Managing Quality Properties of Web Service Compositions

Dissertation for the acquisition of the academic degree
Doktorin der Naturwissenschaften (Dr. rer. nat.)

Submitted to the
Faculty of Electrical Engineering and Computer Science
of the University of Kassel

By M.Sc. Diana-Elena Reichle

Kassel, September 2014

Advisors:
Prof. Dr. Kurt Geihs
Prof. Dr. Birgitta König-Ries

Additional Doctoral Committee Members:
Prof. Dr. Albert Zündorf
Prof. Dr. Arno Wacker

Date of Defense: 9 February 2015

Contents

Abstract

Web services from different partners can be combined to applications that realize a more complex business goal. Such applications built as Web service compositions define how interactions between Web services take place in order to implement the business logic. Web service compositions not only have to provide the desired functionality but also have to comply with certain Quality of Service (QoS) levels. Maximizing the users' satisfaction, also reflected as Quality of Experience (QoE), is a primary goal to be achieved in a Service-Oriented Architecture (SOA). Unfortunately, in a dynamic environment like SOA unforeseen situations might appear like services not being available or not responding in the desired time frame. In such situations, appropriate actions need to be triggered in order to avoid the violation of QoS and QoE constraints.

In this thesis, proper solutions are developed to manage Web services and Web service compositions with regard to QoS and QoE requirements. The Business Process Rules Language (BPRules) was developed to manage Web service compositions when undesired QoS or QoE values are detected. BPRules provides a rich set of management actions that may be triggered for controlling the service composition and for improving its quality behavior. Regarding the quality properties, BPRules allows to distinguish between the QoS values as they are promised by the service providers, QoE values that were assigned by end-users, the monitored QoS as measured by our BPR framework, and the predicted QoS and QoE values. BPRules facilitates the specification of certain user groups characterized by different context properties and allows triggering a personalized, context-aware service selection tailored for the specified user groups. In a service market where a multitude of services with the same functionality and different quality values are available, the right services need to be selected for realizing the service composition. We developed new and efficient heuristic algorithms that are applied to choose high quality services for the composition. BPRules offers the possibility to integrate multiple service selection algorithms. The selection algorithms are applicable also for non-linear objective functions and constraints. The BPR framework includes new approaches for context-aware service selection and quality property predictions. We consider the location information of users and services as context dimension for the prediction of response time and throughput. The BPR framework combines all new features and contributions to a comprehensive management solution. Furthermore, it facilitates flexible monitoring of QoS properties without having to modify the description of the service composition. We show how the different modules of the BPR framework work together in order to execute the management rules. We evaluate how our selection algorithms outperform a genetic algorithm from related research. The evaluation reveals how context data can be used for a personalized prediction of response time and throughput.

Zusammenfassung

Web-Services von verschiedenen Partnern können über das Internet zu Applikationen kombiniert werden, um ein komplexeres Geschäftsziel zu realisieren. Solche Applikationen, die als Web-Service-Kompositionen gestaltet sind, definieren, wie die Interaktionen zwischen Web-Services erfolgen, um eine Geschäftslogik zu implementieren. Web-Service-Kompositionen müssen nicht nur die gewünschte Funktionalität bereitstellen, sondern auch bestimmte Quality-of-Service-Level (QoS-Level) erfüllen. Ein primäres Ziel in einer Service-orientierten Architektur (SOA) ist die Maximierung der Benutzerzufriedenheit, welche durch die Quality-of-Experience (QoE) widergespiegelt wird. In einer dynamischen Umgebung wie SOA können unvorhersehbare Situationen auftreten, wie beispielsweise Services, die nicht erreichbar sind oder die nicht in der gewünschten Zeit antworten. In solchen Situationen müssen Aktionen ausgeführt werden, um die Verletzung von QoS- und QoE-Nebenbedingungen zu verhindern.

