is mapped to the service operation `SendSMS1`. The feature provides three input parameters and produces one output parameter. The parameter `Message` is identical in both interfaces. Hence, it can be mapped directly (using only `Assign`). Note that, for the `Assign` function to work, both sides need to be represented using the same data concept (in this case string). The parameter `SenderNr` is split into the area code and the actual number. This is done using the string operation `SubString`, which takes the start index of the string and the length of the substring as parameters. Afterwards, both substrings are converted to integers using the `ConvertToInt` function. This is necessary as assigning a string to an integer is not possible. The `ReceiverNr` is handled similarly. So far, only input parameters have been mapped (i.e., all information given so far forms the lowering script for this service). The lifting script, which defines how the service output is mapped to the feature output, consists only of a `ConvertToBoolean` and another `Assign` function.

Freeform Structural Adaptation

After covering service rebinding adaptation with and without mediation, the following section will discuss the adaptation types substitution with subflows, parameterization and freeform adaptation. Technically, all these adaptations are implemented as freeform adaptations in PREvent.

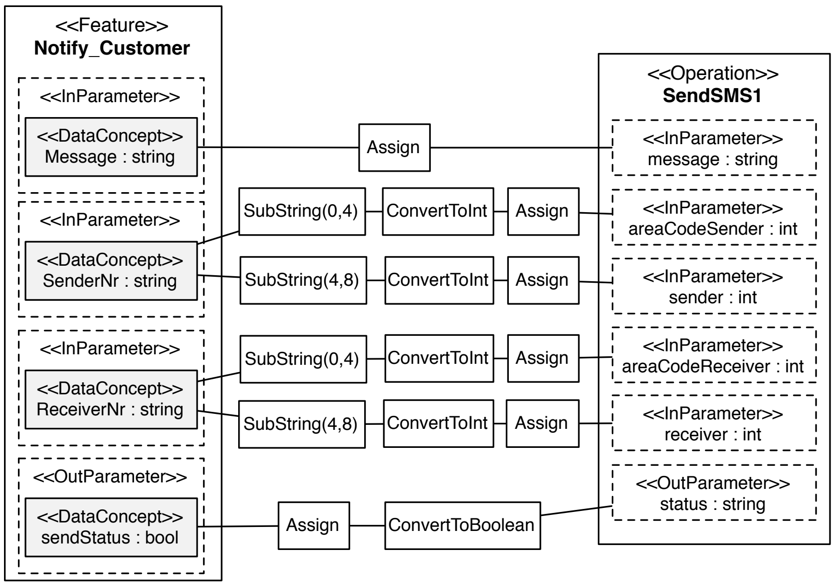


Figure 4.25: VMF Mapping Example

Hence, in the following, we will refer to all of these adaptations uniformly as freeform adaptation.

In PREvent, the notion of aspect-based fragment substitution is used to model how service composition instances can be adapted at runtime. This approach is general, in the sense that it is not specific to any concrete composition model. Instead, it can be applied to many existing block-structured models, including WS-BPEL and WWF. The approach reuses well-known Aspect-Oriented Programming (AOP) [116, 182] terminology to describe adaptations of service compositions. The most important of these terms are *aspects* (cross-cutting concerns, which are turned off and on at design or run-time, e.g., logging), *advices* (business logic which implements aspects, e.g., the code to implement logging), *joinpoints* (points in the application code where advices can potentially be inserted), and *weaving* (the process of dynamically inserting advices in jointpoints). In PREvent, the term *fragment* is often used instead of advice. Note that, in literature, AOP is often discussed as both a design time and a run time technology, i.e., weaving can happen both statically or dynamically. In PREvent, weaving is only considered at runtime, as the primary concern is the adaptation of composition instances, without modification of the underlying definition.

The main concepts of the fragment-based adaptation approach in PREvent are summarized in Figure 4.26. Adaptation actions are defined mostly by a composition fragment, which defines the business logic after adaptation. This fragment is linked to the original composition model (denoted as *target composition*) using two types of *joinpoints*. In-joinpoints mark the beginning of the composition segment to replace, while out-joinpoints mark the end of the segment.

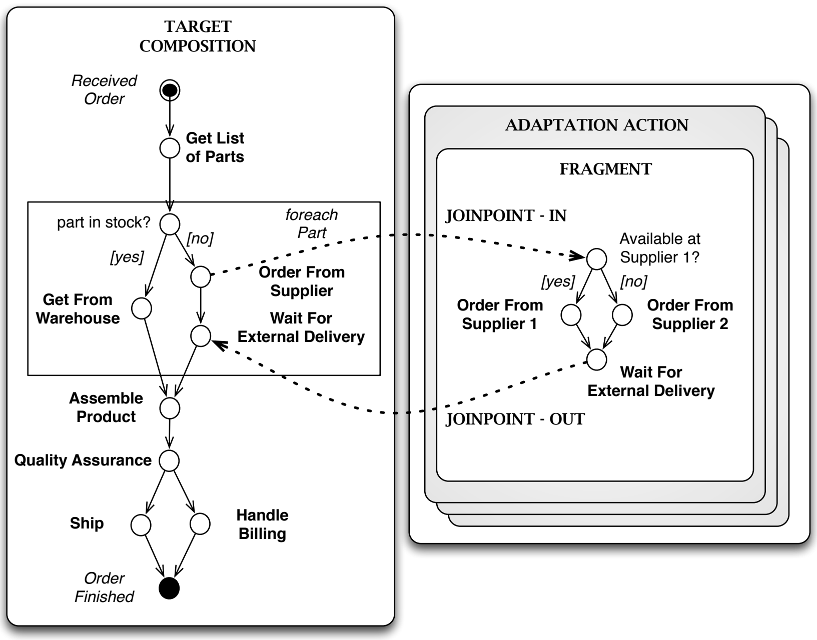


Figure 4.26: Aspect-Based Freeform Adaptation

Composition Fragments

Composition fragments can be considered the core of the freeform adaptation approach. In essence, fragments are full-fledged, even if usually small, service compositions. That is, fragments may contain variables, branches, Web service invocations, parallel executions, loops, scopes, fault handling, compensation, or any other construct which is legal in the used composition model. However, they do not have to follow the same syntactic and semantic rules as the target composition. For example, if WS-BPEL is used as composition model, designers of fragments may access e.g., variables defined in the target composition, even if the respective data is undefined in the fragment itself (syntactic rule). Also, they could specify a receive activity without a corresponding reply activity (semantic rule). The reason for this is that, during weaving, the fragment will be inserted into the composition model of the target composition, essentially becoming part of the composition itself. A fragment definition is valid if it results in an executable composition after weaving, which cannot be checked in isolation.

In addition to all activities provided by the composition metamodel, fragments may contain three additional activity types (`FRAGMENT_START`, in the following referred to as `start`, `FRAGMENT_END`, `end`, and `TRANSPARENT_BLOCK`, `transparent`) with semantics specific to the PREvent approach. These activities are referred to as *virtual activities*, because they are never actually executed. Instead, virtual activities are dropped or replaced during weaving. Virtual activities are solely responsible for demarcating the joinpoints between the fragment and the target composition, marking the segment of the target composition to substitute.

Every fragment starts with exactly one `start` activity and ends with exactly one `end` activity. In-joinpoints, defined via the `start` activity, represent the start of substitution, and out-joinpoints, defined via the `end` activity, represent the end of substitution. All joinpoints can reference any activity in the service composition, either before or after the execution of the activity (i.e., both "immediately before executing Get List of Parts" and "immediately after executing Get List of Parts" are valid joinpoints). However, the in- and out-joinpoints of a fragment need to reference activities in the same sequence in the target composition, i.e., the joinpoints defined in Figure 4.26 are correct, but, for example, it would not be possible to move the in-joint point to the activity "Get List of Parts". The reason for this limitation is that semantic problems arise if in- and out-joinpoints are situated in different sequences. In the previous example, the branching activity "part in stock?" would be removed, but not the actual branches, rendering it impossible to decide which branch to execute.

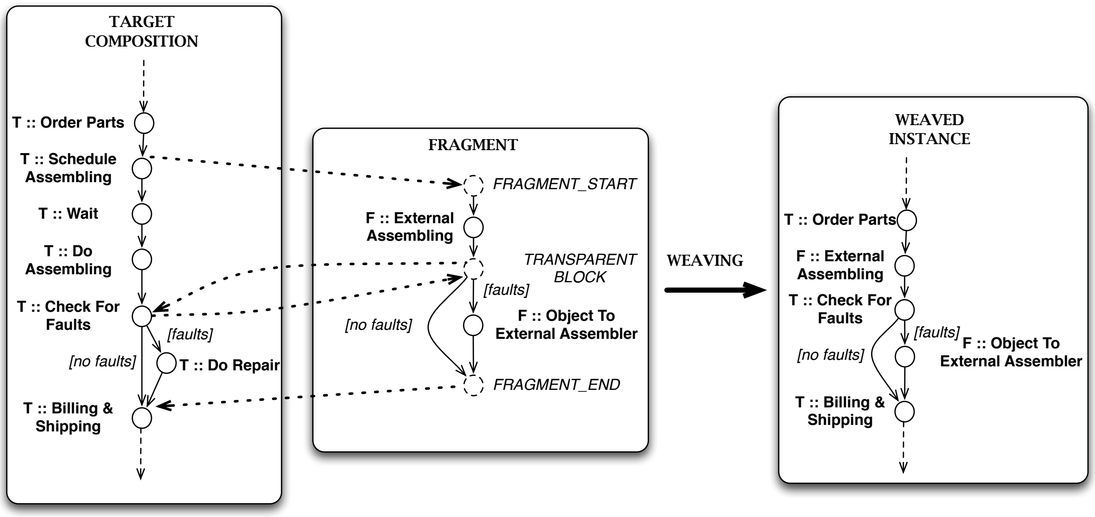


Figure 4.27: Fragment Activities and Linking

It is not only possible to replace a segment of the target composition, even though this is the general case. Trivially, one may also just insert the fragment at a specific joinpoint (the in- and out-joinpoints are identical, and the fragment is non-empty), or remove a segment (substitution with an empty fragment). The sum of all joinpoints of a fragment is referred to as the *linking* of the fragment to the target composition. Figure 4.27 summarizes this linking. The `start` activity specifies that the fragment should be inserted before the activity "Schedule Assembling", while the `end` specifies that the end of the substitution segment is before the activity "Billing & Shipping". On the right-hand side, Figure 4.27 shows the dynamically constructed instance after the fragment has been weaved into the target composition. Activities depicted with the prefix `T` originate from the target composition, while activities with the prefix `F` are specified in the fragment.

`Transparents` are more complicated than `start` or `end`. They are a placeholder, representing a part of the target composition in the fragment. This part is defined in the same way as the substitution segment, i.e., `transparents` have both out- and in-joinpoints. Additionally, the same restrictions apply (in and out-joinpoints need to reference activities in the same sequence). At runtime, `transparents` are replaced by a copy of the part that they represent. The purpose of `transparent` activities is threefold. Firstly, they allow for the definition of fragments substituting segments, while still retaining some of this segment's original functionality. One example of this usage is depicted in Figure 4.27, where the "Check for Faults" activity from the target composition is retained in the fragment. Note that it is not mandatory that a `transparent` references only a single activity. Secondly, transparent activities allow to essentially duplicate activities in the target composition. This is because `transparents` are in fact free to reference any part of the target composition, not only parts which are in the substitution segment (and hence removed during weaving). Additionally, many `transparents` may copy the same activities, multiplying them even further. Thirdly, `transparent` activities allow for the definition of generic fragments.

Generic Fragments

Generic fragments are (unlike the fragments discussed so far) not developed specifically for a given target composition. Instead, they can be applied to a number of compositions. Therefore, generic fragments do not contain any concrete case-specific business logics. They are used to implement adaptation scenarios which can be useful across several concrete target compositions and domains. Figure 4.28 exemplifies three generic fragments. The main property of generic fragments is that they consist only of virtual activities and control flow constructs, i.e., they do not contain any concrete activities such as Web service invocations. These generic fragments are instantiated by defining the linking (i.e., all in- and out-joinpoints) to concrete target compositions. As soon as this linking is defined, the fragment stops being generic, and is as case-specific as any other fragment.

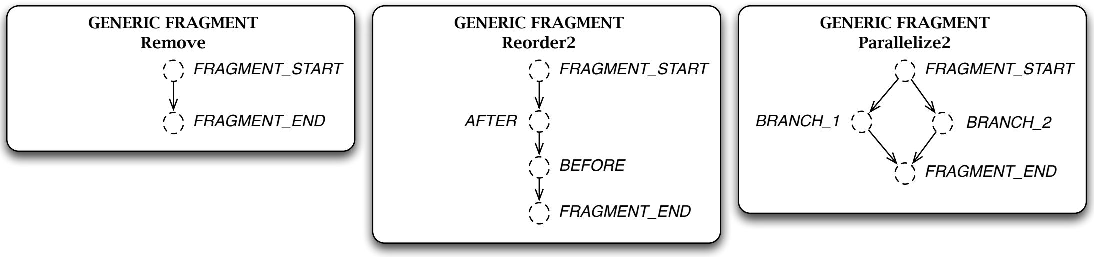


Figure 4.28: Examples of Generic Fragments

The first and most simple generic fragment in Figure 4.28 (`Remove`) has been mentioned before. It is an empty fragment consisting only of a `start` and `end` activity. Using this generic fragment, any segment of the target composition can be deleted. The second example is a generic fragment named `Reorder2`. It consists of `start`, `end`, and two `transparents` ("after" and "before"). Using this fragment two segments in the target composition can be rearranged, e.g., exchanging their order. Trivially, one can also implement similar generic fragments `ReorderX`, rearranging *X* segments instead of just two. Finally, `Parallelize2` consists again of `start`, `end`, and two `transparents` ("branch_1" and "branch_2"), however, this time “branch_1” and “branch_2” are executed in parallel. Using this generic fragment, one can parallelize two segments from the target composition (which presumably have been executed in serial before). Of course, it is again possible to define `ParallelizeX` fragments to parallelize more than two segments at the same time.

Dynamic Weaving

At run-time, one or more fragments are weaved into the running instance of the target composition. As sketched in Listing 4.14, the general weaving algorithm is a simple 2-step procedure. Firstly, the fragment is pre-processed, i.e., for each `transparent` in the fragment the linking to the target composition is resolved, and the `transparent` is replaced by a deep copy of the segment that it represents. Secondly, the linking of the fragment itself is resolved, and the `start` and `end` virtual activities are removed from the fragment (they are not needed anymore). Finally, the segment of the target composition (indicated by the linking) is removed, and the fragment is inserted instead.

```
 1 # name: weave
 2 # input: composition instance instance,
 3          fragment to weave fragment,
 4          weaving mode mode (OFFLINE or ONLINE)
 5 # output: weaved instance
 6
 7 weave(instance, fragment, mode):
 8   if(mode == "OFFLINE")
 9     suspend(instance)
10
11   # step 1 - fragment preprocessing
12   foreach transparent in fragment
13         linking = resolve_linking(transparent,instance)
14         copy = copy_segment(linking,instance)
15         replace_fragments(fragment,transparent,copy)
16
17   # step 2 - fragment substitution
18   seqment := resolve_linking(fragment,instance)
19   remove_start_activity(fragment)
20   remove_end_activity(fragment)
21   replace_fragments(instance,segment,fragment)
22
23   if(mode == "OFFLINE")
24     resume(instance)
25
26   return instance
```

Listing 4.14: Weaving Algorithm

Weaving can be done either online or offline. For offline weaving, the composition instance is halted while the adaptation is applied (see Lines 8-9 in Listing 4.14), and resumed when the adaptation is finished (Lines 23-24). If online adaptation is used the instance continues running during weaving. This has the advantage, that weaving does not introduce additional execution time overhead. However, if after weaving the running instance has already passed the entry point of the fragment (the linking of the fragment’s `start` activity) the weaving fails and is rolled back. This is because PREvent needs to guarantee that a fragment is either executed as a whole, or not at all (which cannot be guaranteed after the instance has begun executing the substitution segment in the target composition). PREvent falls back to offline adaptation as soon as at least one advice which needs to be applied requires it (i.e., if many advices are applied and only one of them requires offline weaving, the composition instance still needs to be suspended before adaptation). Generally, if more than one advice needs to be applied, recursive one-by-one weaving is used, that is, the framework starts by weaving the first fragment into the instance (ignoring any other fragments). The result of this first weaving process is then the input to the weaving of the second fragment. This is continued until all fragments are weaved.

CHAPTER 5

Evaluation

In this chapter, the contributions presented in Chapter 4 are evaluated. To this end, an implementation of the ACMEBOT case, as introduced in Chapter 3, is used. The concrete experimental setup is discussed in Section 5.1. Afterwards, the evaluation again follows the phases of the MAPE loop (Section 5.2 to Section 5.5). Finally, the chapter concludes with an end-to-end evaluation of the PREvent framework in Section 5.6.

5.1 Experimental Setup

In order to quantify the benefits of the PREvent framework, a subset of the case study presented in Chapter 3 is used. More concretely, the evaluation is based on an implementation of the actual order handling part, i.e., the subflow starting with the activity "Receive Order" until the end of the business process. The case study has been implemented using .NET Windows Communication Foundation[1] (WCF) technology and the VRESCo SOA runtime environment on a server running Windows 7 SP1. The server machine is equipped with two 2.99GHz Xeon X5450 processors and 32 GByte RAM. A MySQL 5[2] database is used as data backend, and all necessary components are deployed on the same Windows server machine, in order to reduce the impact of external influences, such as network latency. The service composition itself has been implemented using WWF. A graphical overview over the implemented composition is contained in Appendix B. All the orange boxes are Web service invocations carried out using the custom VRESCo activities mentioned in Section 4.5. These custom activities use VRESCo to dynamically find a suitable service implementation for an abstract feature, and invoke it. Including flow control activities, the technical implementation of this case encompasses more than 40 activities.

40 different metrics of this service composition are monitored by the Composition Monitor component. These include both QoS information, such as the response time of every used Web service, and PPMs, such as customer identifiers or which products have been ordered. In order

[1] http://msdn.microsoft.com/en-us/library/ms735967(VS.90).aspx
[2] http://mysql.com

#	SLO Name	μ	μ^*	σ	σ^*	t_1
1	Order Fulfillment Time	38002	37760	4654	5201	36000
2	Payment Time	6293	5383	48	754	6250
3	Shipping Time	1288	1150	145	271	1300
4	Product Quality	n/a	n/a	n/a	n/a	3
5	Cost Compliance	853	876	204	297	1400

Table 5.1: Case Study SLOs

to bootstrap the PREvent system, the Metrics Database has been initialized with a set of 10000 historical executions of the service composition, which have been generated by executing the composition with various randomly generated combinations of input data. Of those instances, 5000 instances have not been adapted (i.e., these instances are executions of the composition exactly as statically defined). The 5000 remaining instances have been adapted, i.e., at least one adaptation action has been applied to those instances. In this bootstrap phase, adaptations to apply have been selected at random. In the remainder of the evaluation, it is assumed that this service composition has an SLA containing five SLOs. In Table 5.1 these SLOs are sketched, and their basic statistics are given. μ is the mean value of the SLO without adaptation. μ^* is the mean among instances to which some adaptation has been applied. σ and σ^* are the respective standard deviations. t_1 is the violation threshold of the SLO. As SLO 4 is qualitative, mean and standard deviation are not defined. SLO 1 is associated with a staged penalty function with 9 stages, SLO 2 and 3 are both associated with fixed penalty functions, SLO 4 is associated with a linear penalty function with cap, and SLO 5 with a linear penalty function without cap. Additionally, 49 adaptation actions are defined, which have positive and negative influences on some or all of these SLOs. Every action has been associated with a positive cost value. Adaptation actions include data manipulation actions (invoking some services with higher priority), service rebinding without interface differences (using faster services) and structural adaptations (adding additional acivities, for instance additional quality assurance steps).

Unless stated otherwise, this setup is the basis for all experiments described below. Furthermore, unless stated otherwise, all numerical experiments have been repeated at least 100 times (with results being averaged), in order to minimize the inherent randomness of a real-life Web services testbed.

5.2 Evaluation of Monitoring

In the following, two concerns of the PREvent Composition Monitor are evaluated. Firstly, a brief qualitative discussion will argue over the expressiveness of the monitoring approach, i.e., what types of SLOs can be defined and monitored using the Composition Monitor. Secondly, the performance impact of monitoring depending on the number of monitored metrics is evaluated.

Qualitative Discussion

Event-based monitoring as used by the Composition Monitor component is a powerful tool. It enables non-intrusive monitoring of the composition, and is generally rather light-weight.