In dieser Arbeit werden Lösungen vorgestellt, um Web-Services und Web-Service-Kompositionen bezüglich QoS- und QoE-Anforderungen zu verwalten. Die Business-Process-Rules-Sprache (BPRules) wurde entwickelt, um Web-Service-Kompositionen zu managen, wenn unerwünschte QoS- und QoE-Werte festgestellt werden. BPRules liefert eine umfangreiche Menge an Aktionen, welche aufgerufen werden können, um die Service-Komposition zu steuern und um ihr Qualitätsverhalten zu verbessern. BPRules kann zwischen folgenden Qualitätseigenschaften unterscheiden: QoS-Werte, die vom Anbieter versprochen wurden, QoE-Werte, die von End-Benutzern angegeben wurden, QoS-Werte, welche vom BPR-Framework gemessen wurden, und prädizierte QoS- und QoE-Werte. BPRules ermöglicht die Spezifikation von Benutzergruppen, die durch bestimmte Kontexteigenschaften charakterisiert sind, und es unterstützt eine personalisierte, kontextbewusste Service-Selektion, die auf die spezifizierten Benutzergruppen angepasst ist. In einem Service-Markt werden eine Vielzahl von Services mit derselben Funktionalität und unterschiedlichen Qualitätseigenschaften angeboten, und die richtigen Services müssen selektiert werden, um die Service-Komposition zu realisieren. Wir haben neue und effiziente heuristische Algorithmen entwickelt, um Services mit guten Qualitätseigenschaften für die Komposition auszuwählen. BPRules ermöglicht die Integration mehrerer Service-Selektions-Algorithmen. Die Selektions-Algorithmen sind auch für nicht-lineare Zielfunktionen und Nebenbedingungen anwendbar. Das BPR-Framework enthält neue Ansätze für eine kontextbewusste Service-Selektion und Prädiktion von Qualitätseigenschaften. Wir berücksichtigen die Kontextdimension Ort, sowohl des Benutzers als auch des Services, bei der Prädiktion von Antwortzeit und Durchsatz. Das BPR-Framework kombiniert alle neuen Merkmale und Beiträge zu einer umfassenden Managementlösung. Es ermöglicht eine flexible Überwachung von QoS-Eigenschaften, ohne dass die Beschreibung der Service-Komposition manuell geändert werden muss. Wir beschreiben, wie die verschiedenen Module des BPR-

Frameworks interagieren, um die Managementregeln auszuführen. Wir evaluieren, wie unsere Selektions-Algorithmen einen genetischen Algorithmus aus einer verwandten Arbeit übertreffen. In der Evaluierung wird zudem gezeigt, wie Kontextdaten bei einer personalisierten Prädiktion von Antwortzeit und Durchsatz berücksichtigt werden können.

Acknowledgements

During my work at the University of Kassel, I had the chance to write this dissertation and I enjoyed working in interesting projects like ADDOaction and VENUS. It was a nice experience to be confronted with many interesting topics, but also to meet nice people and to get to know beautiful places while traveling to conferences and workshops. I would like to thank several people who supported me in writing this dissertation and were accompanying my research work.

First, I would like to thank my doctoral advisor *Prof. Kurt Geihs* for giving me the chance to work in his research group. Without offering me this opportunity, my life would be different now. I am very grateful to *Kurt* for guiding me in writing this dissertation, but also for his support in writing research articles and working in research projects. I further like to thank *Prof. Birgitta König-Ries* for reviewing my thesis.

Special thanks go to *Harun Baraki* and *Steffen Bleul*, who discussed with me my research topic and gave valuable inputs for my dissertation. *Harun* was an excellent student and further became my colleague and a friend. It was nice working with *Harun* and *Steffen*.

From *Thomas Weise* I learnt what dedication and devotement to work means. *Michael Zapf* contributed with explaining rules of English grammar but also by maintaining a good atmosphere in our group and keeping us always up to date with information from Wikipedia.

Together with my colleagues *Christoph Evers*, *Andreas Witsch* and *Stefan Niemczyk*, we were responsible for the *Adaptation* project in VENUS. We *adapted* to the new requirements of VENUS and researched in an interdisciplinary team. I would like to thank all the VENUS colleagues for their collaboration and also for providing useful insights from their disciplines. I especially enjoyed working with *Olga Kieselmann*. With the knowledge from VENUS, I will definitely design only *social* applications in future.

From the very beginning, I could observe at the boys from our group a deep passion for soccer and robots. My colleagues *Philipp Baer*, *Hendrik Skubch*, *Michael Wagner*, *Daniel Saur*, *Dominik Kirchner*, *Tareq Razaul Haque* and *Stephan Opfer* introduced me to interesting robotic topics. I also want to mention *'Titu' Khan*, *Thang Nguyen*, *Thomas Kleppe*, *Heidemarie Bleckwenn* and *Iris Roßbach* who contributed to a nice atmosphere in our group.

Last but not least, I especially want to thank my husband *Roland* and my father *Tiberiu* for supporting and encouraging me in writing this thesis. Of course, I want to thank my little daughter *Sandra* for distracting me from this dissertation, since with her I spend the most beautiful time.

1 Introduction

1.1 Topic Overview

Applications from different enterprises need to collaborate in order to accomplish business goals. It is required that applications running on different platforms are able to communicate and exchange their data. The Internet may be used as the underlying infrastructure for these applications. Therefore, distributed and heterogeneous applications have to overcome interoperability challenges and deal with a dynamic environment as the Internet. The Web service technology has emerged for overcoming these challenges. A Web service is a software application that is available in a network and can be accessed via its public interface [79]. Web services gained much of interest through their reliance on XML-based standards (e.g. SOAP, WSDL) and their accessibility over the Internet. The service provider offers the Web service and publishes its interface in a service registry. The service consumer looks up for a service in the registry, invokes it and uses its functionality [52].

Web services from different partners may cooperate and form a business process that delivers a higher business value to its clients. To implement a more complex business logic, several Web services may be necessary. In service-oriented computing, business processes are commonly realized as Web service compositions. A business process implies performing a set of activities that together achieve a business objective and produce a business value for the customers or partners [80]. Examples of business processes are: processing customer orders, delivering goods to the customer, or offering services (e.g. renting a car, booking a room at a hotel, approving a credit at a bank). The Web Services Business Process Execution Language, shortly WS-BPEL [16], has emerged as the standard technology for executing such a process. WS-BPEL permits specifying Web service compositions by defining their control structure and service interactions.

Services need to offer and maintain adequate quality, also known as Quality of Service (QoS) (e.g. response time, availability, reliability), to accomplish clients' expectations. Services are supposed to be reliable and to respond to client requests in reasonable time. Solely Web services that ensure the fulfillment of convenient QoS levels may be integrated into applications or business processes in a reliable way. In a dynamic environment as the Internet, guaranteeing of adequate QoS levels is quite a challenge. Therefore, QoS becomes a major concern for the success of business processes and received much attention from the research community.

While service providers measure and advertise their Web services at certain QoS levels, service users have different perceptions on the Web services and they are not necessarily thinking of services in terms of technical parameters. A human user notices whether the service is responding in a reasonable time frame, but he is also interested whether

the service corresponds to his personal requirements regarding, for example, usability and functionality [38, 91]. The degree reflecting the user satisfaction with the service is also known as *Quality of Experience (QoE)*. The QoE of a Web service can be assigned as a mean opinion score (MOS) directly by human users or it can be computed by an algorithm on behalf of the user [54]. QoE is a complex dimension which is depending on several factors such as user expectations and user psychology, context, social and economic factors, etc. [38].

Throughout the thesis, we will refer to QoS and QoE also as Quality Properties (QP). We will use the terms of service process, business process, process and service composition interchangeably and refer to a Web service composition.

1.2 Motivation and Problem Statement

As long as Web services accomplish their tasks and fulfill the promised QoS, these can be used for building more complex business processes. Unfortunately, undesired situations, like a network failure, a service becoming unavailable or not responding in the desired time frame, may happen in a dynamic environment like SOA. Services may become deprecated and newer services with a better QoS performance may take their place. The malfunction of one single service may cause the failure of the entire process and cause clients' dissatisfaction. Therefore, a service composition should be adaptable when service performance is decreasing and customer requirements change. Binding to new services should not alter the process description. Services are invoked by many users, which may have different experiences with a service. Thus, new service users may profit from experiences of other service clients that invoked the service in the past.

In order to avoid undesirable situations, a business process needs to be continuously monitored and managed regarding its QoS and QoE. In case of undesired situations, immediate management actions need to be undertaken to improve the behavior of the business process. Flexible QoS and QoE management remains an important and open issue in service management.

The goal of this thesis is to offer **a comprehensive solution for QoS and QoE management of Web service compositions**. A unified solution is needed that traces the QoS of the business process on the one side, and takes into consideration the users experiences on the other side, whether users are satisfied when interacting with the services and the process.

In order to provide a successful management support, we identified five research areas that have to be explored thoroughly:

- monitoring the QoS dimensions of service compositions
- specification of QP requirements and of appropriate management actions
- selection of services
- prediction of quality properties
- providing a quality management solution

Nevertheless, the aspects from the four research areas have to be combined into a unified solution with adequate tool support. Usually, a person, also referred as the business analyst, is responsible for the development and management of the business processes [80]. Another goal of this work is to provide the business analyst with adequate tool support for QoS and QoE management.

In the following, we discuss the challenges posed by QP management related to the mentioned topics:

- **Flexible Monitoring**

  Inadequate quality levels of services and service compositions need to be detected at the right moment, before unacceptable quality values are reached. In a continuously changing environment like SOA, new quality dimensions may be introduced that have to be monitored. These new quality dimensions have to be easily added into the monitoring process, without many efforts of the business analyst. The quality values of a service composition rely on the quality values of the services and of the other activities that are part of the composition. A **flexible monitoring** at runtime is required which indicates the parts of the service process that cause problems.

- **QoS and QoE Management**

  A service composition has to fulfill certain QoS and QoE requirements in order to be trusted by its clients. While the service provider is supposed to provide its services at the promised QoS levels, the service users demand that the services behave as they expect and achieve their tasks properly. These quality requirements need to be specified by the business analyst in order to be able to react upon undesired services behavior. The analyst is not necessarily a software developer and may not have any programming skills. His specifications need to be interpreted dynamically at runtime. It is necessary that the business analyst may choose from a set of predefined actions that may be applied on the service process. When the QoS requirements are violated, immediate actions should be triggered automatically to improve the process behavior.

- **Service Selection**

  We assume that in future service markets there are multiple services offered by different service providers having the same functionality but with different service levels. Thus, quality properties make the difference between services, and choosing the right service is an important issue for the success of the business process. To ensure that a service composition fulfills certain quality requirements, appropriate services need to be selected for the composition so that the desired QoS and QoE requirements of the composition are met. Choosing an optimal collection of services for the composition is known to be an NP-hard problem [22]. Thus, algorithms for the selection of services with a reasonable computation effort need to be developed.