However, there are two limitations of the PREvent monitoring approach:

- The XML based definition language discussed in Section 4.2 (Listing 4.1 and Listing 4.2) can be easily used to define instance-level metrics (e.g., the response time of the Warehouse Service in this instance). However, typical aggregated metrics, such as the availability of the Warehouse Service, are hard to express. In general, using event-based monitoring only metrics concerning instance-level data can be monitored. In PREvent, this problem is mitigated using external data providers. That is, if metrics need to be monitored which cannot be expressed using events, it is still possible to include those metrics using custom external data providers.

- Using message property paths it is possible to extract arbitrary payload data from events and use them as part of the metric definition, e.g., a customer identifier. However, these property paths are very specific to the interface of the used Web services. Evidently, this clashes with the idea of loose coupling of activities and concrete service implementation, as discussed in Section 4.5. In order to solve this problem, it is vital to transform event payloads to some unified representation (i.e., lifting in semantic Web services speak [100]) before extracting message properties. However, this is out of scope for this thesis.

These limitations notwithstanding, the event-based monitoring approach is still a viable middle ground, as it is less intrusive than other patterns for QoS monitoring [141], and is still able to express all instance-level metrics that can actually be monitored from events emitted by the service composition.

Performance Evaluation

In order to get an impression of the overhead that is caused by event-based monitoring, the execution time of the service composition has been monitored, firstly with monitoring disabled, and secondly with an increasing number metrics being monitored. In Figure 5.1 the results of these experiment are depicted. The y-axis in this figure depicts the additional execution time induced by monitoring. By tendency, the monitoring overhead increases with the number of monitored metrics, but does not do so dramatically.

Overall, the overhead for monitoring a single metric (which also includes enabling eventing in VRESCo, which has a certain overhead in itself) is in the area of 350 to 400 ms. Additional metrics slightly increase the overhead.

5.3 Evaluation of Prediction

In this section, the PREvent prediction approach is evaluated. To this end, a checkpoint is introduced after activity “order_parts” (see Appendix B). Firstly, it is discussed, how time-consuming it is to build the machine learning based prediction models in this checkpoint, and to carry out actual predictions. Secondly, and more importantly, the accuracy of predictions in the checkpoint is evaluated. Please note that both, the time necessary for training and the accuracy of

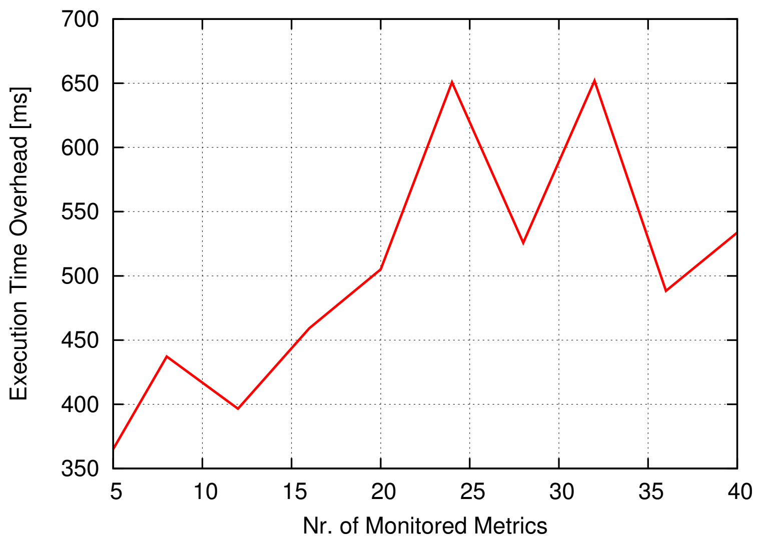


Figure 5.1: Monitoring Overhead Based on Monitored Metrics

the predictions, would be different if a different checkpoint was chosen. This specific checkpoint was used, as it is a reasonable tradeoff between accuracy of prediction and possibilities for adapptation.

Performance Overhead of Prediction

In order to evaluate the performance of runtime prediction, the time necessary for training the machine learning models used in the SLO Predictor have been measured. In Table 5.2, the measured times for two essential operations of the SLO Predictor component are presented. Table 5.2a depicts the amount of time in milliseconds necessary to initially build or retrain a prediction model in a checkpoint. These numbers refer to the training of the ANN used for SLO 1, as it is the SLO with the most influential factors (hence, training of the model takes the longest for this SLO). The time necessary for training mainly depends on the available number of historical composition instances and the number of metrics that are used as input. The values in Table 5.2a indicate, that the time necessary for building the model depends linearly on the available number of historical instances. For 5000 instances, the absolute model training time is approximately 8.1 minutes, which seems acceptable for practice, considering that model rebuilding can be done sporadically and offline. Additionally, after the initial construction of the first prediction model, there is no time when no model is available at all. Instead, whenever retraining is triggered as described in Section 4.3, the new model is trained offline, and exchanged for the old model as soon as training is finished.

As a second interesting performance aspect, the time necessary to carry out predictions has been measured. Table 5.2b sketches the measurements for SLO 1. Note that this overhead is slightly more significant for the runtime performance than the training time, as it refers to the online part of prediction. Fortunately, the results depicted show that the absolute time necessary

Instances	Training [ms]
100	10082
250	24343
500	48573
1000	96949
5000	486164

(a) Training Overhead

Instances	Prediction [ms]
100	81
250	83
500	87
1000	98
5000	179

(b) Prediction Overhead

Table 5.2: Overhead for Training and Prediction Using ANNs

for prediction (between 80 and 180 ms) is small. These results refer to metric SLOs which are predicted using ANNs. For nominal SLOs decision trees are used. Decision trees are much faster to train, and prediction using decision trees takes even less time than using ANNs.

Quality of Prediction

Even more important than the time necessary to generate prediction models or predictions is the quality of these predictions. To visualize the prediction quality of PREvent in the ACMEBOT case, Figure 5.2 plots for 98 distinct instances the predicted value for SLO 1 (blue x) and the respective measured value (green +). Additionally, the SLO violation threshold t_1 is plotted at 36000 ms.

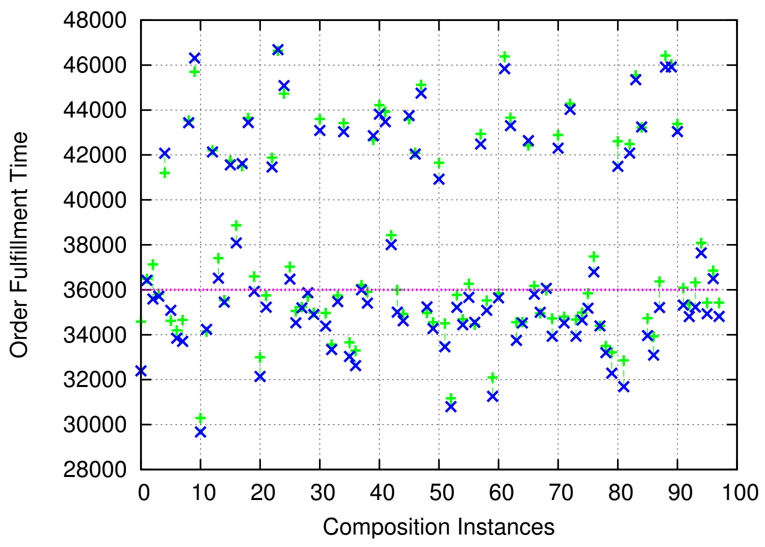


Figure 5.2: Comparison of Predicted and Measured Values for SLO 1

As can be seen, predictions and measured values are generally close, i.e., the prediction quality is good. However, for some isolated instances, the prediction is off by a larger margin, even though the number of wrong predictions (both false positives and false negatives) is small (only 8 of 98 predictions were wrong).

Table 5.3 substantiates this. For each SLO, the three basic metrics for prediction, as defined in Section 4.3 (training data correlation $corr$, prediction error $\bar{e}$, and prediction error variance $\sigma_{\bar{e}}$), are given. Additionally, the number of correct and incorrect predictions are given. SLO 4 is nominal, hence a decision tree is used. For this SLO, only the number of correctly and incorrectly predicted instances is given, as the other metrics cannot be reasonably calculated for nominal SLOs.

SLO	$corr$	$\bar{e}$	$\sigma_{\bar{e}}$	**Correct/Incorrect**
1	0.98	480	360	90/8
2	0.72	135	75	98/0
3	0.99	126	79	85/13
4	n/a	n/a	n/a	98/0
5	0.56	120	93	97/1

Table 5.3: Prediction Quality of SLO Predictors

Evidently, predictions are generally quite good. The weakest predictor has been trained for SLO 3, even though the $corr$ metric would suggest otherwise. This is an indicator that the estimators used in the prediction are not accurate enough. Finally, it can be seen that for SLOs 2 and 5, $corr$ is very low, but the actual predictions are doing fine. This can happen, if the measured metrics are usually either far below or far above t_1, i.e., a larger prediction error is unproblematic because the verdict for most instances is obvious. This is the case for SLOs 2 and 5, but not for SLO 1 (Figure 5.2 visualizes well that most instances are clustered close to the violation theshold).

5.4 Evaluation of Optimization Algorithms

In this section, the optimization algorithms that are used to identify the most suitable actions to prevent predicted violations, are evaluated. All variants of deterministic and heuristic optimization (as presented in Section 4.4) are compared with regard to (1) how quickly the algorithm delivers the final result, measured in how many solutions are evaluated, and (2) the quality of this solution, measured as the TC (defined as in Section 4.4). Concretely, different variants of Branch-and-Bound (without specific sorting of actions, with impact-based sorting and with utility-based sorting) and various representatives of heuristic optimization (Local Optimization, GRASP, GA and MA) are compared.

Deterministic Algorithms

As a first experiment, the suitability of different variants of the Branch-and-Bound algorithm are analyzed.

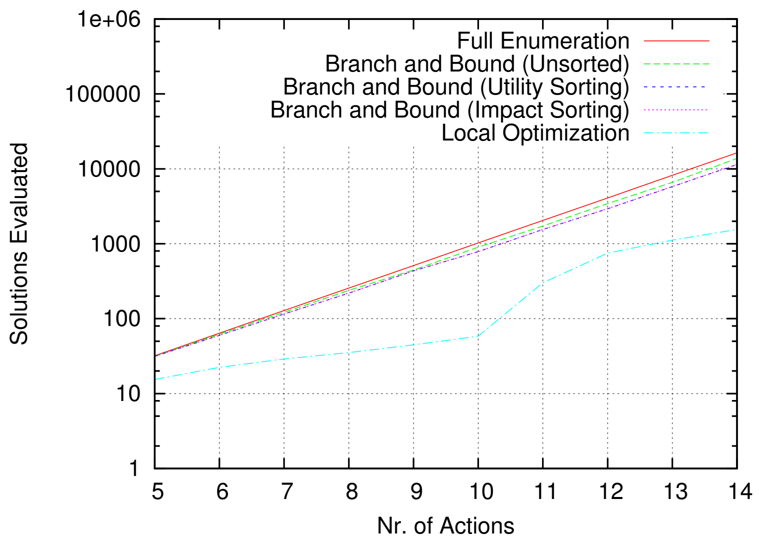


Figure 5.3: Solutions Evaluated For Branch-and-Bound, Average of 5 Random Instances

As all of these algorithms are deterministic, it is guaranteed that the optimal solution is found to any optimization problem eventually. However, the three different versions of the algorithm (Branch-and-Bound with random action sorting, with impact-based sorting, and with utility-based sorting) may differ with regard to their runtime.

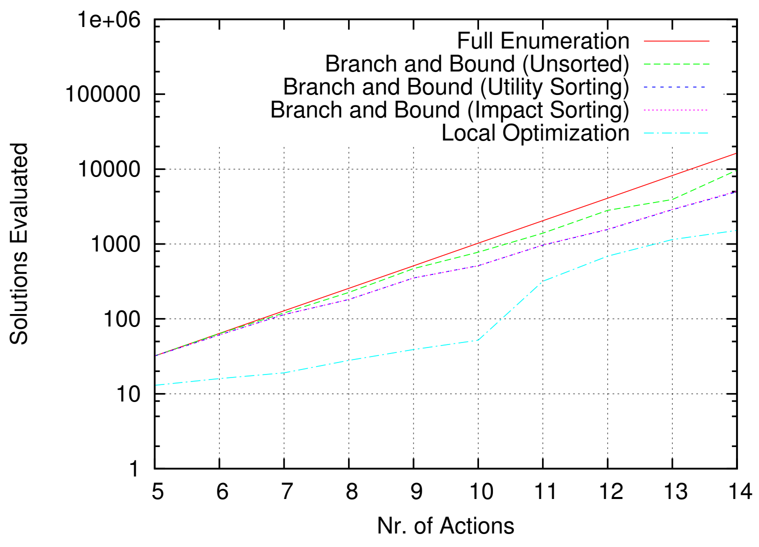


Figure 5.4: Solutions Evaluated For Branch-and-Bound, Without Conflicting SLOs

Figure 5.3 plots the number of solutions that have to be evaluated before the algorithm is finished, depending on the number of adaptation actions that are available (up to a maximum of 14 actions, note the logarithmic scale on the y-axis). The plotted results are the average of the optimization of 5 random instances of the ACMEBOT case. For reasons of comparison, the Local Optimization algorithm is also plotted in this figure. It was not feasible to evaluate Branch-and-Bound for more than 14 actions. For these algorithms, it is not necessary to discuss solution quality, as all algorithms (except Local Optimization) deliver the same optimal solution by definition.

As can be seen, Branch-and-Bound performs poorly in this experiment, being unable to reduce the number of evaluated solutions significantly below full enumeration. This can be explained easily, as the set of adaptation actions used for this experiment contains no actions that cannot be used in conjunction (i.e., pruning condition (1) never goes into effect), and furthermore, the set of SLOs used in the ACMEBOT case are conflicting (oftentimes, preventing violations of SLOs 1 to 4 leads to violations of SLO 5). This means that, unfortunately, pruning condition (2) is neither particularly effective in the ACMEBOT case. For instances of the composition where it is possible to find solutions that do not violate any SLO, a more significant performance boost can be achieved. Such an instance is depicted in Figure 5.4. For this instance, some improvement over full enumeration can be achieved using Branch-and-Bound. Both impact-based and utility-based sorting are performing better than Branch-and-Bound with random action sorting. The difference between impact-based and utility-based sorting is not significant in this experiment. However, note that the optimization effort still increases exponentially, and that the number of solutions that can be pruned is still not particularly large. Hence, in the ACMEBOT case, Branch-and-Bound is viable only for very small optimization problems.

Evaluation of Heuristic Algorithms

For choosing the best among a larger number of potential adaptation actions, heuristic otimization can be used. For these algorithms, there are no guarantees about the quality of the solution. This means, that it is necessary to compare them along two different dimensions. Firstly, and similarly to before, one needs to look at the number of solutions that are evaluated before the algorithm produces the final result (Figure 5.5), as a measure of the execution time of the algorithm. Secondly, it is now also necessary to take into account the quality of the best found solution (Figure 5.6). These results are the average of the same 5 instances as used for Figure 5.3.

In Figure 5.5, one can see that, not surprisingly, all algorithms scale much better than Branch-and-Bound (note the linear scale on the y-axis, and compare with Figure 5.3 and Figure 5.4). GRASP and Local Optimization are both very efficient, exhibiting an almost constant runtime. Note that the number of solutions evaluated for Local Optimization and GRASP is directly proportional to the number of start solutions used. In this experiment 25 start solutions have been used for both algorithms. GA also exhibits a relatively constant runtime, but on much higher level than GRASP and Local Optimization. The slowest algorithm in this experiment is MA, which is due to its unique combination of Local Optimization and GA.

With regards to solution quality, as presented in Figure 5.6, MA is clearly the best algorithm in the experiment. Local Optimization, GA and GRASP perform more or less comparably. Note that the unexpectedly bad performance of GRASP is in contrast to the positive results that have been reported for other settings [105]. The results for all other algorithms are similar to those in

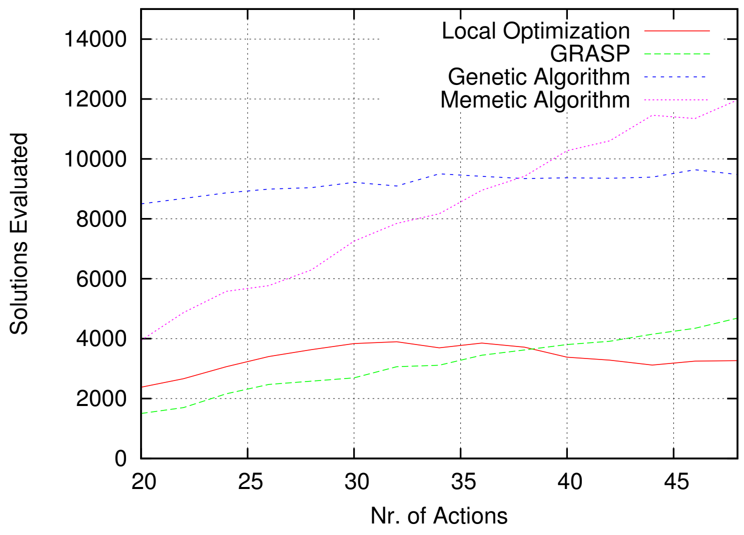


Figure 5.5: Solutions Evaluated Per Heuristic Algorithm

earlier experiments. Hence, the performance of GRASP seems rather case-specific, and should be critically analyzed for each use case individually.

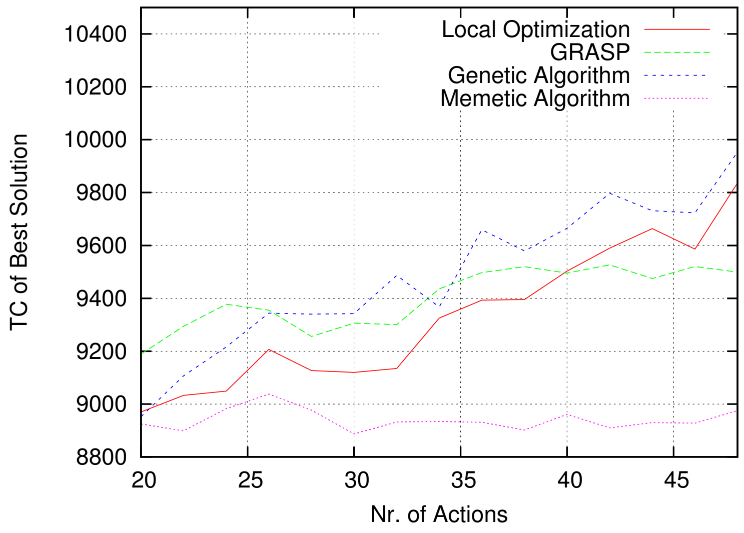


Figure 5.6: Quality of Solution Per Heuristic Algorithm

In order to conclude this comparison of algorithms, it is evident that the deterministic Branch-and-Bound algorithm is applicable only in situations where just a very small set of adaptation actions is available. In general, impact-based or utility-based sorting should be used instead of random sorting, since there is no evident disadvantage to these approaches, and they may be helpful in at least some cases. The experiments discussed here could not reveal a significant difference between those two ordering strategies. If a larger set of actions is available, MA and GRASP are interesting candidate algorithms. GRASP produces good solutions for some cases, but not for all. MA is very promising in case of long-running compositions, where the time necessary to find a solution is not critical. MA usually produces the best solutions in these experiments, but generally takes a long time to do so.

5.5 Evaluation of Adaptation Execution

As final element of the MAPE loop, this section evaluates the runtime adaptation mechanisms provided by PREvent. Firstly, the performance overhead of dynamically invoking Web services using the DAIOS framework is evaluated. This refers to service rebinding without interface mediation. Afterwards, focus is put on adaptation by service rebinding with interface mediation. Similarly to the evaluation of monitoring, this section presents both a qualitative discussion of the expressiveness of the VMF mediation library and a performance evaluation of the framework. Finally, fragment-based structural adaptation is evaluated with regard to covered workflow change patterns [185], performance of runtime weaving and performance of weaved composition instances.

Dynamic Web Service Invocation

As explained in Section 4.5, the DAIOS framework is used for all dynamic Web service invocations in PREvent.

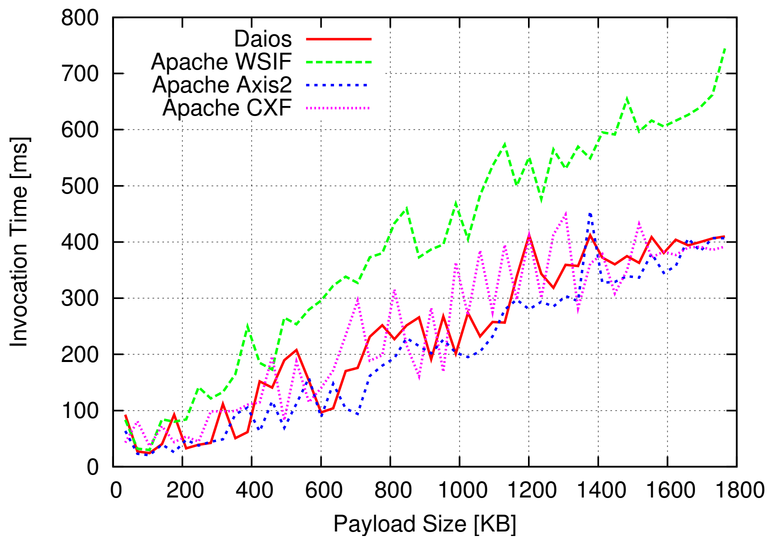


Figure 5.7: Performance Comparison of Web Service Frameworks – String Payload

Hence, the performance of this framework is essential for all adaptation actions. The following experiments compare DAIOS with various existing Apache Web service frameworks

(Apache Axis 2[3], Apache CXF[4] and Apache WSIF[5]). Comparing DAIOS with Apache WSIF is particularly interesting, as WSIF follows similar goals with regard to ease of dynamic invocations.

In the first experiment, the frameworks have been used to invoke a simple Web service operation, which accepts a single string parameter and simply returns the passed string back. This test service has been generated using the Genesis service testbed generator [89, 90]. For all frameworks, the roundtrip time (time between starting the invokation on client-side and receiving the result) has been measured for increasing string sizes. The results of these experiments are depicted in Figure 5.7. Axis 2, CXF and DAIOS perform comparably. WSIF is significantly slower than the other frameworks.

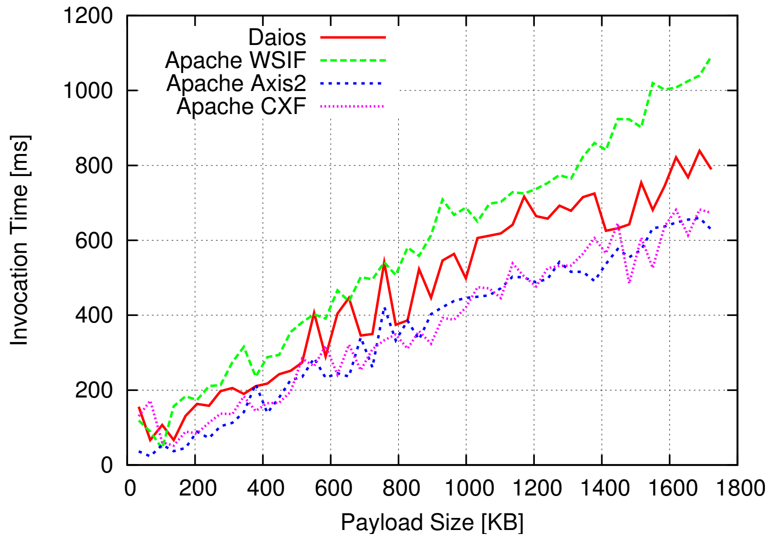


Figure 5.8: Performance Comparison of Web Service Frameworks – Binary Payload

In a second step, this experiment has been repeated with a different service, which accepts a single parameter of binary data, and again returns the binary data unchanged. Figure 5.8 reports on these results. In this experiment, DAIOS is slightly slower than CXF and Axis 2, presumably since DAIOS does not yet implement advanced mechanisms for handling binary data, such as the SOAP Message Transmission Optimization Mechanism (MTOM) [180]. However, DAIOS still performs considerably better than WSIF, the other dynamic Web service invocation framework in the test. Generally speaking, these results show that the runtime performance of the DAIOS framework is comparable to other open source frameworks, hence, building adaptive service compositions on top of DAIOS is not expected to have a significant negative performance impact.

[3]http://axis.apache.org/axis2/java/core/

[4]http://cxf.apache.org/

[5]http://ws.apache.org/wsif/

Rebinding With Interface Mediation

In the following, the VMF framework, which implements the mediation necessary for rebinding services with different interfaces, is evaluated. Firstly, the expressiveness of the VMF mapping language is discussed. Afterwards, the overhead introduced by mediation is quantified in experiments.

Qualitative Discussion

Earlier work has argued that, even though Web services are a largely standardized technology, interoperability issues still exist in practice [135]. Three patterns of mismatch for Web services are discussed in [99]. Essentially, the mismatch patterns presented in this work are divided into service interface mismatches (the Signature Mismatch Pattern) and mismatches on service protocol level (the Missing Message and One-to-Many Mismatch Patterns). The VMF framework can be used to mediate changes that fall into the Signature Mismatch Pattern. All changes that require adaptation of the MEP cannot be covered using VMF. However, changes of this type are covered in this thesis through fragment-based structural adaptation.

Essentially, mediation of differences following the Signature Mismatch Pattern boils down to mediation of XML documents. In [36], the following simple types of mismatch between two XML documents are identified: (1) deletion of a subtree, (2) insertion of a subtree, (3) update of a text node or attribute value, and (4) moving a node or subtree. In principle, VMF allows to resolve any of those mismatches. However, one hard practical problem in mediation is information that is required in one but simply not present in the other XML document. In VMF, this can to some extend be remedied using either `Constant` mapping functions (e.g., if one service does not require an API key but the other does, the key can be inserted using a `Constant` mapping function) or the powerful `CS-Script` mapping function. Using CS-Script it is possible to, e.g., query external data sources at mediation time and retrieve the necessary information there. However, evidently doing so will have a significantly negative impact on the total service invocation time.

Performance Analysis

In order to measure the performance impact of mediation using VMF, the experiment setup used in Section 5.5 has been extended. In a first experiment, a number of mapping functions are applied to an invocation before issuing the request. Note that, for evaluation purposes, all mappings are defined in such a way that the end result of executing the mapping is always the same as the start message.

Figure 5.9 plots the total invocation time (including mediation overhead) depending on the message size for various scenarios (no mediation, applying constant mapping functions, applying mathematical functions, applying string operations, and executing a CS-Script). Not surprisingly, unmediated invocations are generally faster than any type of mediation, however, for smaller messages the mediation overhead seems acceptable (for a message of about 1000 KB, the mediation overhead is roughly 20 ms, or 6% of the total invocation time). One particularly interesting result of this experiment is the degree by which the overhead of mediation becomes

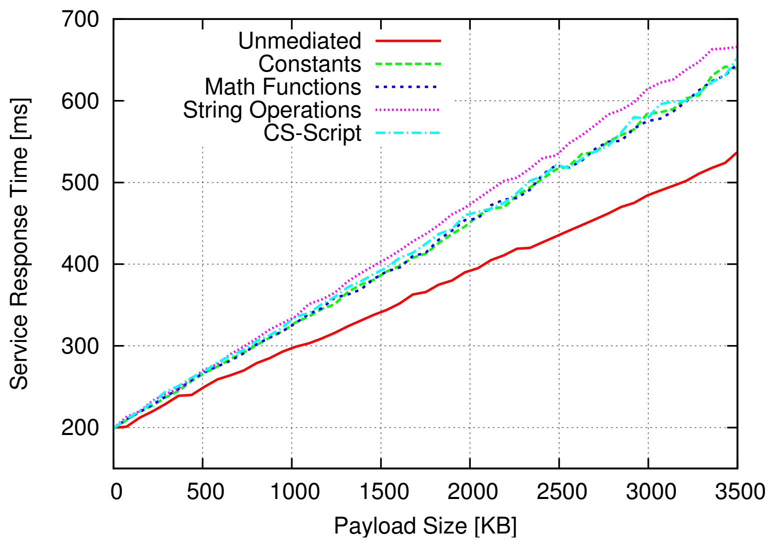


Figure 5.9: Mediation Performance (Message Size)

larger if the actual message becomes larger. This is because it is computationally more expensive to manipulate larger messages. Evidently, this effect is most important for mediation using string operations (e.g., splitting strings, concatenating strings).

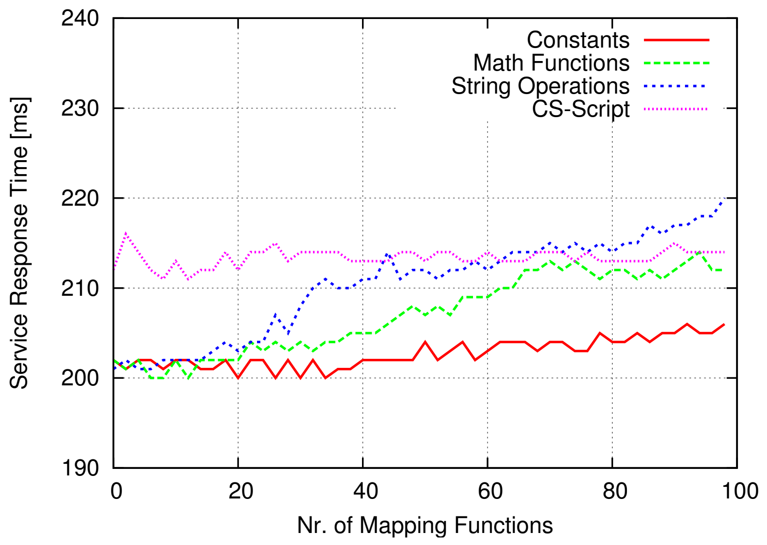


Figure 5.10: Mediation Performance (Mediation Steps)

In Figure 5.10, the overhead introduced by different mapping functions is studied in more detail. It is evaluated how the overhead introduced by mediation depends on the amount of mediation necessary (measured in the number of mapping functions applied). Generally, the additional overhead introduced by a larger number of mapping functions is rather small. The difference between 1 and 100 mapping functions varies between 5 and 20 ms, which seems acceptable. As before, the overhead introduced by string operations heavily depends on the size of the strings to modify. The string used in these experiments had a size of about 73 KB. Note that the total overhead of CS-Script mappings is constantly around 10 ms, as its main constituent is the initialization of the scripting engine, while the execution of the actual script is negligible (under the assumption that the script does not do any expensive operations itself, such as querying external data sources, as discussed above).

Fragment-Based Structural Adaptation

Finally, the AOP based structural adaptation is evaluated. As before, the structural adaptation approach is analyzed qualitatively, by comparing the expressiveness of the PREvent approach with previously published adaptation patterns [185], and quantitatively, by comparing the execution time of dynamically adapted and statically defined composition instances. Additionally, the time necessary for dynamic weaving is analyzed.

Coverage of Adaptation Patterns

In order to discuss the expressiveness of our approach, the adaptation patterns defined in [185] are used. In this work, 14 patterns of structural changes in processes are identified. Using the PREvent approach, 9 of these patterns are fully supported.

ID	Pattern Name	Covered
AP1	Insert Fragment	✓
AP2	Delete Fragment	✓
AP3	Move Fragment	✓
AP4	Replace Fragment	✓
AP5	Swap Fragment	✓
AP6	Extract Sub Process	✗
AP7	Inline Sub Process	✗
AP8	Embed Fragment in Loop	✓
AP9	Parallelize Activities	✓
AP10	Embed Fragment in Conditional	✓
AP11	Add Control Dependency	✗
AP12	Remove Control Dependency	✗
AP13	Update Condition	✗
AP14	Copy Fragment	✓

Table 5.4: Coverage of Adaptation Patterns

The coverage of adaptation patterns is summarized in Table 5.4. The patterns **AP1**, **AP2** and **AP4** are the core feature of structural adaptation, and can be implemented trivially. **AP2** and **AP5** are implemented using the generic fragments described in Section 4.5. Similarly, all of **AP3**, **AP5**, **AP8**, **AP9**, **AP10** **AP14** can be implemented rather elegantly using `transparents`. The patterns concerning subprocesses (**AP6** and **AP7**) cannot be implemented, as PREvent does not support linking to more than one composition at the same time. **AP11** and **AP12** are in simple cases implementable using `transparents`, but PREvent does not provide any explicit support for it, making the implementation rather cumbersome. Similarly, **AP13** can be implemented by replacing the branching node as a whole, but this solution is not in line with the idea of this pattern. Hence, **AP13** has been stated as “not covered” in Table 5.4.

Performance Analysis

In a second step, the runtime implications of structural adaptation are analyzed. For this, the average execution time of dynamically weaved composition instances with an increasing number of activities is monitored, and compared to the execution time of statically defined (identical) instances. Furthermore, online and offline weaving, as well as fragments defined with and without `transparent` activities, are compared.

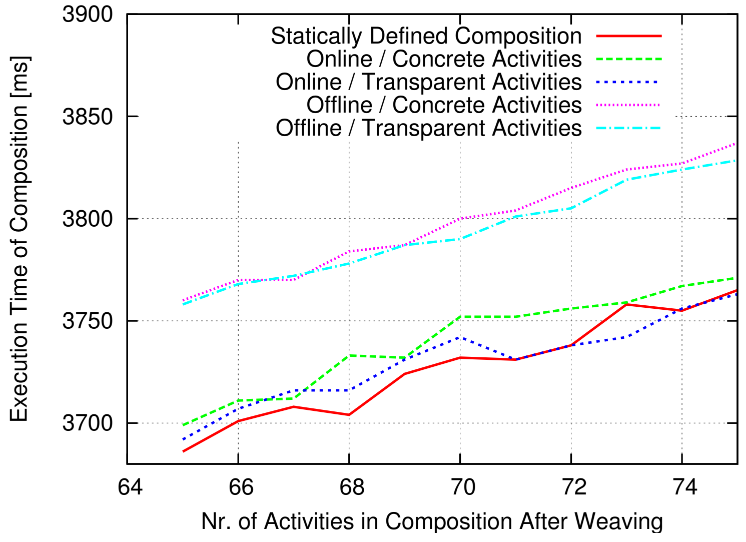


Figure 5.11: Execution Time of Weaved Composition

For simplicity, all compositions and fragments are sequences of “wait” activities in this experiment. Using different types of activities would not have an impact on the evaluation outcome, as the structural adaptation approach handles all non-virtual activities the same way, i.e., weaving an “invoke” activity has a the same overhead as weaving a “wait” activity. The outcomes of these experiments are summarized in Figure 5.11. Online weaved compositions exhibit very little overhead as compared to statically defined compositions. Of course, offline weaving introduces some overhead, which stems from the time necessary to select the fragments, to implement the actual weaving, and to suspend and resume the composition. The largest part of these factors is the actual weaving time. Therefore, this factor is analyzed further in Figure 5.12. There, the weaving time is depicted depending on the number of activities to weave.

Generally, concrete activities are faster to weave than `transparent` activities (this is because the logics of weaving `transparents` is more complicated), and offline weaving is faster than online weaving (as, in the online case, some additional sanity checks are done by the

WWF runtime). However, in general, this increased weaving time for online weaving does not matter too much, as the online weaving time does not directly impact the execution time of the process. Overall, the overhead introduced by weaving is typically between 45 ms and 80 ms, even for large fragments (more than 80 activities).

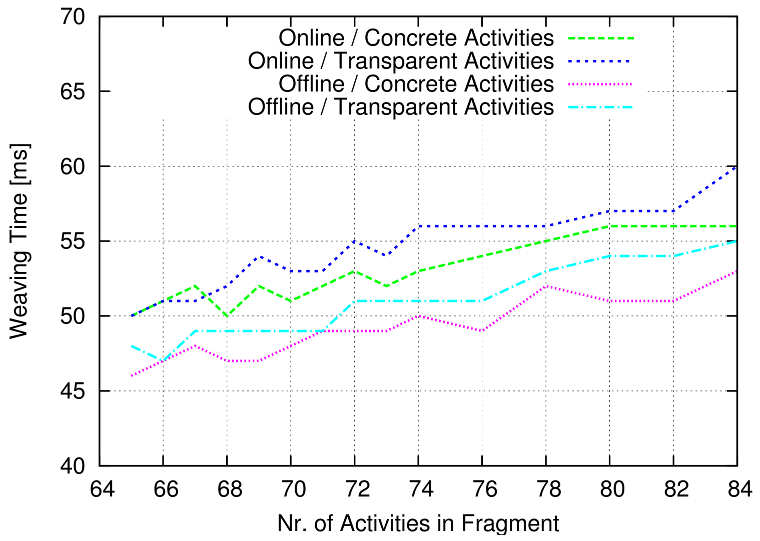


Figure 5.12: Execution Time Overhead of Fragment Weaving

Summarizing these experiments, structural adaptation using dynamic weaving does not introduce a big performance overhead, especially if online weaving is possible. If offline weaving has to be used, an additional weaving overhead, which is generally between 45 ms and 80 ms, is introduced. However, it can be argued that for most application areas this overhead is still far from dramatic.

5.6 End-to-End Evaluation

After evaluating all components of the PREvent framework individually, everything is now put together in an end-to-end evaluation. In this section, the main claim of PREvent (that applying PREvent indeed reduces the costs for service providers) is substantiated. Furthermore, the positive impact of stepwise optimization is shown.

Evaluation of Cost Reduction Using PREvent

In this section, the end-to-end effectiveness of PREvent is evaluated for the ACMEBOT case study. That is, it is analyzed if the system fullfills its main promise, preventing SLA violations and reducing the total costs (TC) for the service provider. Hence, 100 instances of the scenario composition are executed and the TC and the number of violations (after adaptation) are measured. These numbers are compared with the number of violations and the TC that the SLO Predictor predicted beforehand. This prediction forms the baseline of the comparison, i.e., it is assumed that these predictions reflect the violations and costs that the provider would end up with if he did nothing at all. In this experiment, no stepwise optimization has been used (cp. Section 4.4). Hence, in order to give the optimiziation some time to produce results, an artificial delay of five seconds has been added after the checkpoint. This will be relaxed in Section 5.6, when stepwise optimization will be evaluated. However, to be able to produce results within five seconds, a very fast optimization algorithm is needed. Hence, Local Optimization has been used for these experiments. The results of this experiments are depicted in Table 5.5.

	SLO 1	**SLO 2**	**SLO 3**	**SLO 4**	**SLO 5**	**Total**
Considering Costs						
Violations Predicted/Actual	244/205	0/134	339/48	76/69	0/37	659/493 (-25%)
Avg. TC Predicted/Actual	8751/6956	0/268	6102/864	928/876	0/35	15781/11154 (-29%)
Ignoring Costs						
Violations Predicted/Actual	255/160	0/1	381/48	72/287	0/87	708/583 (-18%)
Avg. TC Predicted/Actual	9043/3956	0/2	6872/379	974/3078	0/32	16889/21133 (+25%)

Table 5.5: End-to-End Results of Applying PREvent

Evidently, the usage of PREvent fulfills its main promise in the ACMEBOT case. Using PREvent, the total number of SLO violations decreases about 25%. However, one can also see that PREvent does not primarily prevent violations, but rather aims at minimizing the costs of violations. For instance, for SLO 2 and 4 the total number of violations even increases. This is because these SLOs are conflicting with the first SLOs, and violations of those are more expensive for the provider. Hence, PREvent trades violations of SLO 4 and 5 for preventing violations of the other SLOs. Thereby, the TC for the service provider can be reduced by 29%. Note that the Avg. TC are more than the sum of all SLO costs in the table, as the TC also include the costs of adaptation in addition to the sum of all penalty payments.

The lower part of the table validates the claim that it makes sense to incorporate the costs of adaptation into the decision process. To that end, the target function of the optimization has been modified in such a way that the costs of adaption are ignored. In this configuration, the TC after adaptation are 125% of the predicted TC. That means that, in this experiment, it is in fact much more expensive for the provider to prevent adaptations (in the way that optimization ignoring costs suggests) than doing nothing at all.

Evaluation of Stepwise Optimization

Now, in a last batch of experiments, the potential benefits of stepwise optimization are evaluated. To this end, the artificial delay mentioned in Section 5.6 is removed from the service composition, and MA is used as an optimization algorithm instead of Local Optimization. In this setting, stale results as discussed in Section 4.4 will surely become a problem.

In the following, the TC are compared for the following scenarios: (1) no adaptation at all (basically the ACMEBOT composition is just executed without any usage of PREvent), (2) suspending the composition while optimizing, (3) optimizing asynchronously and ignoring stale results (for simplicity, we refer to this as “regular optimization” in the following, as it is the approach that has been used in Section 5.6), and (4) asynchronous and stepwise optimization. In

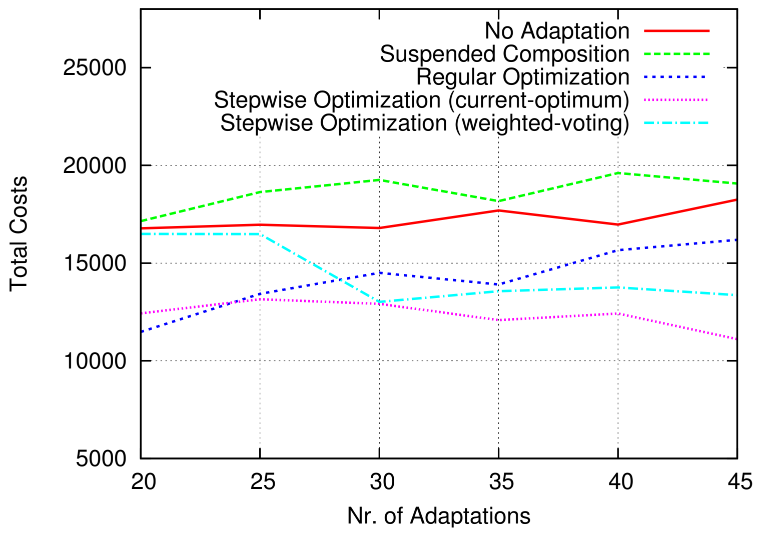


Figure 5.13: Average Total Costs for Different Optimization Approaches

Figure 5.13, the TC are plotted for each approach, depending on the number of adaptations that are available. Evidently, deciding which adaptations to apply becomes more difficult the more adaptations are available.