- **QoS and QoE Predictions**

  The selection of services may be improved by considering predicted values of QoS and QoE for services. Considering past experiences of clients which have

already invoked the services may contribute in selecting future high quality services for the composition. It is desirable to analyze whether the consideration of context data would improve the prediction quality. Users with similar context data can be regarded as a user group. Another challenge is to build a customized service process that is suited to a specific user group.

- **A Quality Management Solution**

  The mentioned aspects have to be combined into a unified solution that supports quality management including QoS monitoring, service selection, the collection of QP feedbacks, and QP predictions. The solution needs to be sustained by adequate tools that assist the business analyst in his work to manage the service composition properly.

1.3 Contribution

In this thesis, we propose the BPRules language and the BPR framework that offer comprehensive support for QoS and QoE management for Web service compositions (see Figure 1.1). The contribution of this thesis resides in providing support for the four research challenges monitoring, QoS and QoE management, Web service selection and QP predictions. We contribute with novel algorithms, new features and tool support which build a comprehensive solution for QoS and QoE management. We developed the BPR framework which facilitates management actions on the service composition by means of executing rules.

- **Flexible Monitoring**

  Within the BPR framework, BPEL processes are continuously monitored during runtime. The quality values of the service composition or of composition parts are computed out of the quality values of the building blocks, represented by atomic services or by other BPEL activities. For the monitoring task, we focused on the flexibility aspect. Thus, we keep the monitoring artefacts apart from the process description file, and the necessary sensors for monitoring are deployed automatically [47]. The aggregation is performed by our *measurement algorithm* which can be triggered in an *online* or *offline* mode. The *online* mode refers to aggregation of running process instances, and the *offline* mode targets the process instances that already finished their execution. We also implemented the estimation algorithm similar to how it was proposed by [39]. The estimation algorithm is used to make an estimation of QoS and QoE of the process necessary for the selection of services.

- **QoS and QoE Management**

  For a successful QoS and QoE management of Web services and Web service compositions, we designed and developed a novel language, namely the **BPRules (Business Process Rules) language** [50] [48]. As its name suggests, BPRules is a rule-based language with special features that we envision as mandatory for the QoS and QoE management of services. The business analyst may specify rules with BPRules and can state what should happen if certain QoS requirements are

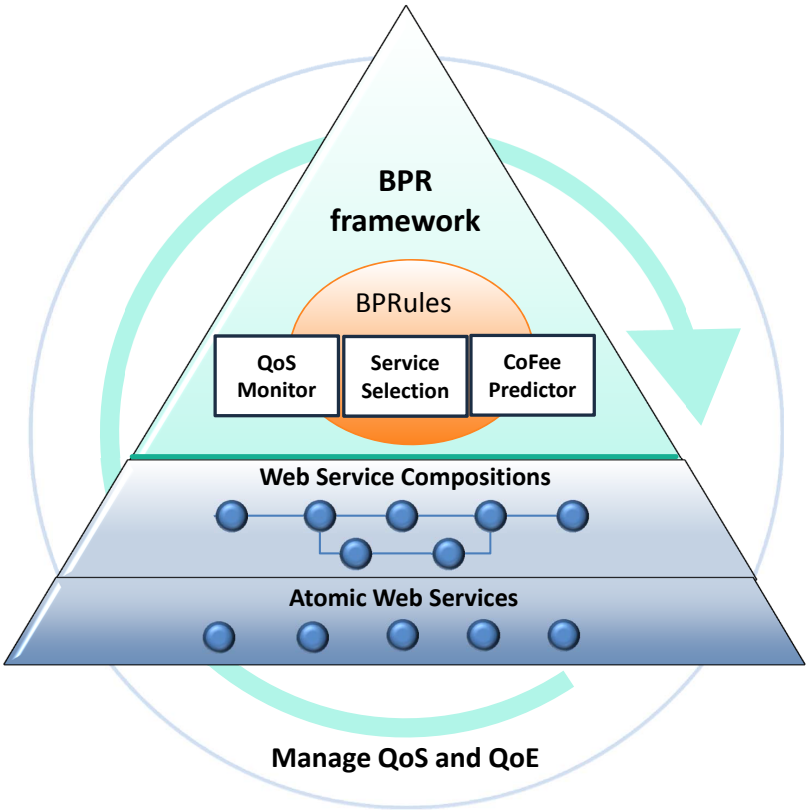

Figure 1.1: Architecture Overview

violated. He is not required to have any programming skills as BPRules relies on XML and it is simple to use. BPRules goes beyond the state of the art by offering novel features like *instance set handling* and a *dynamic rule set change*, but also by improving the flexibility of already established features, as e.g. the *selection of services*, *QP data retreival* for different periods of time and *section control*. With regard to these features BPRules provides more sophisticated means to specify management rules compared to already existing approaches. With BPRules we may distinguish between monitored QoS and assigned QoE, QP as advertised by the service provider and QP predictions. BPRules has its focus on the service selection action, and the business analyst may specify more detailed requirements for the selection. He may choose among multiple service selection algorithms and may specify the QP constraints to be fulfilled and an objective function to be optimized. The service selection may be triggered to create personalized service processes for specified user groups which are characterized by certain context properties.

- **Service Selection**

  Selecting adequate services to realize the desired functionality is a crucial task for the success of the service or process. We propose novel and very efficient **selection algorithms** [49], $OPTIM\_S$, $OPTIM\_PRO$, and $OPTIM\_HWeight$, that receive as input the quality requirements for the service composition and

return the services to be selected for the composition. Being heuristic algorithms, the $OTPIM\_PRO$ and $OPTIM\_HWeight$ algorithms offer near-optimal solutions in affordable computation time. Applying the $OPTIM\_S$ algorithm, we can even choose between computation time and the quality of the solution by only changing its parameters. The $OPTIM\_HWeight$ algorithm is based on gradient ascent and the $OPTIM\_PRO$ algorithm uses priority factors to process the BPEL nodes.

- **QoS and QoE Predictions**

  We developed CoFee (the Feedback Collector and Predictor), a module that collects feedbacks from service users and uses the feedbacks to make QP predictions [24]. We show exemplarily by response time and throughput how context data of users and services can be used in making predictions. We discuss how the QoE may be influenced by context dimensions (e.g. age, profession) of the users. With BPRules, we may create customized processes for a certain user group consisting of users with similar context data.

- **A Unified QoS and QoE Management Solution**

  We developed the BPR framework which provides support for the QP management of Web services and Web services compositions. The rules specified with BPRules are deployed and interpreted by the BPR framework. The BPR framework is built of several modules which offer support for monitoring, execution of rules, service selection and the prediction of QPs. Thus, the BPR framework combines all the artefacts necessary for a successful QP management and represents a unified solution.

1.4 Structure of the Thesis

This thesis is composed of eleven chapters starting with the presentation of the foundations in Chapter 2 followed by an overview of the solution in Chapter 3. A comparison with related work studies is presented in Chapter 4, and a detailed description of the solution can be found from Chapter 5 to Chapter 9. We evaluate our approach in Chapter 10 and summarize the main results of this work in Chapter 11. In the following, we give a short overview of each of the chapters:

- **Chapter 2** introduces some foundations that represent the basis for this thesis. We give some insights about Web services and Web service compositions, the Service-Oriented Architecture, QoS and QoE properties, workflows, and business processes.
- **Chapter 3** gives a short overview of the solution including the main modules of the BPR framework and their functionalities.
- **Chapter 4** contains other related studies that we compare with this work. We regard the four mentioned research domains: monitoring, quality property management, service selection and quality property prediction.
- **Chapter 5** presents the BPRules language, our solution to QoS and QoE management for Web service compositions. We show how rules specified with the BPRules language can be applied in order to manage a service composition.

- **Chapter 6** shows how the QoS properties of the services are monitored and how the quality values of the composition can be aggregated with the *measurement algorithm* and the *estimation algorithm*.
- **Chapter 7** presents the three algorithms $OPTIM\_S$, $OPTIM\_HWeight$ and $OPTIM\_PRO$ that we propose to apply for the selection of services for the composition.
- **Chapter 8** describes how the response time and throughput values of the atomic services are predicted using multiple linear regression. We also discuss how QoE can be predicted. It is shown how these predictions are further used within the selection of services. The context data of services and users may be used to create context-aware processes.
- **Chapter 9** presents the BPR framework. We give some implementation details about the BPR framework and show how the modules of the BPR framework work together in order to achieve the quality management.
- **Chapter 10** shows how we evaluated our approach. It includes a comparison of our selection algorithms with a genetic algorithm from related work [39]. For the prediction of response time and throughput we use a real dataset from [112] [17] and compare the prediction quality and computation time of multiple regression functions.
- **Chapter 11** summarizes the main results of this work and discusses future work.

2 Foundations

In this chapter, we describe selected foundations that represent the basics for this thesis. We give some insights about the Service-Oriented Architecture and the Web service technology. Concepts such as workflows, business processes and the definitions of QoS and QoE are introduced. We present the BPEL language that we used for building service compositions.

2.1 Service-Oriented Architecture

Nowadays, enterprises, as part of a global marketplace, have to be competitive and adaptable to new business requirements. Applications of different enterprises need to cooperate independently of their locations and platforms. When systems become more complex, flexibility is an important goal to be reached. Complex and distributed systems and applications need to be developed and maintained, while it is desirable to minimize the development and administrative costs. SOA is the emerging paradigm that addresses these challenges.

Services [87, 74, 62] are at the core of a Service-Oriented Architecture (SOA). A service is a software component or program that may solve a computational problem or a specific business task and is available in a network. Services are loosely coupled, they offer support for overcoming interoperability issues and distributivity challenges that come with an increased collaboration demand between organizations. They provide solutions for business integration, where partners may cooperate by using the Internet as the underlying infrastructure. A service has a service description that is accessible for the service consumers. This service description contains all the necessary artefacts (like service methods, bindings, communication protocols) that the consumer needs to know in order to be able to call the service. The service description has to be independent from the platform of the service.