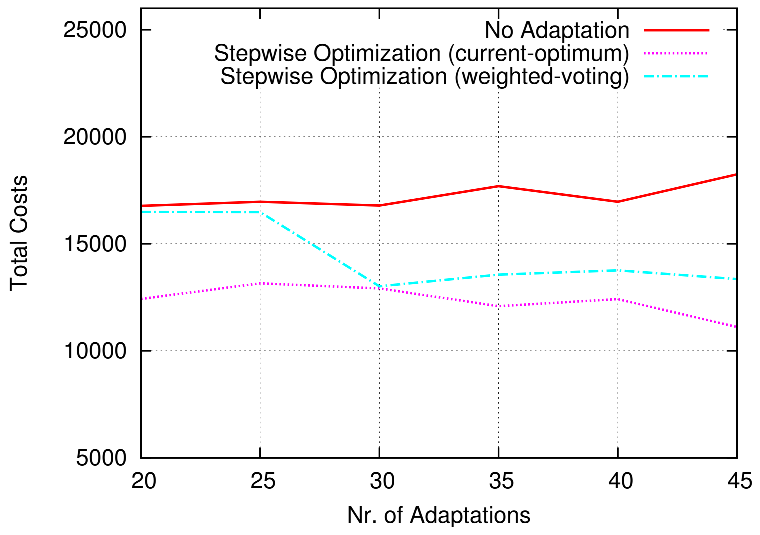


Figure 5.14: Average Total Costs for Different Optimization Approaches

As can be seen in Figure 5.13, using stepwise optimization improves the TC for the composite service provider as soon as the optimization problem becomes hard enough, that it cannot always be solved on time anymore (in this case, this happens at about 25 available adaptations).

It is also interesting to see that stepwise optimization is the only approach where the TC are actually (slightly) decreasing with additional adaptations becoming available (as one would intuitively expect). For regular optimization, the additional time necessary to decide what to do apparently offsets the potential benefits of additional choices. Suspending the service composition is evidently not a viable approach in any case, as, in this scenario, the additional overhead of suspension cannot be compensated by the potential benefits of adaptation.

Figure 5.14 takes a closer look at stepwise optimization. In this figure, the two decision strategies presented in Section 4.4 (weighted-voting and current-optimum) are compared. In these experiments, the simpler current-optimum based decision is clearly the way to go, bearing better results than weighted-voting for all setups.

Figure 5.15 plots the best solution found (in the case of suspended optimization) or the stepwisely constructed solution, again depending on the number of adaptations that are available. As is to be expected, the best solution found by regularly letting the algorithm finish is generally better than what can be constructed using stepwise optimization. However, as the end-to-end experiments in Figure 5.13 and Figure 5.14 show, this is usually offset by the overhead of suspending the service composition. Additionally, one can see in this figure that weighted-voting based decision performs considerably worse than current-optimum based decision, further explaining the poorer end-to-end performance of this approach as depicted in Figure 5.14.

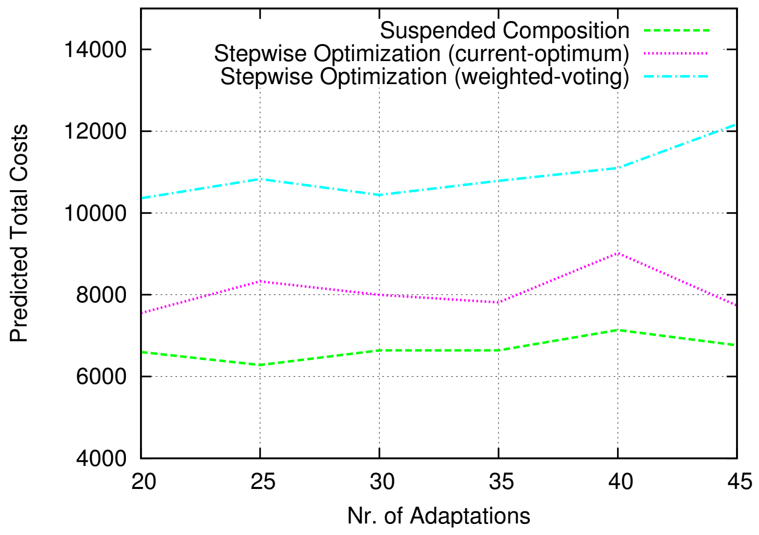


Figure 5.15: Average Best Solution

Another interesting aspect in stepwise optimization is how much overhead is incurred by the brief suspensions necessary for deciding on adaptations and changing the optimization problem. This overhead is depicted in Figure 5.16, where the cumulative execution time overhead of all decisions is plotted against the number of available adaptations. Evidently, the total overhead in milliseconds increases slightly with the number of actions, but is still very low for all experiments (between 55 and 65 milliseconds in total). Weighted-voting based decision takes a little longer than current-optimum based decision, which is due to the fact that in the latter strategy only one solution needs to be analysed, while the former is an aggregation of a whole population of solutions. Generally, this experiment shows well that stepwise optimization produces little overhead while still preventing stale results.

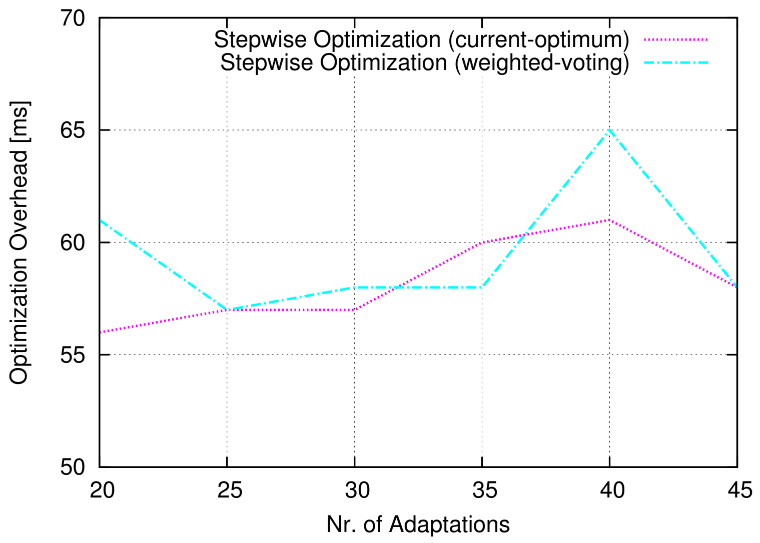


Figure 5.16: Decision Time of Stepwise Optimization

To summarize, these experiments demonstrate that, for compositions like the ACMEBOT case, usage of regular optimization can be suboptimal if the decision process takes too long. The proposed approach of stepwise optimization is better suited for such compositions. Alternatively, one could also try to prevent the problem of stale results by choosing quicker heuristic algorithms (e.g., GRASP with a very small number of start solutions) instead of MA as used in these experiments. A more comprehensive comparison between stepwise optimization and using a faster heuristic algorithm remains part of future research.

CHAPTER 6

Related Work

In the following chapter, the most important research work related to this thesis, or parts of it, is surveyed. Most attention is put on work that relates to more than one part of the thesis (e.g., monitoring and prediction), and on work that has received significant attention from the international research community.

6.1 Related Autonomous SLA Management Frameworks

This section discusses some research work that proposed frameworks for autonomous management of systems, with special focus on frameworks for SLA management.

SLA Management

One obvious cornerstone of the work presented in this thesis is the ability to formulate and negotiate SLAs. Currently, the languages most commonly used to these ends are WSLA [39, 96] and WS-Agreement [3]. WSLA is much more than just a language to define SLAs. In fact, WSLA is a full framework for defining, negotiating, monitoring and enacting agreements. For instance, the WSLA framework also contains elements to define SLAs by mapping partner-specific terminology, and to monitor the concrete execution of service-based systems for SLA violations. WS-Agreement originates from the area of Grid computing, and hence puts a strong focus on the requirements and specifics of the scientific computing domain. However, WS-Agreement is well-known for powerful mechanisms for SLA negotiation. One step beyond SLA negotiation is SLA meta-negotiation [21, 22], which discusses how service client and provider actually decide on the negotiation protocols used for SLA negotiation.

An earlier approach, which is less relevant today, is the SLAng language [167]. SLAng is an XML-based dialect for formulating SLAs. SLAng does not define a monitoring, enactment or negotiation infrastructure. Another earlier SLA management infrastructure is WSOI (Web Service Offerings Infrastructure) [171], which contains a Web Service Offerings Language (WSOL) [172]. In essence, WSOL is a language for specifying the features of a service along with the SLAs that the service promises to conform to. The PREvent framework does not define any SLA language of its own. In fact, all work presented in this thesis is SLA language agnostic, i.e., the framework can in principle be used with WSLA, WS-Agreement or any other language that follows the common base model as presented in Section 2.3.

Autonomic Computing

At its core, the PREvent system is a self-optimizing framework for service compositions. As such, the framework follows the seminal steps of the MAPE loop (see Section 2.4), as initially presented in [97]. Furthermore, the framework bears many features of autonomic and self-adaptive systems. More information on such systems can be found in [164], which provides an extensive survey of the current state of the art and research challenges in autonomic computing.

Autonomic Service-Based Systems

Of course, PREvent is not the first system that applies autonomic computing principles to SOA. The general mechanisms that lead to the development of self-adaptive service-based systems have been surveyed in [43]. Essentially, the paper states that the evolution of requirements, particularly business goals, leads to the need for highly dynamic and adaptable systems. A similar point is also made by [202]. This work argues, that dynamic SOAs are not manageable manually. Instead, an adaptive service-based software system (ASBS) needs to continually monitor its own state, and adapt by changing service configurations. The PREvent approach follows a very similar notion, even if adaptation is done differently. A formal approach for developing adaptable service-based systems is presented in [102], although the authors of this paper focus more on adaptability from a software design point of view, and less on self-adaptive systems.

Frameworks for Autonomic Service Compositions

In addition to these more conceptual contributions, various concrete systems have been proposed to implement adaptable service compositions. One prominent example is the MASC (Manageable and Adaptive Service Compositions) platform [52, 53]. In MASC, policy assertions, that define the correct behavior of the service composition, are specified in a WS-Policy dialect. MASC then aims to fulfill these assertions at runtime. MASC focuses more on functional aspects of the service composition, e.g., recovering from faults, while PREvent is geared towards non-functional aspects (SLA conformance). Hence, it can be said that MASC is a system for self-healing compositions, while PREvent is for self-optimization. An alternative system for self-healing is DISC (Declarative Integrated Self-Healing Web Services Composition) [205]. This framework's contribution is mainly the unification of process design, process verification and runtime monitoring. In that sense, the scope of DISC is quite different to PREvent, which does not cover process design or verification. However, unlike DISC, PREvent includes means to predict and prevent SLA violations at runtime. Finally, early work on adaptive service composition has been carried out within the eFlow project [26, 27]. While this project predated many current composition languages and notions, this work still introduced the idea of dynamic service compositions, which are able to transparently adapt to changes in the environment. However, as in [102], eFlows are more geared towards functional adaptation (that is, adaptation to changing business requirements) than to self-adaptation based on non-functional properties.

6.2 Related Work on Monitoring

In this section, some related research work on monitoring of service-based applications, that is, atomic Web services and service compositions, is discussed. Earlier research has already presented software patterns of how this monitoring can be implemented [141]. However, one should always keep in mind, that the process of monitoring itself is not for free. Contrary, monitoring often induces a severe overhead in itself. Hence, monitoring (for atomic Web services as well as for compositions) has to be restricted to the information that is strictly necessary to the application using the monitoring data [73].

Atomic QoS Monitoring

As discussed in Section 2.2, four basic models exist to monitor QoS of atomic Web services. The most simple and intuitive one is client-side monitoring, where QoS information is gathered by the service client. Essentially, this can either be done by evaluating the service invocations that the client issues as part of her regular service usage. Alternatively, the service client can issue artificial test requests to probe the service before using it for real. This has been proposed in [162]. In reality, clients will often aim to use regular invocation information if these data are sufficient, and issue probe requests if too little production data are available. This hybrid approach has been referred to as online testing in literature [74, 124]. Another approach, which can also be counted towards the class of client-side monitoring, is discussed in [132]. In this work, monitoring data are collected by weaving QoS monitoring aspects into clients using AspectJ[1].

The second common type of monitoring is server-side monitoring, where the Web service itself measures its quality. Evidently, this information is inherently unreliable for the client, hence, server-side monitoring information is enriched with client-side data [128].

The third approach is event-based monitoring, as presented for instance in [209]. The general principle of event-based QoS monitoring is, that the service-based application emits status events, which can be analyzed and aggregated into meaningful quality information. Another early approach to event-based monitoring has been presented in [163]. This work also introduces a simple SLA definition language.

Finally, the fourth type of QoS monitoring is monitoring based on the feedback provided by (human) service users. This has been explored in [88]. The approach presented therein takes reputation of clients and the overhead of data exchange into account. A similar feedback-based selection process has also been discussed in [108]. In this work, the focus is put on intuitive user interfaces for service selection based on past user experience. In general, [183, 184] surveys more approaches which deal with selecting Web services based on user feedback. In PREvent, event-based monitoring is used, conceptually quite similar to the ideas presented in [209].

[1] http://www.eclipse.org/aspectj/

Monitoring of Service Compositions

Less work has been published regarding the monitoring of entire service compositions (as opposed to monitoring of atomic services, which are being used as part of the composition). However, some important results in this regard have been reported in [9]. Monitoring information is collected by modifying the WS-BPEL process in a preprocessing step. The collected data are mainly used for functional verification, i.e., SLA monitoring is not in the scope of this work. Similar work is also contained in [7]. Emphasis is put on the distinction between instance-level monitoring (monitoring single composition instances) and class-level monitoring (monitoring of aggregated data). This is very much in line with the distinction between instance-level SLOs and aggregated SLOs, as presented in Section 2.3. As part of the S-Cube research project [123], these two approaches have recently been integrated [10, 12] into a common monitoring framework for WS-BPEL.

Additionally, monitoring service compositions also has some clear relationship to the idea of BAM. Essentially, monitoring of service compositions (especially if done in an event-based fashion) is a subset of the features that typical BAM solutions, such as the one sold by IBM [181], provide. However, BAM is mostly a monitoring tool for human business analysts, while this thesis discusses an autonomous framework (where no humans are directly in the loop).

Evidently, monitoring of service compositions also has other relevant use cases not directly related to PREvent. For instance, [17] has proposed monitoring as a viable tool to check for runtime compliance of service-based applications with government regulations.

6.3 Related Work on Service Composition Analysis and Prediction

Probably the most influential work concerning the analysis of service compositions is the idea of business process mining [176]. Business process mining aims at making business processes, service compositions, or workflows explicit from collected execution traces. These traces are often implemented by event logs, as produced by event-based monitoring. It should be noted that, while definitely being related to the research in this thesis, process mining is orthogonal to the goals of PREvent.

Prediction of SLA violations is similar to other types of quality prediction. One noteworthy example is finish time prediction [44], i.e., the prediction of how long a given process (for instance a business process) is going to take. In [44], regression models are used to calculate typical cycle times. This generic model has the advantage that it can be applied to almost any type of process (including service compositions). However, prediction models specialized on service compositions (see the discussion below) can arguably improve on this generic model. Another domain, where the proactive identification of performance problems has received considerable research interest, is the database domain [45]. However, most solutions in this domain are not directly applicable to service compositions.

Generally, research work on the prediction of SLA violations can be separated into two broad classes. On the one hand, there are explaining approaches, which analyze performance problems a posteriori, and mainly explain what has led to these problems. These approaches are helpful in the sense, that they inform a service provider about the weak points of the composition, allowing for composition optimization and redesign. On the other hand, there are predicting approaches, which aim at discovering potential problems a priori (before the problem has actually manifested). This has the advantage, that the problem may still be prevented. However, prediction is inherently uncertain (there may be a prediction error). This distinction has been first described in [28]. In the following, approaches in both classes are discussed.

Explaining Approaches

One typical approach in the class of explaining approaches is dependency analysis, as already presented in Section 4.2. Dependency analysis has first been introduced in [186, 187]. Dependency analysis aims at finding the factors that most often contribute to SLA violations using techniques from the area of machine learning. Similar goals also govern the Mode4SLA project, as presented in [18, 19]. However, Mode4SLA uses static analysis instead of machine learning. Another important difference is that Mode4SLA mostly uses a top-down approach (starting from the service composition the performance of the atomic services is analyzed), while dependency analysis uses a bottom-up approach (starting from monitored data, violations in the composition are explained).

Another group of research papers that can be attributed to the class of explaining approaches deals with QoS aggregation, as introduced in [84] and recently refined in [46]. QoS aggregation is the principle of estimating the quality of a service compostion by adding up (aggregating) the quality of all used atomic services. Obviously, different flow patterns and different QoS metrics require different aggregation functions (e.g., the response time of a `Sequence` activity is the sum of the response time of all atomic services, but the response time of a `Flow` activity is the maximum of the response times of all atomic services). These ideas have been generalized to SLA aggregation by different authors, including [174] and [71]. These approaches typically differ in the aggregation functions that are identified and, consequently, in the flow patterns that can be supported, but the general principle is usually very similar.

Preventing Approaches

While explaining approaches are still more prevalent in practice, preventing approaches are a very common research topic. Hence, a plethora of of approaches have been suggested in the last years. In PREvent, a machine learning based prediction approach is used, as discussed in Section 4.3. The PREvent approach builds on similar earlier work. Most importantly, these influencial earlier works include [28] and [210]. The main contribution of PREvent over these existing approaches is the inclusion of estimated and external data into the prediction, as well as the inclusion of PPM data. However, the PREvent framework could also be built on top of any of these earlier prediction models. Another approach similar to the PREvent model is discussed in [95]. In this work, the idea of dependency analysis, as discussed in Section 4.2, is extended to also include automated triggering of adaptations, based on the identified factors of influence. In this sense, this work is obviously similar to PREvent, but it does not include as sophisticated means to decide on the best actions to apply.

Alternatively, SLA violations can also be predicted using runtime verification [62] or static analysis of the service composition [82]. These approaches are different from the machine learning based ones in the sense, that they do not base predictions on monitored runtime data. Instead, they look at the structure of the service composition to infer quality properties. Finally, the online testing approaches mentioned before [74, 124] can also be considered an alternative approach to quality prediction. Similar to the machine learning based approaches, online testing is data based. However, data processing is usually performed in a relatively simple rule-based fashion in these approaches.

6.4 Related Work on Optimization of Service Compositions

Optimization is at the core of any autonomous system. It enables decisions, which promise to improve the currently sensed system state into a better one. Hence, it is obvious that optimization is also central for PREvent, as well as for many other systems in services research. However, the general idea of analysis, prediction and optimization has alredy been successfully applied to the real world, for instance within iBOM [29], a platform for business operations management. In the following section, an overview over various strategies of QoS optimization is given. Afterwards, some work related to the cost-benefit tradeoff discussed in Section 4.4 is presented.