The *Service-Oriented Architecture (SOA)* [87, 79, 74, 62, 114] is a paradigm where services represent the basic components of the software architecture. A SOA is an architectural style that integrates services that may be distributed on different servers. Respecting the SOA guidelines and principles, flexible and cost effective business processes can be created by employing services [79, 87]. Functionalities of applications can be encapsulated into services and may be offered to the interested parties. Therefore, the complexity of integrating applications is reduced and heterogeneity is overcome. SOA is defined in [87] as follows:

Definition 1 *"SOA is an architectural style for building enterprise solutions based on services. More specifically, SOA is concerned with the independent construction of business-aligned services that can be combined into meaningful, higher-level business processes and solutions within the context of the enterprise."*

The OASIS reference model [15] describes SOA as *"a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains."* Thus, the services may be implemented by different organizations. Business solutions are created more flexible by building new applications or by recomposing applications by reusing existing services. SOA is a new way of building distributed applications, independent of the specific technologies. A SOA is commonly implemented using Web services, but also other technologies may be employed such as J2EE or CORBA. Web services are preferred as they conform to the SOA principles of loose coupling, platform independence, interoperability and distributivity.

Common functionalities within an enterprise can be encapsulated into services. In this way, SOA crosses boundaries and easily integrates applications from different departments of an enterprise. SOA also offers support for the collaboration between an enterprise and its partners. Suppliers or trading partners of an enterprise also participate within a business process and need to be included in the process as well. *External services* provided by other organizations or partners, which are external to the enterprise, allow new functionalities outside the enterprise to be easily integrated into the own business processes [87, 79]. Services can be composed as needed to accomplish a business task. The integration of external services into a business process is one of our goals in this thesis.

2.1.1 SOA Roles

SOA defines the roles that occur in the communication with the Web services. The main participant roles in a SOA are: the *service provider*, the *service consumer*, the *service registry* [79, 74] and the *business analyst* [80]. The service provider offers its services to potential clients, who may call the services and use their functionalities. The roles are associated with some operations like publishing the services, searching for them and binding to the services.

- **Service Provider**
  The service provider is an organization that offers the service and makes it available in a network. The provider owns the service and is responsible for the platform where the service is deployed. Other tasks of the provider are the specification of the service description and their maintenance so that the service is available and keeps certain QoS levels. The service description contains three types of information about the *business*, including the provider of the service or its implementer, information about the *service* and *technical* information regarding the methods and other technical details. The provider *registers* the service in the service registry and publishes its service description so that potential clients may find it.
- **Service Registry**
  The service registry is a directory that is used by service consumers to search for

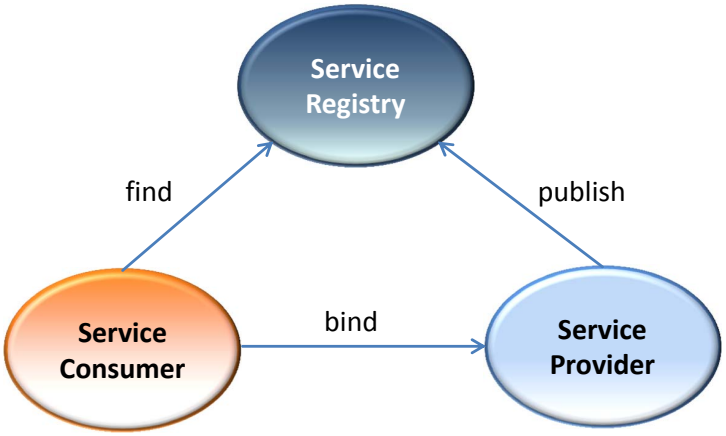

Figure 2.1: SOA Roles [80]

services. A service registry is similar to the yellow pages but for Web services. The registry offers the service provider the possibility to *register* services and to *publish* service descriptions. The service consumer uses the registry to *search* for services. There may be multiple service registries that can be searched for services. The service registry may contain services that provide the same functionality, owned by the same or by different service providers. The service registry contributes to low coupling between the services. A well-known service registry is the *Universal Description, Discovery and Integration (UDDI)* [1].

- **Service Consumer**
  A service consumer may be an application, such as a Web portal, or it may be a business process that needs to integrate the service to use its functionality. The service consumer *searches* for services in the service registry by submitting a query that contains the criteria for the searched service. A set of services that are discovered and correspond to the requesting query are returned to the customer. The appropriate service is chosen (selected) from the set of discovered services. The consumer uses the service description to *bind* to the service and call its operations. In our case, the service selection is mediated by our BPR framework. Further details about our selection approach is provided in Chapter 7.

- **Business Analyst**
  The business analyst is responsible to "*develop, manage and monitor business processes*"[80]. The business analyst facilitates the communication between the business side and the IT department of an enterprise. Thus, he may be seen as "*the person in the middle*" [90]. Typically, the business analyst is not required to have any technical skills. He has to ensure that the business requirements are met. One of our goals in this thesis is to offer the business analyst adequate tool support for managing the business process properly.

The communication between the service provider, the service consumer and the service registry is known as the *publish-find-bind* process and is represented in Figure 2.1.

2.1.2 SOA Layers

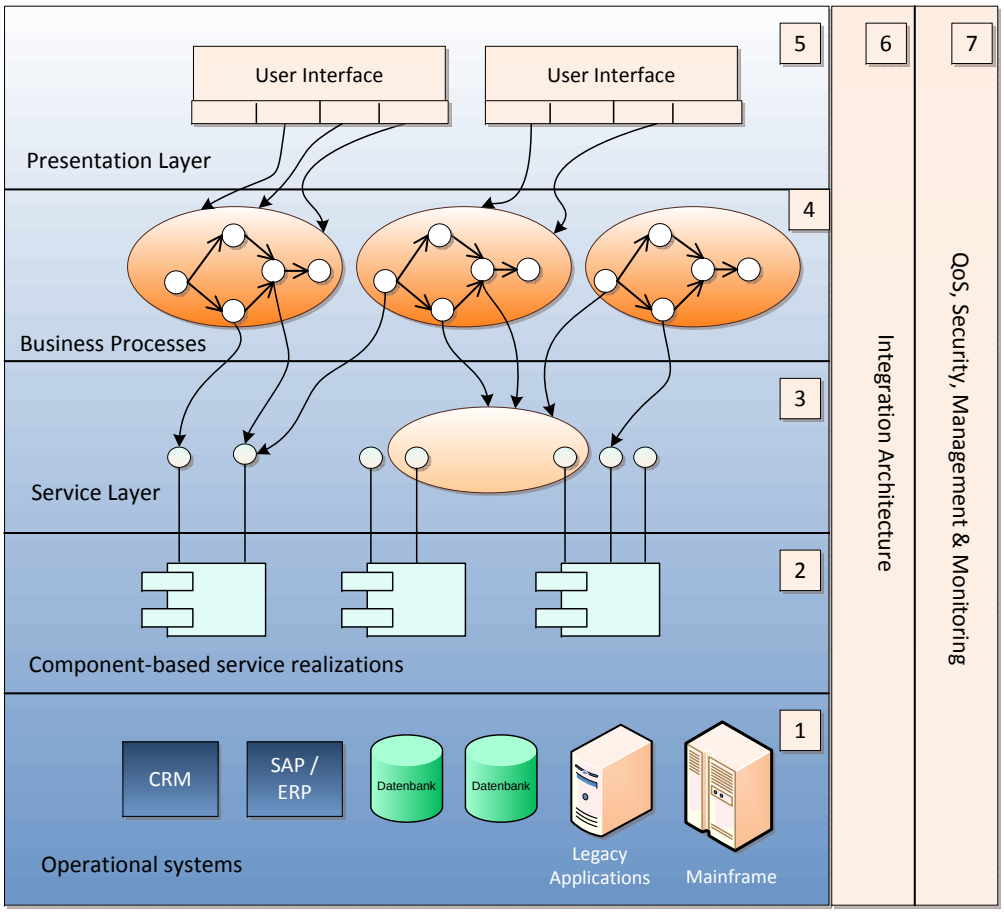

Figure 2.2: SOA Layers [13]

In the literature, several slightly different descriptions of the SOA architecture [13, 79] exist. In this work, we rely on the IBM description and terminology. SOA consists of different layers as represented in Figure 2.2. One layer relies on the layer below and uses its functionalities. The bottom layer, the *operational systems layer*, consists of existing enterprise applications and systems such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems, legacy applications or databases. The components from the components layer are responsible for ensuring appropriate QoS levels of the services and for providing enterprise functionalities. The systems from the operational layer can be integrated and used by the components from the *components layer* to build the desired functionalities. These functionalities are used to implement the services from the *service layer*. A service represents a business task that can be automated and reused by several business processes. The service description can be exposed so that the service can be discovered or statically bound and called via its interface. A service can be used as an atomic or as a composite service. Even business processes may be created from the interaction of multiple services which are triggered in a more complex flow. Business processes are realized as service compositions through orchestration or choreography at the *business process*

layer of a SOA. Since human users interact with business processes or directly with services, they are provided with a user interface. SOA separates the user interface into the *presentation layer*.

The *integration layer* is commonly represented by the Enterprise Service Bus (ESB) which assures the infrastructure for the integration of services. The ESB includes capabilities for data transformations, intelligent routing, protocol mediation, service logging, etc. The *QoS layer* of a SOA, includes capabilities for maintaining appropriate QoS and security.