QoS Optimization

In its most general form, optimization of service compositions is maybe the most prevalent research topic in Web services research as of this writing. Most commonly, this research considers the instantiation of abstract service compositions using a pool of candidate atomic services, i.e., to each service invocation in the abstract composition a concrete atomic service from the pool is assigned. Usually, this instantiation is done in such a way, that global constraints on QoS and other metrics (e.g., costs) are satisfied. In research, this process is often referred to as QoS optimization. QoS optimization can be done in a multitude of ways, with the earliest and maybe still most well-known being Integer Linear Programming [208] (ILP) and variations thereof (e.g., H1_RELAX_IP [15]). The advantage of ILP is that the model is relatively simple, and ILPs can be solved quickly and deterministically using the well-known Simplex algorithm up to modest problem sizes. However, for very large abstract service compositions, ILP does not scale. For such cases, many researchers have experimented with different heuristics and meta-heuristics. For instance, heuristic search (similar to the Local Optimization algorithm used in PREvent) has been proposed to solve this problem in [125, 156]. Another example is the immune algorithm, as discussed in [199]. The immune algorithm is a biologically-inspired optimization technique, which imitates the biological immune system. This algorithm has the advantage of a multimodal target function, i.e., the algorithm is able to optimize different (usually conflicting) goals at the same time. Another frequently used biologically inspired algorithm is GA [161], which has also been made use of in PREvent. Similar to the PREvent approach, the performance of the generic GA is improved by incorporating ideas of local optimization of individuals in each generation. Some custom heuristics for the problem are proposed in [204]. Additionally, this work details two different ways of how the QoS optimization problem can be formulated. In more recent papers, there seems to be a trend towards using a combination of different techniques. One example of such an approach is [2], where large compositions are split into small parts, each of which is optimized locally. The result of this local optimizations is then again optimized from a global perspective. Similar to most approaches discussed so far, this local/global optimization is not deterministic, as it is not guaranteed that the combination of optimal local results is also globally optimal. Finally, some authors propose a probabilistic service selection process for QoS optimization [98]. In such an approach, not every instance of a composition uses the same services. Instead, every service from the candidate pool is used with a given probability. These probabilities are adapted based on how well the service performs when it is selected. So far, there is no agreement yet in the research community about which of the multitude of approaches is the most suitable one for dealing with real-life compositions. QoS optimization is related to the decision of which adaptation action to apply, as discussed in Section 4.4, and similar algorithms can be used to solve both problems. However, the actual optimization problem that is being solved in this thesis is distinctly different.

Orthogonal to developing algorithms for solving the QoS optimization problem, different authors have worked on suitable languages to formulate the abstract compositions that are being optimized. One straight-forward approach is to use an abstraction of WS-BPEL, as done for instance in BPELLight [139]. A different path has been chosen in [158, 159], where VCL (the Vienna Composition Language) has been proposed as a Domain Specific Language (DSL) for defining abstract compositions, which are then instantiated using information from a VRESCo service registry [129]. Recently, some researchers have experimented with radically different approaches towards modelling compositions, such as the notion of service composition using chemical programming [6, 41]. It is not yet clear, whether this direction will be able to have a lasting impact on service compositions in industrial practice.

Cost-Benefit Tradeoff

Generally speaking, Section 4.4 describes the common business scenario of a cost-benefit tradeoff, i.e., a scenario where certain beneficial actions (adaptations which prevent SLA violations) are possible, but each action also generates a negative side-effect (in this case, every adaptation action generates costs). The tradeoff is now to decide on the best combination of actions, which maximizes the benefit, but at the same time also minimizes the negative side-effects. Similar problems also exist in many other domains. One example of a similar problem in a related domain is revenue optimization in hosting [119]. Similar to the work conducted in PREvent, this paper deals with preventing SLA violations using runtime adaptation. However, the domain in the case of this paper is not service compositions, but Web service hosting. Similar work has earlier been presented for application servers in [177] as well as for enterprise services [64]. While these approaches are interesting, they are not directly relevant for PREvent, as the problem of optimizing instances of service compositions significantly differs to the problem of optimally aligning service hosts or application servers.

In service-based systems, researchers have identified various other tradeoffs with characteristics similar to the cost-benefit tradeoff. For instance, in [203] the tradeoff between system performance and security is discussed. Similarly, [211] mentions the tradeoff between better QoS and better QoS monitoring (as monitoring generally introduces some overhead, monitoring itself typically decreases QoS). The way how these decisions are modelled is usually similar to the formalization of the optimization problem in Section 4.4.

6.5 Related Work on Adaptation

One of the core promises of SOA has always been enabling a high degree of adaptability and dynamicity. Therefore, it is not surprising that from early on, research has focused on mechanisms for adapting service-based applications at runtime. As discussed in Section 4.5, this adaptation can happen at multiple levels. Most importantly, adaptation on service level (adaptation by service rebinding) and adaptation on composition level (adaptation by changing the composition structure) can be distinguished. In the following, important approaches on both levels are discussed. However, note that many frameworks discussed below exhibit some characteristics of both classes, i.e., many frameworks can be used to implement both types of adaptation. Strictly speaking, adaptation on composition structure level is a superset of service rebinding, hence all frameworks that can implement adaptation of the composition structure are inherently also able to implement service rebinding.

Service Rebinding

In a simplistic manner, service rebinding can in fact be carried out directly with WS-BPEL alone, using dynamic partner links. However, practical problems, such as finding the right service to bind to (based on QoS), or the need to resolve interface differences, often demand for more sophisticated service rebinding approaches. Such mechanisms are, for instance, provided in the PAWS (Processes with Adaptive Web Services) framework [4]. PAWS processes dynamically self-optimize by dynamically selecting Web services from a candidate pool based on QoS information. Similar goals have also guided the development of the WS-Binder framework [149]. As a third example, others have presented an unnamed framework for improving the dynamic service binding capabilities of WS-BPEL in [133]. The most important contribution of this work, as compared to the others, is that this approach is transparent both to the composition developer and to the composition engine. Additionally, unlike the other mentioned approaches, this work is able to rebind stateful services. In PREvent, the DAIOS framework is used to implement dynamic service invocation, and, hence, service rebinding. One advantage of DAIOS as compared to related work is that DAIOS is inherently independent from any concrete composition language, and could be used with various models (even though the prototype implementation of this thesis was done in C#).

Service rebinding directly leads to the problem of service mediation, i.e., the problem of modifying a service invocation originally meant for one service in such a way that it now represents a semantically equivalent invocation of another service. Essentially, two types of mediation can be distinguished. On the one hand, interface mediation is the process of mapping single service invocations (or message), while protocol mediation maps MEPs between client and service. For interface mediation, a common approach is to use adapters, e.g., [30, 117]. Adapters are small software components that typically reside on client-side. These adapters receive service invocations and the service that the invocation is targeted to, and modify the message so that it has the correct format for this service. Today, in many industry applications, the role of the adapter is implemented by the ESB, which for instance uses XSLT to transform SOAP messages. A similar XSLT based mediation approach has also been presented in [132]. Contrary, in research, semantic Web services [120] technology is often seen as state-of-the-art for mediation. Early work on dynamic invocation of semantic services, including interface adaptation, has been presented in [49]. In this paper, the Resource Description Framework (RDF) is used as a data integration language. More widely known is the WSMO (Web Service Modeling Ontology) project [34]. One of the core elements of WSMO are mediators, which connect WSMO clients with semantic Web services. Mediators implement message transformation based on defined ontologies. Outside of semantic technologies, interface mediation techniques have also been proposed for other domains, for instance Grid computing [170] or context-aware computing [136]. Protocol mediation has also been extensively researched in the past. For instance, [14] identified mismatch patterns for message exchange patterns and proposed generic solutions for each mismatch. [47] specified a visual notation for mapping between services with different protocols. Finally, semantic technology has also been used for protocol mediation, for instance in [191]. In PREvent, mediation is only done on interface level. If protocol mediation (reflected by the adaptation type Substitution With Subflow in Section 4.5) is necessary the fragment-based adaptation approach can be used (i.e., essentially rebinding with protocol mediation is covered using the same methods as structural adaptation).

Structural Adaptation

Structural changes of service compositions are a fundamentally different type of adaptation, as compared to service rebinding. Hence, this type of adaptation comes with its own set of research challenges and problems. A relatively simplicistic structural adaptation approach is parametrization, as discussed in [93]. Parametrization does not allow for arbitrary modifications of the service composition. Instead, some variation points are already statically built into the composition. At adaptation time, this predefined variations can be triggered. Evidently, only adaptations, that have been foreseen when the composition has been designed, can be handled this way. For more arbitrary runtime adaptations, the AOP paradigm is widely used. Systems that implement structural adaptation in that way include AO4BPEL [31, 32] and BPEL'n'Aspects [92]. While these two approaches have overlapping goals, the latter approach is more open in the way that adaptations are defined, while the former is better integrated with monitoring facilities. In addition to these general-purpose structural adaptation frameworks, some research has been conducted on domain-specific solutions, for example for the telecommunications domain [138]. In PREvent, structural adaptation is also implemented in an aspect-based fashion. In fact, the fragment-based approach used in this thesis is conceptually built on top of BPEL'n'Aspects, but integrated with the PREvent tooling for predicting and preventing SLA violations. The general idea of process fragments has also been previously discussed [50], but with a completely different focus (outsourcing). It should be noted that the AOP paradigm has also successfully been applied to the adaptation of atomic Web services, for instance in [99, 137, 169]. However, as PREvent and this thesis do not directly deal with this type of adaptation (services are taken as a black box, and are not adapted), this is not directly relevant here.

CHAPTER 7

Conclusions

In this final chapter, the main results of this thesis are summarized. Main focus is put on the main research results that have been discussed, and how the state of the art in research has been advanced as part of this thesis work. Furthermore, the research questions, that have been introduced in Section 1.2, are revisited, critically analyzing to what extend those questions could be answered in this thesis. Finally, the thesis is concluded with an outlook on future research directions, which are derived from the aspects of the work that could not be sufficiently answered in the time frame of this thesis.

7.1 Summary

This thesis has introduced the PREvent framework, an autonomic framework for cost-optimal SLA management of service compositions. PREvent follows the seminal four steps of the MAPE loop: monitoring of the service composition (acquiring the necessary base data to decide on the performance of a composition instance), analyzing the monitored data (predicting violations), planning preventive actions (deciding on the optimal set of adaptation actions to apply) and executing the adaptation (applying these actions to the running composition instance).

Monitoring in PREvent is based on CEP techniques. The approach is able to integrate more common QoS information with domain-specific PPMs and external data. Predictions are generated using machine learning technqiues. More concretely, predictions of nominal SLOs are generated using decision trees, while ANNs are used for continuous SLOs. The decision process of selecting the best adaptation actions to prevent given violations has been modelled as a one-dimensional optimization problem (minimizing the total costs for the service provider). This problem cannot be solved with analytical means. Hence, a number of efficient algorithms, both deterministic and heuristic, have been proposed to find suitable solutions at runtime. However, even by using efficient meta-heuristic algorithms, the time necessary to solve the optimization problem remains a critical factor for shorter-running compositions. If the optimization takes too long, stale optimization results (decisions that have been produced too late) invalidate the optimization. In this thesis, a stepwise optimization approach has been proposed to prevent stale results. Finally, PREvent supports adaptation of running instances on service rebinding and structural level. For service rebinding, the DAIOS dynamic Web service invocation framework is used to dynamically exchange endpoints. Using the lightweight semantic model provided by VRESCo, it is also possible to mediate between services with different technical interfaces. For adaptations on structural level a fragment-based adaptation model has been proposed. This approach uses the notions of AOP to define runtime changes in the definition model of service compositions.

All aspects discussed above have been implemented in a C#/.NET based prototype tool. This implementation has been extensively evaluated, both quantitatively and qualitatively. In these experiments, it has been shown that the concepts introduced in this thesis are able to reduce both the total costs for the provider of a composite service and the number of SLA violations, exemplified based on the case of an assembling composite service (the ACMEBOT case study).

7.2 Research Questions Revisited

Section 1.2 introduced the research questions that guided the work on this thesis. In this section, these questions are now revisited. For both questions, it is summarized how they have been answered within PREvent, and what limitations there are with regard to the PREvent solution.

Research Question I:
How can SLA violations in service compositions be prevented in an ex ante way?

As summarized in Section 7.1, PREvent implements an autonomic system, which is able to predict and prevent violations in running service compositions. However, all current results are strictly limited to the scope of single instances. Using the techniques researched in this thesis, it is not possible to manage aggregated SLOs, as defined in Section 2.3. For such SLOs, it is necessary to not only consider monitoring data of single instances, but also include information such as trends or seasons. Furthermore, for aggregated SLOs different adaptation actions need to be developed, as all adaptations discussed in this thesis consider single instances.

Research Question II:
How can the costs of adaptation be considered in the action selection process of ex ante prevention of SLA violations?

In PREvent, adaptation costs are included in the action selection process of adaptation actions, as costs are explicitly included in the target function of the underlying optimization problem. Using the target function presented in this thesis, costs and benefits of adaption are distilled into a single, meaningful number for the service provider (total costs), on which the action selection can be based. However, the formalization presented in this thesis ignores one factor, which may be important to some service providers. Namely, the current formalization considers penalty payments as the only costs incurred by violating SLAs. In practice, such violations also have other undesired results as well, such as reduced satisfaction of service consumers. These aspects of SLA violations are currently not captured in the optimization model, and hence ignored in the decision process.

7.3 Future Work

Based on the discussion in Section 7.2, it is evident that some important aspects of cost-based optimization were out of scope for this thesis. These aspects open up possibilities for future research:

- For practical reasons, the generalization of the PREvent approach to aggregated SLOs seems most important. Unfortunately, this generalization is not straight-forward, as the machine learning based prediction approach (which is one of the cornerstones of the current PREvent approach) cannot easily be applied to aggregated SLOs. One feasible approach to predict aggregated SLOs is time series analysis, for instance using the the Box-Jenkins methodology [20] (autoregressive moving averages) or using Hidden Markov Models (HMMs) for time series analysis [212].

- The issues discussed above are not only relevant for aggregated SLOs. Indeed, even for instance-level SLOs, it may prove advantageous to also take other instances into account at prediction-time. For instance, if an unusually long execution time is predicted for one instance (e.g., because some of the base services are currently rather unresponsive), it is a reasonable assumption that similar problems will also be evident in other currently running instances.

- However, as indicated above, generalizing the prediction approach alone is not sufficient for full support of aggregated SLOs. In addition, it is necessary to also develop new adaptation actions, which adapt not single instances, but the service composition itself. For some actions, this is trivial, e.g., instead of rebinding a service in one instance it can be rebound for all future instances, or instead of changing the composition structure for one instance it can be changed for all future instances. However, there are also entirely new adaptation actions possible, such as upgrading the server machine running the service composition. With the advent of cloud computing [5, 75], such adaptations have now become feasible at runtime.

- Additionally, the optimization problem used in PREvent needs to be generalized to include additional cost factors besides SLA penalties, such as reduced customer satisfaction or damaged reputation. One simple approach would be to quantify these aspects (that is, express them in monetary terms) and include them as additional cost terms into the target function. However, quantifying these terms may not always be easy. Additionally, these cost factors may not be constant. To give one example, the (additional) cost of violating the SLA of a given customer for the first time ever may be small (the customer may interpret the occurrence as a one-time lapse), but if many such violations follow, the customer may become increasingly dissatisfied with the provider and start searching for alternatives. This also opens up an entirely new angle of optimization: is it better to repeatedly violate the SLAs of single customers (and preventing violations for all others), or to distribute violations evenly among all customers (hence, reducing the satisfaction of all customers, but to a lesser degree)?

- Finally, an important continuing effort is the further dissemination and improvement of Contribution V of this thesis (the ACMEBOT case study). To this end, joint effort of members of the S-Cube Network of Excellence [123], along with practitioners from the transports and logistics domain, is expected to lead to a more realistic update of both, the business process (as presented in Chapter 3) and the experimental implementation of the process (as discussed in Section 5.1), which should reflect real-world business dependencies more accurately. This improved process will then be published in a high-profile venue, in order to establish the ACMEBOT case as a benchmark for evaluation of future research work in the wider field of adaptive service-based applications. Evidently, these benchmarks will also be used to further validate the PREvent framework.

It is expected that further work on cost-based optimization will follow the general lines of these open issues.

Bibliography

[1] Business Process Modeling Notation Specification. Technical report, Object Management Group (OMG), 2006. http://www.omg.org/bpmn/Documents/OMG_Final_Adopted_BPMN_1-0_Spec_06-02-01.pdf, Last Visited: 2011-07-19. → pages 8, 19

[2] Mohammad Alrifai and Thomas Risse. Combining Global Optimization With Local Selection for Efficient QoS-Aware Service Composition. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 881–890, New York, NY, USA, 2009. ACM. → pages 103

[3] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Open Grid Forum (OGF), 2006. http://www.gridforum.org/documents/GFD.107.pdf, Last Visited: 2011-07-19. → pages 12, 97

[4] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. PAWS: A Framework for Executing Adaptive Web-Service Processes. *IEEE Software*, 24(6):39–46, 2007. → pages 3, 14, 104

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010. → pages 57, 109

[6] Jean-Pierre Banâtre, Thierry Priol, and Yann Radenac. Service Orchestration Using the Chemical Metaphor. In *Proceedings of the 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'08)*, pages 79–89, Berlin, Heidelberg, 2008. Springer-Verlag. → pages 103

[7] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 63–71, Washington, DC, USA, 2006. IEEE Computer Society. → pages 100

[8] L Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Toward Open-World Software: Issue and Challenges. *IEEE Computer*, 39(10):36–43, 2006. → pages 66

[9] Luciano Baresi and Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the 3rd International Conference of Service-Oriented Computing (ICSOC'05)*, pages 269–282, Berlin, Heidelberg, 2005. Springer-Verlag. → pages 100

[10] Luciano Baresi, Sam Guinea, Raman Kazhamiakin, and Marco Pistore. An Integrated Approach for the Run-Time Monitoring of BPEL Orchestrations. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet (ServiceWave'08)*, pages 1–12, Berlin, Heidelberg, 2008. Springer-Verlag. → pages 100

[11] Luciano Baresi, Sam Guinea, and Liliana Pasquale. Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine. In *Proceedings of the International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE'07)*, pages 11–20, New York, NY, USA, 2007. ACM. → pages 14

[12] Luciano Baresi, Sam Guinea, Marco Pistore, and Michele Trainotti. Dynamo + Astro: An Integrated Approach for BPEL Monitoring. In *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*, pages 230–237, Washington, DC, USA, 2009. IEEE Computer Society. → pages 11, 100

[13] Thomas Barothy, Markus Peterhans, and Kurt Bauknecht. Business Process Reengineering: Emergence of a New Research Field. *SIGOIS Bulletin*, 16:3–10, August 1995. → pages 1

[14] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing Adapters for Web Services Integration. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE 2005)*, Lecture Notes in Computer Science, 2005. → pages 105

[15] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for QoS-aware Web Service Composition. In *Proceedings of the 2006 IEEE International Conference on Web Services (ICWS'06)*, pages 72–82, Los Alamitos, CA, USA, 2006. IEEE Computer Society. → pages 102

[16] Dinabandhu Bhandari and C. A. Murthy. Genetic Algorithm with Elitist Model and Its Convergence. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(6):731–747, 1996. → pages 47, 51

[17] Aliaksandr Birukou, Vincenzo D'Andrea, Frank Leymann, Jacek Serafinski, Patrícia Silveira, Steve Strauch, and Marek Tluczek. An Integrated Solution for Runtime Compliance Governance in SOA. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10)*, pages 122–136. Springer, 2010. → pages 100

[18] Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, and Michael Jaeger. Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In *Proceedings of the*

2008 IEEE International Conference on Services Computing (SCC'08), pages 21–29, Washington, DC, USA, 2008. IEEE Computer Society. → pages 101

[19] Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, and Michael C. Jaeger. Analyzing Impact Factors on Composite Services. In *Proceedings of the 2009 IEEE International Conference on Services Computing (SCC '09)*, pages 218–226, Los Alamitos, CA, USA, 2009. IEEE Computer Society. → pages 2, 101

[20] George Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 1970. → pages 109

[21] Ivona Brandic, Dejan Music, Philipp Leitner, and Schahram Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. In *Proceedings of the 6th International Workshop on Grid Economics and Business Models*, pages 60–73, Berlin, Heidelberg, 2009. Springer-Verlag. → pages 13, 24, 97

[22] Ivona Brandic, Srikumar Venugopal, Michael Mattess, and Rajkumar Buyya. Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services. In *Proceedings of the Workshop on Service-Oriented Engineering and Optimizations*, 2008. → pages 13, 97

[23] Leo Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996. → pages 39

[24] Paul A. Buhler, Christopher W. Starr, William H. Schroder, and José M. Vidal. Preparing for Service-Oriented Computing: A Composite Design Pattern for Stubless Web Service Invocation. In *Proceedings of the International Conference on Web Engineering (ICWE)*, pages 603–604, 2004. → pages 59

[25] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. QoS-Aware Replanning of Composite Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 121–129, Washington, DC, USA, 2005. IEEE Computer Society. → pages 3

[26] Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and Dynamic Service Composition in eFlow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, pages 13–31, London, UK, 2000. Springer-Verlag. → pages 98

[27] Fabio Casati and Ming-Chien Shan. Event-Based Interaction Management for Composite E-Services in eFlow. *Information Systems Frontiers*, 4:19–31, April 2002. → pages 98

[28] Malu Castellanos, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. Intelligent Management of SLAs for Composite Web Services. In *Databases in Networked Information Systems*, 2003. → pages 101

[29] Malu Castellanos, Fabio Casati, Ming-Chien Shan, and Umeshwar Dayal. iBOM: A Platform for Intelligent Business Operation Management. In *Proceedings of the 21st*

International Conference on Data Engineering (ICDE'05), pages 1084–1095, Washington, DC, USA, 2005. IEEE Computer Society. → pages 102

[30] Luca Cavallaro and Elisabetta Di Nitto. An Approach to Adapt Service Requests to Actual Service Interfaces. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'08)*, pages 129–136, New York, NY, USA, 2008. ACM. → pages 104

[31] Anis Charfi, Tom Dinkelaker, and Mira Mezini. A Plug-in Architecture for Self-Adaptive Web Service Compositions. In *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*, pages 35–42, Washington, DC, USA, 2009. IEEE Computer Society. → pages 105

[32] Anis Charfi and Mira Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web*, 10:309–344, September 2007. → pages 105

[33] Issam Chebbi, Schahram Dustdar, and Samir Tata. The View-Based Approach to Dynamic Inter-Organizational Workflow Cooperation. *Data and Knowledge Engineering*, 56:139–173, February 2006. → pages 7

[34] Emilia Cimpian, Adrian Mocan, and Michael Stollberg. Mediation Enabled Semantic Web Services Usage. In *Proceedings of the Asian Semantic Web Conference (ASWC)*, volume 4185, pages 459–473, 2006. → pages 105

[35] Benoit Claudel, Noël De Palma, Renaud Lachaize, and Daniel Hagimont. Self-Protection for Distributed Component-Based Applications. In *Proceedings of the 8th International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, pages 184–198, Berlin, Heidelberg, 2006. Springer-Verlag. → pages 14

[36] Gregory Cobena, Serge Abiteboul, and Amelie Marian. Detecting Changes in XML Documents. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 41, Washington, DC, USA, 2002. IEEE Computer Society. → pages 88

[37] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20:273–297, September 1995. → pages 16

[38] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6:86–93, March 2002. → pages 1

[39] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard P. King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer, and Alaa Youssef. Web Services on Demand: WSLA-Driven Automated Management. *IBM Systems Journal*, 43:136–158, January 2004. → pages 12, 97

[40] Thomas H. Davenport and James E. Short. The New Industrial Engineering: Information Technology and Business Process Redesign. *Sloan Management Review*, pages 11–27, 1990. → pages 1

[41] Claudia Di Napoli, Maurizio Giordano, Zsolt Németh, and Nicola Tonellotto. Using Chemical Reactions to Model Service Composition. In *Proceeding of the Second International Workshop on Self-Organizing Architectures (SOAR'10)*, pages 43–50, New York, NY, USA, 2010. ACM. → pages 103

[42] Elisabetta Di Nitto, Massimiliano Di Penta, Alessio Gambi, Gianluca Ripa, and Maria Luisa Villani. Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, pages 295–306, Berlin, Heidelberg, 2007. Springer-Verlag. → pages 13, 24

[43] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A Journey to Highly Dynamic, Self-Adaptive Service-Based Applications. *Automated Software Engineering*, 15:313–341, December 2008. → pages 98

[44] B. F. Dongen, R. A. Crooy, and W. M. Aalst. Cycle Time Prediction: When Will This Case Finally Be Finished? In *Proceedings of the 2008 OTM Confederated International Conferences*, pages 319–336, Berlin, Heidelberg, 2008. Springer-Verlag. → pages 100

[45] Songyun Duan and Shivnath Babu. Proactive Identification of Performance Problems. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*, pages 766–768, New York, NY, USA, 2006. ACM. → pages 100

[46] Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, Yong Yang, and Liang Zhang. Aggregate Quality of Service Computation for Composite Services. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10)*, pages 213–227. Springer, 2010. → pages 101

[47] Marlon Dumas, Murray Spork, and Kenneth Wang. Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In *Proceedings of the International Conference Business Process Management (BPM'06)*, volume 4102 of *Lecture Notes in Computer Science*, pages 65–80, 2006. → pages 105

[48] Schahram Dustdar and Wolfgang Schreiner. A Survey on Web Services Composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005. → pages 1

[49] Andreas Eberhart. Ad-hoc Invocation of Semantic Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 116–, Washington, DC, USA, 2004. IEEE Computer Society. → pages 105

[50] Hanna Eberle, Tobias Unger, and Frank Leymann. Process Fragments. In *Proceedings of the 2009 OTM Confederated International Conferences*, volume 5870 of *Lecture Notes in Computer Science*, pages 398–405. Springer, 2009. → pages 105

[51] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. → pages 1

[52] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Policy-Driven Middleware for Self-Adaptation of Web Services Compositions. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (Middleware'06)*, pages 62–80, New York, NY, USA, 2006. Springer-Verlag New York, Inc. → pages 98

[53] Abdelkarim Erradi, Vladimir Tosic, and Piyush Maheshwari. MASC - .NET-Based Middleware for Adaptive Composite Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'07)*, pages 727–734, Los Alamitos, CA, USA, 2007. IEEE Computer Society. → pages 98

[54] Usama M Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. *From Data Mining to Knowledge Discovery: an Overview*, chapter 1, pages 1–34. American Association for Artificial Intelligence, 1996. → pages 54

[55] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995. → pages 46

[56] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, CA, 2000. → pages 58

[57] Nancy Forbes. *Imitation of Life - How Biology Is Inspiring Computing*. MIT Press, 2004. → pages 16

[58] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory (COLT '90)*, pages 202–216, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. → pages 39

[59] Yoav Freund and Llew Mason. The Alternating Decision Tree Learning Algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, pages 124–133, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. → pages 29, 35

[60] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29:131–163, November 1997. → pages 16

[61] A. G. Ganek and T. A. Corbi. The Dawning of the Autonomic Computing Era. *IBM Systems Journal*, 42:5–18, January 2003. → pages 14

[62] Andreas Gehlert, Antonio Bucchiarone, Raman Kazhamiakin, Andreas Metzger, Marco Pistore, and Klaus Pohl. Exploiting Assumption-Based Verification for the Adaptation of Service-Based Applications. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10)*, pages 2430–2437, New York, NY, USA, 2010. ACM. → pages 101

[63] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, April 1995. → pages 1

[64] Daniel Gmach, Stefan Krompass, Andreas Scholz, Martin Wimmer, and Alfons Kemper. Adaptive Quality of Service Management for Enterprise Services. *ACM Transactions on the Web*, 2(1):1–46, 2008. → pages 103

[65] D. E. Goldberg and K. Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms*, pages 69–93, 1991. → pages 47, 48

[66] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989. → pages 46

[67] Sherif A. Gurguis and Amir Zeid. Towards Autonomic Web Services: Achieving Self-Healing using Web Services. *SIGSOFT Software Engineering Notes*, 30:1–5, May 2005. → pages 14

[68] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18, 2009. → pages 34

[69] M. Hammer. Reengineering Work: Don't Automate, Obliterate. *Harvard Business Review*, pages 104–112, 1990. → pages 1

[70] R. W. Hamming. Error Detecting and Error Correction Codes. *Bell Systems Technical Journal*, 29(2):147–160, 1950. → pages 45

[71] Irfan Haq, Altaf Huqqani, and Erich Schikuta. Aggregating Hierarchical Service Level Agreements in Business Value Networks. In *Proceedings of the 7th International Conference on Business Process Management (BPM'09)*, pages 176–192, Berlin, Heidelberg, 2009. Springer-Verlag. → pages 101

[72] Simon Haykin. *Neural Networks and Learning Machines: A Comprehensive Foundation*. Prentice Hall, third edition, 2008. → pages xv, 16, 17, 30

[73] Garth Heward, Jun Han, Ingo Müller, Jean-Guy Schneider, and Steven Versteeg. Optimizing the Configuration of Web Service Monitors. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10)*, pages 587–595. Springer, 2010. → pages 99

[74] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet (ServiceWave'08)*, pages 122–133, Berlin, Heidelberg, 2008. Springer-Verlag. → pages 11, 99, 102

[75] D. Hilley. Cloud Computing: A Taxonomy of Platform and Infrastructure-Level Offerings, 2009. → pages 109

[76] Michael N. Huhns and Munindar P. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, 9:75–81, January 2005. → pages 1, 7

[77] Waldemar Hummer, Philipp Leitner, and Schahram Dustdar. SEPL – A Domain-Specific Language and Execution Environment for Protocols of Stateful Web Services. *Distributed and Parallel Databases*, 29:277–307, August 2011. → pages 10

[78] Waldemar Hummer, Philipp Leitner, and Schahram Dustdar. WS-Aggregation: Distributed Aggregation of Web Services Data. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC'11)*, pages 1590–1597, New York, NY, USA, 2011. ACM. → pages 26

[79] Waldemar Hummer, Philipp Leitner, Anton Michlmayr, and Florian Rosenberg. *VRESCo – Vienna Runtime Environment for Service-Oriented Computing*, pages 299–324. Springer Berlin, Heidelberg, 2010. → pages 60

[80] M. Hutter. Fitness Uniform Selection to Preserve Genetic Diversity. In *Proceedings of the World on Congress on Computational Intelligence*, pages 783–788, 2002. → pages 48

[81] Manuel Iori. Metaheuristic Algorithms for Combinatorial Optimization Problems. *A Quarterly Journal of Operations Research (4OR)*, 3(2):163–166, 2005. → pages 45

[82] Dragan Ivanović, Manuel Carro, and Manuel Hermenegildo. An Initial Proposal for Data-Aware Resource Analysis of Orchestrations with Applications to Predictive Monitoring. In *Proceedings of the 2009 International Conference on Service-Oriented Computing (ICSOC'09)*, pages 414–424, Berlin, Heidelberg, 2009. Springer-Verlag. → pages 101

[83] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, 1993. → pages 18, 29

[84] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. QoS Aggregation for Web Service Composition using Workflow Patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04)*, pages 149–159, Washington, DC, USA, 2004. IEEE Computer Society. → pages 54, 101

[85] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. QoS Aggregation in Web Service Compositions. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 181–185. IEEE Computer Society, 2005. → pages 54

[86] Anil K. Jain, Jianchang Mao, and K. M. Mohiuddin. Artificial Neural Networks: A Tutorial. *IEEE Computer*, 29:31–44, March 1996. → pages 16, 17

[87] Kenneth De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2002. $\rightarrow$ pages 46

[88] Radu Jurca, Boi Faltings, and Walter Binder. Reliable QoS Monitoring Based on Client Feedback. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 1003–1012, New York, NY, USA, 2007. ACM. $\rightarrow$ pages 11, 99

[89] Lukasz Juszczyk and Schahram Dustdar. Script-Based Generation of Dynamic Testbeds for SOA. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*, pages 195–202, Washington, DC, USA, 2010. IEEE Computer Society. $\rightarrow$ pages 33, 87

[90] Lukasz Juszczyk, Hong-Linh Truong, and Schahram Dustdar. GENESIS - A Framework for Automatic Generation and Steering of Testbeds of ComplexWeb Services. In *Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*, pages 131–140. IEEE Computer Society, March 2008. $\rightarrow$ pages 87

[91] Arthur B. Kahn. Topological Sorting of Large Networks. *Communications of the ACM*, 5:558–562, November 1962. $\rightarrow$ pages 55

[92] Dimka Karastoyanova and Frank Leymann. BPEL'n'Aspects: Adapting Service Orchestration Logic. In *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*, pages 222–229, Washington, DC, USA, 2009. IEEE Computer Society. $\rightarrow$ pages 105

[93] Dimka Karastoyanova, Frank Leymann, Jörg Nitzsche, Branimir Wetzstein, and Daniel Wutke. Parameterized BPEL Processes: Concepts and Implementation. In *Proceedings of the International Conference Business Process Management (BPM'06)*, 2006. $\rightarrow$ pages 57, 105

[94] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990. $\rightarrow$ pages 16

[95] Raman Kazhamiakin, Branimir Wetzstein, Dimka Karastoyanova, Marco Pistore, and Frank Leymann. Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In *Proceedings of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, pages 395–404, 2009. $\rightarrow$ pages 3, 101

[96] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal on Network and Systems Management*, 11:57–81, March 2003. $\rightarrow$ pages 2, 11, 12, 13, 97

[97] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003. $\rightarrow$ pages xv, 5, 13, 14, 15, 23, 98

[98] Adrian Klein, Fuyuki Ishikawa, and Shinichi Honiden. Efficient QoS-Aware Service Composition with a Probabilistic Service Selection Policy. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10)*, pages 182–196. Springer, 2010. $\rightarrow$ pages 103

[99] Woralak Kongdenfha, Hamid Reza Motahari-Nezhad, Boualem Benatallah, Fabio Casati, and Regis Saint-Paul. Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters. *IEEE Transactions on Services Computing*, 2(2):94–107, 2009. $\rightarrow$ pages 88, 105

[100] Jacek Kopecky, Dumitru Roman, Matthew Moran, and Dieter Fensel. 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)*, page 127, Washington, DC, USA, 2006. IEEE Computer Society. $\rightarrow$ pages 79

[101] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11:60–67, November 2007. $\rightarrow$ pages 9

[102] Leen Lambers, Leonardo Mariani, Hartmut Ehrig, and M Pezzè. A Formal Framework for Developing Adaptable Service-Based Applications. In *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE)*, pages 1–15, 2008. $\rightarrow$ pages 98, 99

[103] E. L. Lawler and D. E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, 1966. $\rightarrow$ pages 42

[104] P Leitner, B Wetzstein, F Rosenberg, A Michlmayr, S Dustdar, and F Leymann. Runtime Prediction of Service Level Agreement Violations for Composite Services. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, pages 176–186, Berlin, Heidelberg, 2009. Springer-Verlag. $\rightarrow$ pages 4, 5

[105] Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. Cost-Based Optimization of Service Compositions. *IEEE Transactions on Services Computing (TSC)*, 2011. To appear. $\rightarrow$ pages 4, 5, 84

[106] Philipp Leitner, Waldemar Hummer, Benjamin Satzger, and Schahram Dustdar. Stepwise and Asynchronous Runtime Optimization of Web Service Compositions. In *To appear at The 12th International Conference on Web Information System Engineering (WISE 2011)*, 2011. $\rightarrow$ pages 5

[107] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. End-to-End Versioning Support for Web Services. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08)*, pages 59–66, Washington, DC, USA, 2008. IEEE Computer Society. $\rightarrow$ pages 67

[108] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. Selecting Web Services Based on Past User Experiences. In *Proceedings of the 2009 Asia-Pacific Services Computing Conference (APSCC 2009)*, 2009. → pages 11, 99

[109] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'10)*, pages 369–376, Los Alamitos, CA, USA, 2010. IEEE Computer Society. → pages 4, 5

[110] Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Daios: Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13:72–80, 2009. → pages 4, 5, 53

[111] Philipp Leitner, Florian Rosenberg, Anton Michlmayr, Andreas Huber, and Schahram Dustdar. A Mediator-Based Approach to Resolving Interface Heterogeneity of Web Services. In *Emerging Web Services Technology Volume III*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 55–74, 2010. → pages

[112] Philipp Leitner, Branimir Wetzstein, Dimka Karastoyanova, Waldemar Hummer, Schahram Dustdar, and Frank Leymann. Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'10)*. Springer, 2010. → pages 4, 5

[113] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's Inside the Cloud? An Architectural Map of the Cloud Landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD'09)*, pages 23–31, Washington, DC, USA, 2009. IEEE Computer Society. → pages 57

[114] Leo Breiman. *Classification and Regression Trees*. Crc Pr Inc, 1984. → pages 17

[115] Tammo van Lessen, Jörg Nitzsche, and Frank Leymann. Formalising Message Exchange Patterns using BPEL Light. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08)*, pages 353–360, Washington, DC, USA, 2008. IEEE Computer Society. → pages 10

[116] Karl Lieberherr, Doug Orleans, and Johan Ovlinger. Aspect-Oriented Programming with Adaptive Methods. *Communications of the ACM*, 44:39–41, 2001. → pages 70

[117] Baoping Lin, Naijie Gu, and Qing Li. A Requester-Based Mediation Framework for Dynamic Invocation of Web Services. In *Proceedings of the International Conference on Services Computing (SCC 2006)*, pages 445–454, Washington, DC, USA, 2006. IEEE Computer Society. → pages 104

[118] Alexander Maedche and Steffen Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16:72–79, March 2001. → pages 66

[119] Michele Mazzucco, Isi Mitrani, Jennie Palmer, Mike Fisher, and Paul McKee. Web Service Hosting and Revenue Maximization. In *Proceedings of the Fifth European Conference on Web Services (ECOWS'07)*, pages 45–54, Washington, DC, USA, 2007. IEEE Computer Society. $\rightarrow$ pages 103

[120] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2), 2001. $\rightarrow$ pages 9, 56, 66, 105

[121] Daniel A. Menascé. QoS Issues in Web Services. *IEEE Internet Computing*, 6(6):72–75, 2002. $\rightarrow$ pages 2, 10

[122] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37:316–344, 2005. $\rightarrow$ pages 69

[123] Andreas Metzger and Klaus Pohl. S-Cube: Enabling the Next Generation of Software Services. In *Proceedings of the 5th International. Conference on Web Information Systems and Technologies (WEBIST 2008)*, Lecture Notes in Business Information Processing, pages 40–47, Berlin Heidelberg, 2009. Springer-Verlag. $\rightarrow$ pages 100, 110

[124] Andreas Metzger, Osama Sammodi, Klaus Pohl, and Mark Rzepka. Towards Pro-Active Adaptation With Confidence: Augmenting Service Monitoring With Online Testing. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*, pages 20–28, New York, NY, USA, 2010. ACM. $\rightarrow$ pages 99, 102

[125] Harald Meyer and Mathias Weske. Automated Service Composition Using Heuristic Search. In Schahram Dustdar, Jos Luiz Fiadeiro, and Amit P. Sheth, editors, *Proceedings of the 4th International Conference on Business Process Management (BPM'06)*, volume 4102 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2006. $\rightarrow$ pages 102

[126] Anton Michlmayr, Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Publish/Subscribe in the VRESCo SOA Runtime. In *Proceedings of the Second International Conference on Distributed Event-Based Systems (DEBS'08), Demo Track*, pages 317–320, New York, NY, USA, 2008. ACM. $\rightarrow$ pages 26

[127] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, pages 115–125, New York, NY, USA, 2008. ACM. $\rightarrow$ pages 23, 26, 36

[128] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC'09)*, pages 1–6, New York, NY, USA, 2009. ACM. $\rightarrow$ pages 11, 13, 25, 33, 99

[129] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Transactions on Services Computing*, 3:193–205, July 2010. → pages 1, 4, 13, 23, 53, 103

[130] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards Recovering the Broken SOA Triangle: a Software Engineering Perspective. In *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07)*, pages 22–28, New York, NY, USA, 2007. ACM. → pages 4, 13, 23, 53

[131] John Mingers. An Empirical Comparison of Pruning Methods for Decision Tree Induction. *Machine Learning*, 4(2):227–243, 1989. → pages 18

[132] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Non-Intrusive Monitoring and Service Adaptation for WS-BPEL. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*, pages 815–824, New York, NY, USA, 2008. ACM. → pages 99, 105

[133] Adina Mosincat and Walter Binder. Transparent Runtime Adaptability for BPEL Processes. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, pages 241–255, Berlin, Heidelberg, 2008. Springer-Verlag. → pages 104

[134] Adina Mosincat and Walter Binder. Automated Performance Maintenance for Service Compositions. In *11th IEEE International Symposium on Web Systems Evolution (WSE-2009)*, pages 131–140, 2009. → pages 3

[135] H.R. Motahari Nezhad, Boualem Benatallah, Fabio Casati, and F. Toumani. Web Services Interoperability Specifications. *IEEE Computer*, 39(5):24–32, May 2006. → pages 88

[136] Michael Mrissa, Chirine Ghedira, Djamal Benslimane, Zakaria Maamar, Florian Rosenberg, and Schahram Dustdar. A Context-Based Mediation Approach to Compose Semantic Web Services. *ACM Transactions on Internet Technology*, 8(1):4, 2007. → pages 105

[137] N. C. Narendra, Karthikeyan Ponnalagu, Jayatheerthan Krishnamurthy, and R. Ramkumar. Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming. In *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC'07)*, pages 546–557, Berlin, Heidelberg, 2007. Springer-Verlag. → pages 105

[138] Jörg Niemöller, Roman Levenshteyn, Eugen Freiter, Konstantinos Vandikas, Raphaël Quinet, and Ioannis Fikouras. Aspect Orientation for Composite Services in the Telecommunication Domain. In *Proceedings of the 7th International Joint Conference*

on Service-Oriented Computing (ICSOC'09), pages 19–33, Berlin, Heidelberg, 2009. Springer-Verlag. → pages 105

[139] Jörg Nitzsche, Tammo Van Lessen, Dimka Karastoyanova, and Frank Leymann. BPELlight. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, pages 214–229, Berlin, Heidelberg, 2007. Springer-Verlag. → pages 103

[140] Web Services Business Process Execution Language Version 2.0. Technical report, 2007. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel, Last Visited: 2011-07-19. → pages 7, 56

[141] Ernst Oberortner, Uwe Zdun, and Schahram Dustdar. Patterns for Measuring Performance-Related QoS Properties in Distributed Systems. In *Proceedings of the 17th Conference on Pattern Languages of Programs (PLoP)*, 2010. → pages 11, 79, 99

[142] Chun Ouyang, Marlon Dumas, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. From BPMN Process Models to BPEL Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 285–292, Washington, DC, USA, 2006. IEEE Computer Society. → pages 8

[143] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998. → pages 14

[144] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, November 2007. → pages 1, 7

[145] Papazoglou, Mike P. Service -Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)*. IEEE Computer Society, 2003. → pages 1, 59

[146] Cesare Pautasso. RESTful Web service composition with BPEL for REST. *Data & Knowledge Engineering*, 68(9):851–866, 2009. → pages 8, 58

[147] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web Services vs. "Big“ Web Services: Making the Right Architectural Decision. In *Proceeding of the 17th international conference on World Wide Web (WWW'08)*, pages 805–814, New York, NY, USA, 2008. ACM. → pages 58

[148] Chris Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36:46–52, October 2003. → pages 7

[149] Massimiliano Di Penta, Raffaele Esposito, Maria Luisa Villani, Roberto Codato, Massimiliano Colombo, and Elisabetta Di Nitto. WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services. In *Proceedings of the International Workshop on Service-Oriented Software Engineering (SOSE'06)*, pages 74–80, New York, NY, USA, 2006. ACM. → pages 104

[150] Harald Psaier and Schahram Dustdar. A Survey on Self-Healing Systems: Approaches and Systems. *Computing*, 91:43–73, January 2011. $\rightarrow$ pages 14

[151] J. Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, March 1986. $\rightarrow$ pages 16, 17

[152] J. Ross Quinlan. Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996. $\rightarrow$ pages 18, 29, 30

[153] J. Ross Quinlan and Ronald L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80:227–248, March 1989. $\rightarrow$ pages 18

[154] Nicholas Radcliffe and Patrick Surry. Formal Memetic Algorithms. *Evolutionary Computing*, 865:1–16, 1994. $\rightarrow$ pages 48

[155] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, 2007. $\rightarrow$ pages 8

[156] Pablo Rodriguez-Mier, Manuel Mucientes, and Manuel Lama. Automated Web Service Composition with a Heuristic-Based Search Algorithm. In *Proceedings of the 2011 International Conference on Web Services (ICWS'11)*. Springer, 2011. To appear. $\rightarrow$ pages 102

[157] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005. $\rightarrow$ pages 9

[158] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An End-to-End Approach for QoS-Aware Service Composition. In *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference (EDOC 2009)*, pages 151–160, 2009. $\rightarrow$ pages 103

[159] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Pedrak Celicovic, and Schahram Dustdar. Towards Composition as a Service – A Quality of Service Driven Approach. In *Proceedings of the 1st Workshop on Information and Software as Service (WISS)*, pages 1733–1740, Washington, DC, USA, 2009. IEEE Computer Society. $\rightarrow$ pages 103

[160] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, and Schahram Dustdar. Integrated Metadata Support for Web Service Runtimes. In *Proceedings of the Middleware for Web Services Workshop (MWS'08)*, pages 361–368, Washington, DC, USA, 2008. IEEE Computer Society. $\rightarrow$ pages 9, 56, 64

[161] Florian Rosenberg, Max Benjamin Müller, Philipp Leitner, Anton Michlmayr, Athman Bouguettaya, and Schahram Dustdar. Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions. In *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC'10)*, pages 97–104, Washington, DC, USA, 2010. IEEE Computer Society. $\rightarrow$ pages 102

[162] Florian Rosenberg, Christian Platzer, and Schahram Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 205–212, Washington, DC, USA, 2006. IEEE Computer Society. → pages 11, 33, 99

[163] Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad P. A. van Moorsel, and Fabio Casati. Automated SLA Monitoring for Web Services. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, 2002. → pages 2, 11, 99

[164] Mazeiar Salehie and Ladan Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autononmous and Adaptive Systems*, 4:1–42, May 2009. → pages 14, 23, 98

[165] Oleg Shilo. CS-Script – The C# Script Engine, 2009. http://www.csscript.net/, Last Visited: 2011-01-19. → pages 69

[166] Dharma Shukla and Bobby Schmidt. *Essential Windows Workflow Foundation*. Microsoft .NET Development Series, 2006. → pages 7

[167] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise Service Level Agreements. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society. → pages 12, 97

[168] Florian Skopik, Daniel Schall, and Schahram Dustdar. Trust-Based Adaptation in Complex Service-Oriented Systems. In *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 31–40, Washington, DC, USA, 2010. IEEE Computer Society. → pages 11

[169] Haitao Song, Yingyu Yin, and Shixiong Zheng. Dynamic Aspects Weaving in Service Composition. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)*, pages 1003–1008, Washington, DC, USA, 2006. IEEE Computer Society. → pages 105

[170] Martin Szomszor, Terry R. Payne, and Luc Moreau. Automated Syntactic Medation for Web Service Integration. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, 2006. → pages 105

[171] Vladimir Tosic, Wei Ma, Bernard Pagurek, and Babak Esfandiari. Web Service Offerings Infrastructure (WSOI) – A Management Infrastructure for XML Web Services. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'04)*, pages 817–830, 2004. → pages 97

[172] Vladimir Tosic, Bernard Pagurek, Kruti Patel, Babak Esfandiari, and Wei Ma. Management Applications of the Web Service Offerings Language (WSOL). *Information Systems*, 30(7):564–586, 2005. → pages 97

[173] UDDI.org. UDDI Technical White Paper, 2000. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, Last Visited: 2011-07-19. → pages 1

[174] Thomas Unger, Frank Leymann, and Thorsten Scheibler. Aggregation of Service Level Agreements in the Context of Business Processes. In *Proceedings of the 12th IEEE Enterprise Distributed Object Conference (EDOC 2008)*, pages 43–52. IEEE Computer Society, 2008. → pages 54, 101

[175] Wil Van Der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business Process Management: a Survey. In *Proceedings of the 2003 International Conference on Business Process Management (BPM'03)*, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag. → pages 1, 2

[176] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. → pages 100

[177] Daniel Villela, Prashant Pradhan, and Dan Rubenstein. Provisioning Servers in the Application Tier for E-Commerce Systems. *ACM Transactions on Internet Technology*, 7(1):7, 2007. → pages 103

[178] Markus Voelter, Michael Kircher, and Uwe Zdun. *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. Wiley Software Patterns Series, 2004. → pages 61

[179] Markus Voelter, Michael Kircher, Uwe Zdun, and Michael Englbrecht. Patterns for Asynchronous Invocations in Distributed Object Frameworks. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLOP 2003)*, pages 269–284, 2003. → pages 61

[180] World Wide Web Consortium (W3C). SOAP Message Transmission Optimization Mechanism, 2005. http://www.w3.org/TR/soap12-mtom/, Last Visited: 2011-07-19. → pages 87

[181] Ueli Wahli, Vedavyas Avula, Hannah Macleod, Mohamed Saeed, and Anders Vinther. *Business Process Management: Modeling Through Monitoring Using WebSphere V6.0.2 Products*. IBM Redbooks, 2007. → pages 100

[182] Robert J. Walker, Elisa L. A. Baniassad, and Gail C. Murphy. An Initial Assessment of Aspect-Oriented Programming. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pages 120–130. ACM, 1999. → pages 70

[183] Yao Wang and Julita Vassileva. A Review on Trust and Reputation for Web Service Selection. In *Proceedings of the 1st International Workshop on Trust and Reputation Management in Massively Distributed Computing Systems (TRAM'07)*, pages 25–, Washington, DC, USA, 2007. IEEE Computer Society. → pages 99

[184] Yao Wang and Julita Vassileva. Toward Trust and Reputation Based Web Service Selection: A Survey. *International Transactions on Systems Science and Applications (ITSSA)*, 3(2), 2007. → pages 11, 99

[185] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008. → pages 86, 90

[186] Branimir Wetzstein, Philipp Leitner, Florian Rosenberg, Ivona Brandic, Schahram Dustdar, and Frank Leymann. Monitoring and Analyzing Influential Factors of Business Process Performance. In *Proceedings of the 13th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'09)*, pages 118–127, Piscataway, NJ, USA, 2009. IEEE Press. → pages 11, 30, 33, 101

[187] Branimir Wetzstein, Philipp Leitner, Florian Rosenberg, Schahram Dustdar, and Frank Leymann. Identifying Influential Factors of Business Process Performance Using Dependency Analysis. *Enterprise Information Systems*, 4(3):1–8, July 2010. → pages 2, 33, 101

[188] Branimir Wetzstein, Steve Strauch, and Frank Leymann. Measuring Performance Metrics of WS-BPEL Service Compositions. In *Proceedings of the Fifth International Conference on Networking and Services (ICNS'09)*. IEEE Computer Society, April 2009. → pages 25

[189] Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4(2):65–85, 1994. → pages 46

[190] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988. → pages 14

[191] Stuart K. Williams, Steven A. Battle, and Javier Esplugas Cuadrado. Protocol Mediation for Adaptation in Semantic Web Services. In *Proceedings of the European Semantic Web Conference (ESWC 2006)*, pages 635–649, 2006. → pages 105

[192] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005. → pages 18, 29, 54

[193] D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5:214–259, 1992. → pages 39

[194] World Wide Web Consortium (W3C) . Web Services Choreography Description Language Version 1.0, 2004. http://www.w3.org/TR/ws-cdl-10/, Last Visited: 2011-07-19. → pages 8

[195] World Wide Web Consortium (W3C) . Web Application Description Language, 2009. http://www.w3.org/Submission/wadl/, Last Visited: 2011-07-19. → pages 61

[196] World Wide Web Consortium (W3C). SOAP Version 1.2 Part0: Primer, 2003. http://www.w3.org/TR/soap12-part0/, Last Visited: 2011-07-19. → pages 1

[197] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0 Part 0: Primer - W3C Candidate Recommendation 27 March 2006, 2006. http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/, Last Visited: 2011-07-19. → pages 1, 9

[198] World Wide Web Consortium (W3C). *Semantic Annotations for WSDL and XML Schema*, 2007. http://www.w3.org/TR/sawsdl/, Last Visited: 2011-07-19. → pages 56

[199] Jiuyun Xu and Stephan Reiff-Marganiec. Towards Heuristic Web Services Composition Using Immune Algorithm. In *Proceedings of the 2008 IEEE International Conference on Web Services (ICWS'08)*, pages 238–245, Washington, DC, USA, 2008. IEEE Computer Society. → pages 102

[200] Lei Xu and Brendan Jennings. A Cost-Minimizing Service Composition Selection Algorithm Supporting Time-Sensitive Discounts. In *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC'10)*, pages 402–408, Washington, DC, USA, 2010. IEEE Computer Society. → pages 40

[201] Jun Yan, Ryszard Kowalczyk, Jian Lin, Mohan B. Chhetri, Suk Keong Goh, and Jianying Zhang. Autonomous Service Level Agreement Negotiation for Service Composition Provision. *Future Generation Computer Systems*, 23:748–759, July 2007. → pages 13

[202] Stephen S. Yau, Nong Ye, Hessem S. Sarjoughian, Dazhi Huang, Auttawut Roontiva, Mustafa Baydogan, and Mohammed A. Muqsith. Toward Development of Adaptive Service-Based Software Systems. *IEEE Transactions on Services Computing*, 2:247–260, July 2009. → pages 98

[203] Stephen S. Yau, Yin Yin, and Ho G. An. An Adaptive Tradeoff Model for Service Performance and Security in Service-Based Systems. In *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*, pages 287–294, Washington, DC, USA, 2009. IEEE Computer Society. → pages 103

[204] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. *ACM Transactions on the Web*, 1, May 2007. → pages 102

[205] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. DISC: A Declarative Framework for Self-Healing Web Services Composition. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*, pages 25–33, Washington, DC, USA, 2010. IEEE Computer Society. → pages 98

[206] Stelios H. Zanakis and James R. Evans. Heuristic Öptimization": Why, When, and How to Use It. *Interfaces*, 11(5):84–91, 1981. → pages 45

[207] Uwe Zdun, Markus Voelter, and Michael Kircher. Design and Implementation of an Asynchronous Invocation Framework for Web Services. In *Proceedings of the European Conference on Web Services (ECOWS'03)*, pages 11–35. Springer-Verlag Berlin, Heidelberg, 2003. → pages 61

[208] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004. $\rightarrow$ pages 102

[209] Liangzhao Zeng, Hui Lei, and Henry Chang. Monitoring the QoS for Web Services. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, pages 132–144, Berlin, Heidelberg, 2007. Springer-Verlag. $\rightarrow$ pages 11, 99

[210] Liangzhao Zeng, Christoph Lingenfelder, Hui Lei, and Henry Chang. Event-Driven Quality of Service Prediction. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, pages 147–161, Berlin, Heidelberg, 2008. Springer-Verlag. $\rightarrow$ pages 101

[211] Yue Zhang, Mark Panahi, and Kwei-Jay Lin. Service Process Composition with QoS and Monitoring Agent Cost Parameters. In *Proceedings of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE)*, pages 311–316, Washington, DC, USA, 2008. IEEE Computer Society. $\rightarrow$ pages 103

[212] Walter Zucchini and Iain L. MacDonald. *Hidden Markov Models for Time Series: An Introduction Using R*. Chapman and Hall/CRC, 2009. $\rightarrow$ pages 109

APPENDIX A

Glossary

ANN	Artificial Neural Network
AOP	Aspect-Oriented Programming
API	Application Programming Interface
BAM	Business Activity Monitoring
BI	Business Intelligence
BPM	Business Process Management
BPMN	Business Process Modelling Notation
CEP	Complex Event Processing
CPWSI	Composite Pattern for Stubless Web Service Invocation
CPDL	Checkpoint Definition Language
DAIOS	Dynamic And Asynchronous Invocation of Services
DII	Dynamic Invocation Interface
DSL	Domain Specific Language
ESB	Enterprise Service Bus
GA	Genetic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedure
GUI	Graphical User Interface

HMM	Hidden Markov Model
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MA	Memetic Algorithm
MEP	Message Exchange Pattern
MTOM	SOAP Message Transmission Optimization Mechanism
PPM	Process Performance Metric
PREvent	Prediction and Prevention of SLA Violations Based on Events
QoS	Quality of Service
RCS	Restricted Candidate Set
RDF	Resource Description Framework
REST	Representational State Transfer
RPC	Remote Procedure Call
SAWSDL	Semantically Annotated WSDL
SEPL	Service Protocol Language
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SVM	Support Vector Machine
UDDI	Universal Description Discovery and Integration
VRESCo	Vienna Runtime Environment for Service-Oriented Computing
VMF	VRESCo Mapping Framework
VQL	VRESCo Query Language
WADL	Web Application Description Language
WS-BPEL	Web Service Business Process Execution Language

WS-CDL	Web Service Choreography Description Language
WSDL	Web Service Definition Language
WSLA	Web Service Level Agreement
WSFL	Web Service Flow Language
WSMO	Web Service Modeling Ontology
WWF	Windows Workflow Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

APPENDIX B

Case Study Implementation

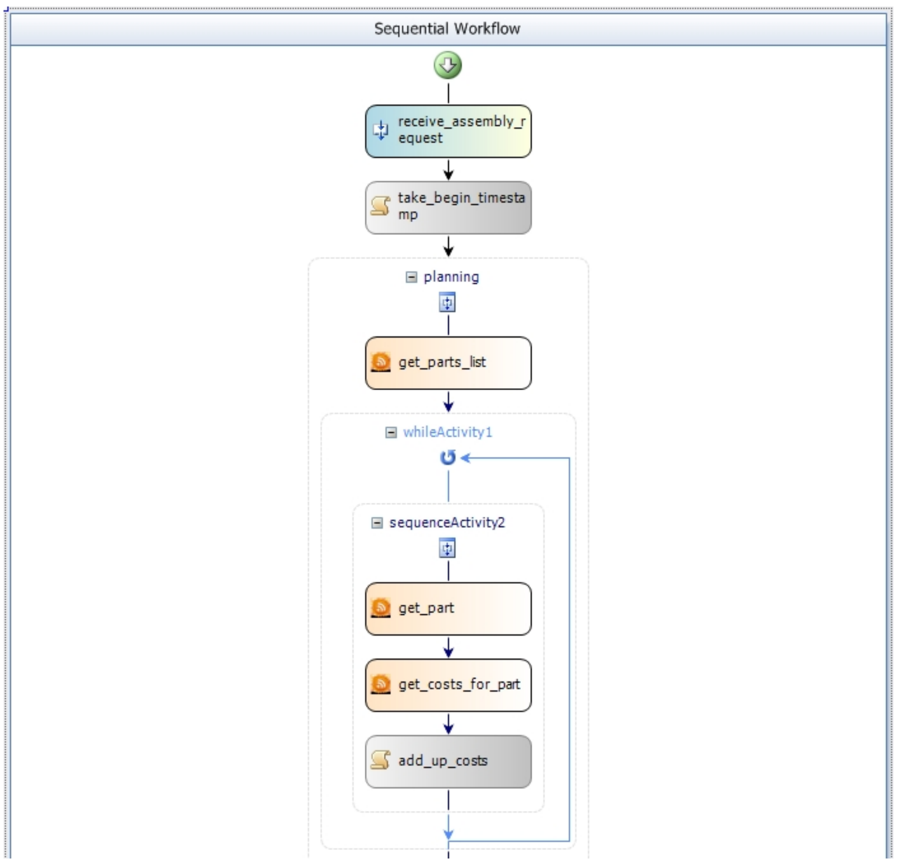


Figure B.1: Implementation of Evaluation Composition – Part 1

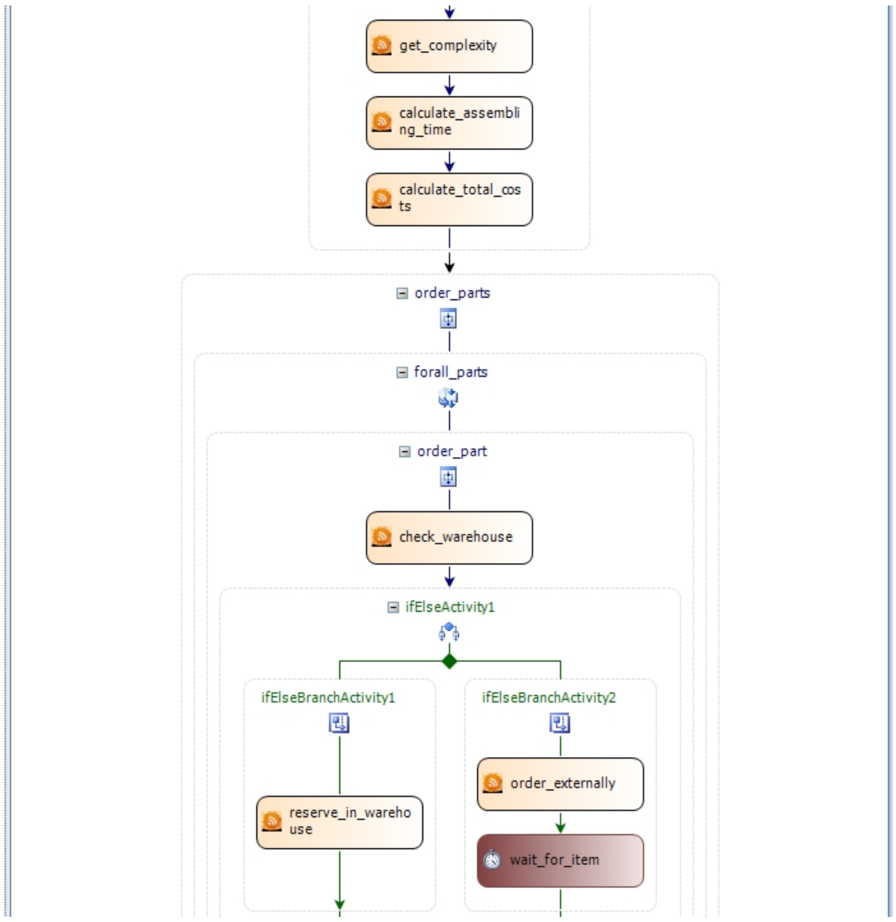


Figure B.2: Implementation of Evaluation Composition – Part 2

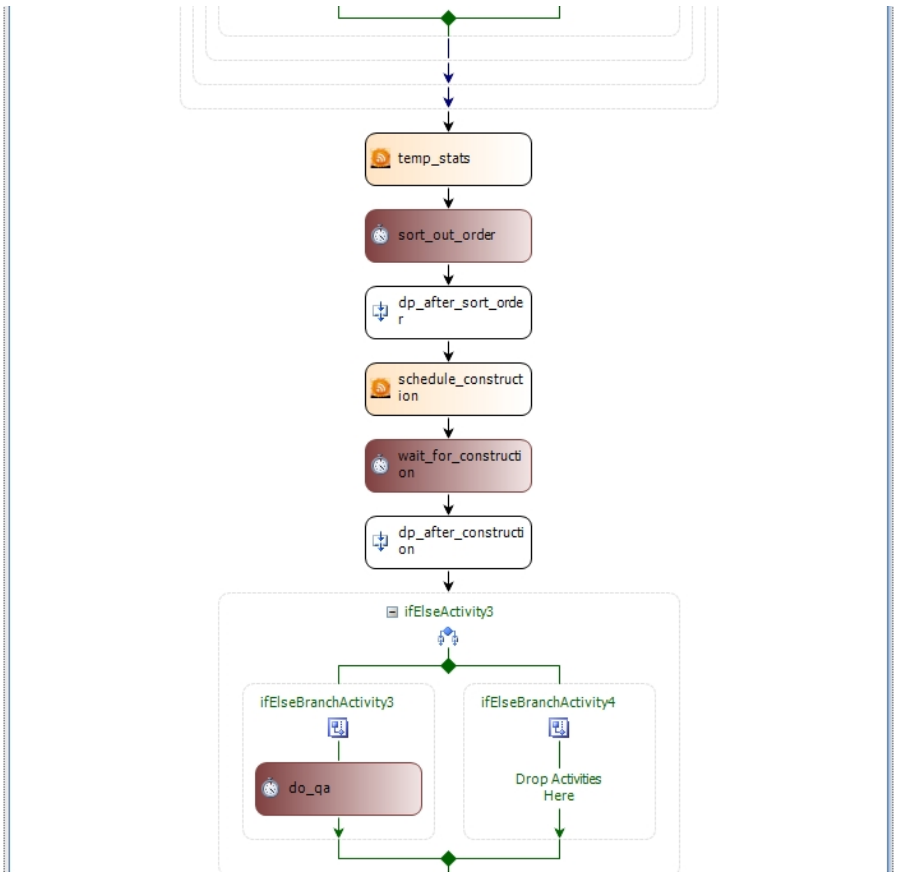


Figure B.3: Implementation of Evaluation Composition – Part 3

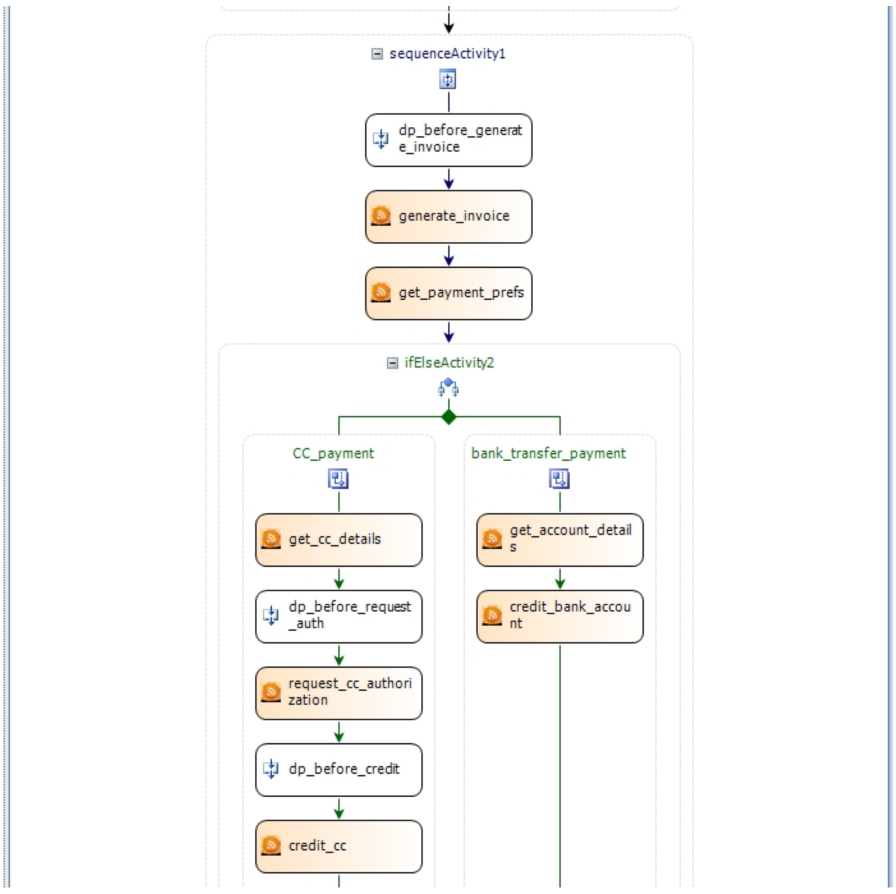


Figure B.4: Implementation of Evaluation Composition – Part 4

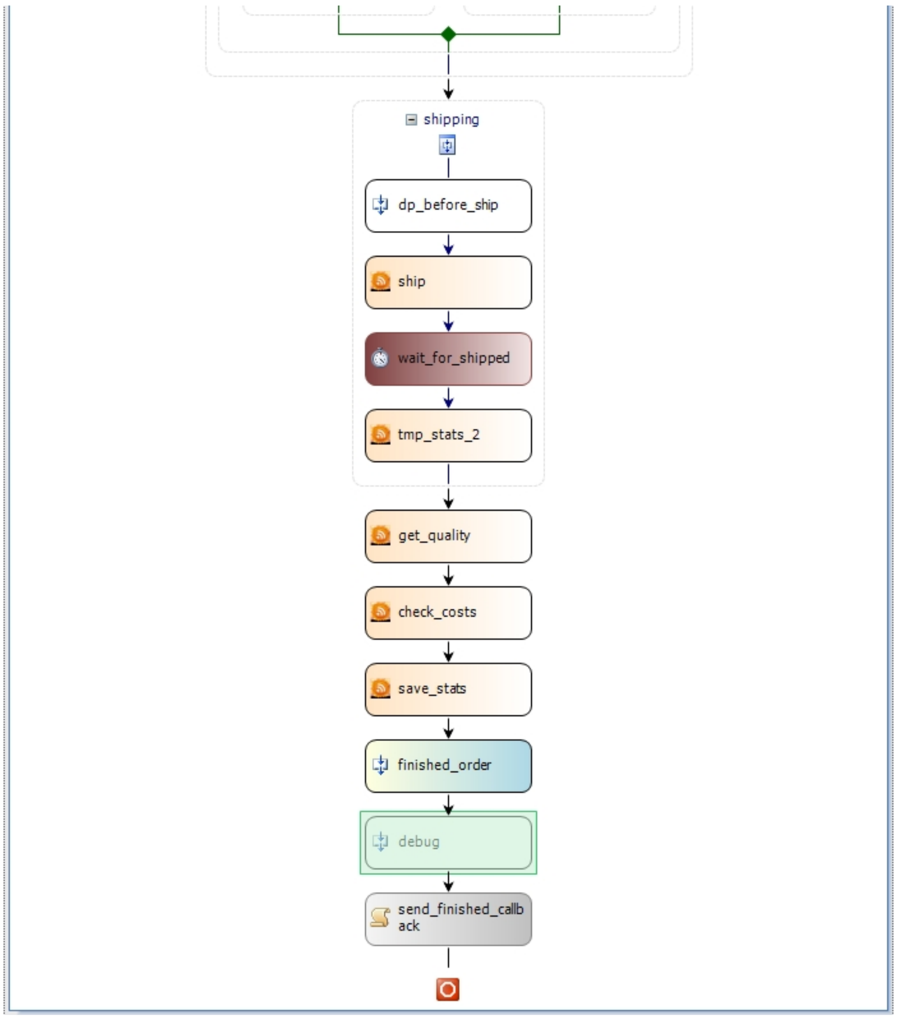


Figure B.5: Implementation of Evaluation Composition – Part 5

APPENDIX C

Curriculum Vitae

Vienna University of Technology
Distributed Systems Group
Argentinierstrasse 8
A-1040 Wien, Austria

Phone:	+43-1-58801-18413
Email:	leitner@infosys.tuwien.ac.at
Homepage:	http://www.infosys.tuwien.ac.at/staff/leitner/
Born:	Born on November 25, 1982 Citizen of Austria

Education

Ph.D. Candidate in Business Informatics, Vienna University of Technology	ongoing
M.S. Business Informatics, Vienna University of Technology	2007
B.S. Business Informatics, Vienna University of Technology	2005

Employment

University Assistant at Vienna University of Technology (pre-PhD faculty staff)	2008 - ongoing
Project Assistant at Vienna University of Technology	2007 - 2008
Software Engineer at Siemens Program and Systems Engineering Vienna	2005 - 2007
Independent Software Engineer, various smaller Web engineering projects	2003 - 2006
Teaching Assistant at Vienna University of Technology	2003 - 2007

Teaching Activities

Courses Taught

- ◇ Distributed Systems Lab (184.167 – LU 2.0 – Verteilte Systeme)
- ◇ Technologies for Distributed Systems (184.260 – VL 4.0)
- ◇ Advanced Internet Computing (184.269 – VL 2.0)
- ◇ Grid Computing (184.265 – VU 2.0)
- ◇ Internet Computing Lab Work (184.254 – PR 4.0 – Praktikum aus Internet Computing)

Master's Theses Supervised (as Assistant Supervisor)

- ◇ Petra Bierleutgeb: VRoxy – Enriching SOAs With Proxy-Based Dynamic Binding and Reporting Based on VRESCo
- ◇ Mathias Hess: Reducing SLA Violations of Composite Services Deployed to the Cloud
- ◇ Anton Korosec: Deploying a Web Service Runtime Environment into the Cloud
- ◇ Stefan Prennschütz-Schützenau: SOA Security Policy Validation and Authoring
- ◇ Klaus Zettler: Planung, Konzeption und Implementierung eines Virtual Campus im Rahmen von S-Cube
- ◇ Waldemar Hummer: Towards a Domain-Specific Language for Defining Intra-Service Protocols of Stateful Web Services

Program Commitee Memberships

- ◇ 4th Central-European Workshop on Services and their Composition (ZEUS) 2012
- ◇ 2nd Workshop on Provisioning and Management of Service Oriented Architecture and Cloud Computing (ProMASC) 2011, co-located with ICSOC'11

◇ 2nd Performance Assessment and Auditing in Service Computing Workshop (PAASC) 2011, co-located with ICSOC'11

◇ 6th Workshop on Emerging Web Services Technology (WEWST), co-located with ECOWS'11

◇ 4th International Workshop on Evolutionary Business Processes (EVL-BP), co-located with EDOC'11

◇ 1st Performance Assessment and Auditing in Service Computing Workshop (PAASC) 2010, co-located with ICSOC'10

◇ 5th Workshop on Emerging Web Services Technology (WEWST), co-located with ECOWS'10

◇ 3rd International Workshop on Dynamic and Declarative Business Processes (DDBP'10), co-located with EDOC'10

◇ 4th Workshop on Emerging Web Services Technology (WEWST), co-located with ECOWS'09

Publications

Journal Papers

◇ **P. Leitner**, W. Hummer and S. Dustdar: *Cost-Based Optimization of Service Compositions*, IEEE Transactions on Services Computing (TSC). To appear.

◇ W. Hummer, **P. Leitner** and S. Dustdar: *SEPL - A Domain-Specific Language and Execution Environment for Protocols of Stateful Web Services*, Distributed and Parallel Databases (DPD), Volume 29, Number 4, pp. 277-307. 2011

◇ B.Wetzstein, **P.Leitner**, F. Rosenberg, S. Dustdar and F.Leymann: *Identifying Influential Factors of Business Process Performance Using Dependency Analysis*, Enterprise Information Systems (EIS), Volume 5, Issue 1, 2011

◇ A.Michlmayr, F.Rosenberg, **P.Leitner** and S.Dustdar: *End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo*, IEEE Transactions on Services Computing (TSC), Volume 3, Number 3, 2010

◇ A.Michlmayr, F.Rosenberg, **P.Leitner** and S.Dustdar: *Selective Service Provenance in the VRESCo Runtime*, International Journal on Web Services Research (JWSR), vol 7, no. 2, pp. 65-86, 2010, doi: 10.4018/jwsr.2010040104

◇ **P.Leitner**, F.Rosenberg, S.Dustdar, Daios: *Efficient Dynamic Web Service Invocation*, IEEE Internet Computing, vol. 13, no. 3, pp. 72-80, May/June 2009, doi:10.1109/MIC.2009.57

Conference and Workshop Papers

◇ **P.Leitner**, B.Satzger, C.Inzinger, W.Hummer and S.Dustdar: *CloudScale – a Novel Middleware for Building Transparently Scaling Cloud Applications*, to appear at the 27th Symposium On Applied Computing (SAC), Track on Cloud Computing, 2012

◇ C.Inzinger, B.Satzger, W.Hummer, **P.Leitner** and S.Dustdar: *Non-Intrusive Policy Optimization for Dependable and Adaptive Service-Oriented Systems*, to appear at the 27th Symposium On Applied Computing (SAC), Track on Dependable and Adaptive Distributed Systems, 2012

◇ **P.Leitner**, F.M.Nardini, G.Tolomei, F.Silvestri and S.Dustdar: *Mining Lifecycle Event Logs for Enhancing Service-based Applications*, to appear at the 1st International Workshop on Adaptive Services for the Future Internet (WAS4FI), colocated with Service-Wave'11, 2011

◇ **P.Leitner**, W.Hummer, B.Satzger and S.Dustdar: *Stepwise and Asynchronous Runtime Optimization of Web Service Compositions*, presented at the 12th International Conference on Web Information System Engineering (WISE'11), short paper, 2011

- ◇ W.Hummer, B.Satzger, **P.Leitner** and S.Dustdar: *Distributed Continuous Queries Over Web Service Event Streams*, presented at the 7th International Conference on Next Generation Web Services Practices (NWeSP'11), 2011
- ◇ W.Hummer, **P.Leitner**, B.Satzger and S.Dustdar: *Dynamic Migration of Processing Elements for Optimized Query Execution in Event-based Systems*, presented at the International Symposium on Distributed Objects, Middleware, and Applications (DOA 2011), 2011
- ◇ B.Satzger, W.Hummer, **P.Leitner** and S.Dustdar: *Esc: Towards an Elastic Stream Computing Platform for the Cloud*, presented at the 4th IEEE International Conference on Cloud Computing (CLOUD'11), 2011
- ◇ W.Hummer, O.Raz, O.Shehory, **P.Leitner** and S.Dustdar: *Test Coverage of Data-Centric Dynamic Compositions in Service-Based Systems*, presented at 4th International Conference on Software Testing, Verification and Validation (ICST), 2011
- ◇ W.Hummer, **P.Leitner** and S.Dustdar: *WS-Aggregation: Distributed Aggregation of Web Services Data*, presented at 26th Symposium On Applied Computing (SAC), SOAP Track , 2011
- ◇ W.Hummer, **P.Leitner** and S.Dustdar: *A Step-By-Step Debugging Technique To Facilitate Mashup Development and Maintenance*, presented at 4th International Workshop on Web APIs and Services Mashups (Mashups'10) co-located with ECOWS 2010, December 1, 2010
- ◇ **P.Leitner**, B.Wetzstein, D.Karastoyanova, W.Hummer, S.Dustdar and F.Leymann: *Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution*, presented at International Conference on Service-Oriented Computing (ICSOC) , 2010
- ◇ **P.Leitner**, A.Michlmayr, F.Rosenberg and S.Dustdar: *Monitoring, Prediction and Prevention of SLA Violations in Composite Services*, presented at IEEE International Conference on Web Services (ICWS) Industry and Applications Track, 2010
- ◇ F.Rosenberg, M.Müller, **P.Leitner**, A.Michlmayr, A.Bouguettaya and S.Dustdar: *Metaheuristic Optimization of Large-Scale QoS-Aware Service Compositions*, presented at IEEE International Services Computing Conference (SCC), 2010
- ◇ **P.Leitner**, B.Wetzstein, F.Rosenberg, A.Michlmayr, S.Dustdar and F.Leymann: *Runtime Prediction of Service Level Agreement Violations for Composite Services*, presented at 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, co-located with ICSOC 2009, 2009
- ◇ A.Michlmayr, F.Rosenberg, **P.Leitner** and S.Dustdar: *Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection*, presented at 4th International Workshop on Middleware for Service Oriented Computing (MW4SOC), co-located with MIDDLEWARE'09, 2009

◇ **P.Leitner**, A.Michlmayr, F.Rosenberg and S.Dustdar: *Selecting Web Services Based on Past User Experiences*, presented at 2009 Asia-Pacific Services Computing Conference (APSCC 2009), 2009

◇ I.Brandic, D.Music, **P.Leitner**, S.Dustdar: *VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management*, presented at 6th International Workshop on Grid Economics and Business Models (Gencon), co-located with Euro-Par'09, 2009

◇ B.Wetzstein, **P.Leitner**, F.Rosenberg, I.Brandic, F.Leymann and S.Dustdar: *Monitoring and Analyzing Influential Factors of Business Process Performance*, presented at IEEE International Enterprise Computing Conference (EDOC), 2009

◇ F.Rosenberg, P.Celikovic, A.Michlmayr, **P.Leitner** and S.Dustdar: *An End-to-End Approach for QoS-Aware Service Composition*, presented at IEEE International Enterprise Computing Conference (EDOC), 2009

◇ **P.Leitner**, A.Michlmayr, F.Rosenberg, S.Dustdar: *End-to-End Versioning Support for Web Services*, presented at IEEE Services Computing Conference (SCC), 2008

◇ A.Michlmayr, F.Rosenberg, **P.Leitner** and S.Dustdar: *QoS-Aware Service Provenance in Web Service Runtimes*, presented at IEEE International Conference on Web Services (ICWS), 2009

◇ F.Rosenberg, **P.Leitner**, A.Michlmayr, P.Celikovic and S.Dustdar: *Towards Composition as a Service - A Quality of Service Driven Approach*, presented at 1st Workshop on Information and Software as Service (WISS), co-located with ICDE'09, 2009

◇ **P.Leitner**, A.Michlmayr and S.Dustdar: *Towards Flexible Interface Mediation for Dynamic Service Invocations*, presented at 3rd Workshop on Emerging Web Services Technology (WEWST), co-located with ECOWS'08, 2008

◇ F.Rosenberg, **P.Leitner**, A.Michlmayr and S.Dustdar: *Integrated Metadata Support for Web Service Runtimes*, presented at Middleware for Web Services Workshop (MWS), co-located with EDOC'08, 2008

◇ A.Michlmayr, F.Rosenberg, **P.Leitner** and S.Dustdar: *Advanced Event Processing and Notifications in Service Runtime Environments*, presented at IEEE Int'l Conference on Distributed Event-Based Systems (DEBS), 2008

◇ R.Marin, J.Vivero, **P.Leitner**, A.Neppach and M.Zach: *Securing the Madeira Network Management System*, presented at IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCom), 2007

◇ M.Leitner, **P.Leitner**, M.Zach, S.Collins and C.Fahy: *Fault Management based on peer-to-peer paradigms; A case study report from the CELTIC project Madeira*, presented at IFIP/IEEE International Symposium on Integrated Management (IM), 2007

◇ R.Marin, J.Vivero, H.Nguyen, J.Serrat, **P.Leitner**, M.Zach and C.Fahy: *A Distributed Policy Based Solution in a Fault Management Scenario*, presented at IEEE Global Communications Conference (GLOBECOM), 2006

Book Chapters

◇ W.Hummer, **P.Leitner**, A.Michlmayr, F.Rosenberg, S.Dustdar: *VRESCo – Vienna Runtime Environment for Service-oriented Computing*, appeared in Service Engineering. European Research Results, Springer, 2010

◇ A.Michlmayr, **P.Leitner**, F.Rosenberg and S.Dustdar: *Event Processing in Web Service Runtime Environments*, appeared in Handbook of Research on Advanced Distributed Event-Based Systems, Publish/Subscribe and Message Filtering Technologies, IGI Global, 2010

◇ **P.Leitner**, F.Rosenberg, A.Michlmayr, A.Huber and S.Dustdar: *A Mediator-Based Approach to Resolving Interface Heterogeneity of Web Services*, Post-Proceedings of the 3rd International Workshop on Emerging Web Service Technologies (WEWST'08), co-located with ECOWS'08, Dublin, Ireland, Birkhäuser, 2009

◇ C.Fahy, M.Ponce de Leon, S.van der Meer, R.Marin, J.Vivero, J.Serrat, N.Georgalas, **P.Leitner**, S.Collins and B.Baesjou: *Modelling Behaviour and Distribution for the Management of Next Generation Networks*, appeared in Whitestein Series in Software Agent Technologies and Autonomic Computing, Special issue on Advanced Autonomic Networking and Communication, Birkhäuser, 2008

Last updated: 19th October 2011

`http://www.infosys.tuwien.ac.at/staff/leitner/`