2.1.3 Web Services

The Web services technology is commonly used to implement services in a SOA. A Web service is a software component that is accessible in a network via its well defined interface. Commonly, Web services are accessed by their clients in the Internet via HTTP. It is also possible that services are available in the intranet. The clients are not aware of the service's implementation but only have to know its interface in order to use the service. An application may easily call a Web service from another server and use its functionality. A Web service is defined by [79] in the following way, which corresponds to our understanding as well.

Definition 2 *A **Web service** is a platform-independent, loosely coupled, self-contained, programmable Web-enabled application that can be described, published, discovered, coordinated, and configured using XML artifacts (open standards) for the purpose of developing distributed, interoperable applications.*

Some examples of real Web services which can be accessed through the Internet, are: a service for the weather forecast provided by the *National Weather Service*, the *Google Maps* Web services that provide geographic data for maps, the *Terra Service* from Microsoft which may provide photos or a topographic image for a certain location, the *IP2Geo* Web service from *CDYNE* for mapping an IP Address to the geographic location (e.g. latitude, longitude, country, etc.), and the *Trading Web services* for accessing the *eBay* marketplace by *eBay*. Among the service providers that offer Web services through the Internet are *Google, Amazon, eBay* and *PayPal*. There are also websites, where different Web services are advertised, as e.g. *webservices.seekda.com, xmethods.net, webservicex.net* and *cdyne.com*.

The Web services may be implemented with different technologies. Currently, there are two widely spread specifications for Web services: *SOAP* and *REST*. The SOAP Web services use SOAP as communication protocol, which is based on XML messages. REST stands for *REpresentational State Transfer* and it is an architectural style for enabling communication with Web services. For more information about REST Web services please refer to [60]. REST supporters argue that it is simpler and more efficient than SOAP. However, these advantages come together with other drawbacks including lack of standardization, lack of support regarding transactions, asynchronous communication and stateful operations. REST uses solely HTTP or HTTPS as communication protocol, while SOAP may employ other protocols as well. Thus, the SOAP overhead brings also

advantages. In addition, for services over SOAP there exist a series of WS* extensions (e.g. *WS-Transactions*, *WS-Coordination*, textitWS-Security, *WS-Reliable Messaging*) that offer additional support for transactions, security, reliability, etc. Both, SOAP and REST Web services are nowadays employed, and the choice between one technology and the other depends on the particular case when it is applied.

In this thesis, we deal with SOAP Web services, which we discuss in more detail in the following paragraphs of this section. However, even if our current implementation targets SOAP Web services, our approach is not limited to services over SOAP and may be easily extended to other service technologies as well.

The previously described Web service examples are also implemented with different technologies. The *Terra Service*, the *Weather Service*, the *IP2Geo Service* and the *Trading Services* are SOAP Web services and use the XML format, SOAP communication and WSDL interfaces. The *Terra Service* was developed with the .NET framework. The *Google Maps Services* have REST interfaces and they exchange JSON objects.

Web services have the following properties [79]:

- **Interoperable:** The interoperability problem is overcome by the service concept, since a service may be implemented in any programming language and the service consumer only deals with its interface. Thus, different applications may be integrated with each other, independently of their implementation platform. The Web services rely on the XML standard which can be interpreted by many applications.
- **Loosely coupled:** Multiple concepts of a SOA and Web services, such as the service registry, service descriptions which are independent of the service implementation and platforms, and the standard protocols, contribute to a loosely coupled communication between services. Web services also support asynchronous communication.
- **Programmatically accessible:** Web services can be called by remote applications which use the services' functionalities.
- **Service description:** As mentioned before, the client calls the Web service via its interface. The service operations are defined in the interface, which is based on XML standards and the WSDL language.
- **Dynamically discoverable:** The Web service consumers can search automatically for a service in the service registry. Thus, they can find out the service description and can invoke it.
- **Distributivity:** Web services may be distributed on different servers and are usually accessed via Internet. Using HTTP as communication protocol, the services are also accessible behind firewalls.
- **Stateful or stateless:** Services may be developed as stateful or stateless Web services. Stateful Web services may keep the state between multiple operation invocations. This means that the local variables and object values that were set during an operation invocation are kept and available for the next operation invocation. For a stateless Web service this is not the case. Therefore, the local variable values are lost after the client invoked the service operation. The client

requests are independent of each other. In this thesis, solely stateless services are bound into the service composition.

- **Synchronous or asynchronous:** The communication with the Web service may be synchronous or asynchronous. With synchronous communication, the client waits for the service until it receives the response. These services are RPC-style services. With asynchronous communication, the client doesn't wait for the service response and it continues to process other tasks. The response is not received immediately. Asynchronous services are document-style or message-driven services. In this thesis, we deal with synchronous Web services.

We employ JAX-RPC Web services which expose a WSDL interface and communicate over SOAP. Therefore, we provide some foundations about SOAP and WSDL.

2.1.3.1 SOAP Protocol

SOAP [9, 74, 79, 60] is the standard communication protocol for Web services and it is based on XML-messaging. SOAP may be employed not solely for Web services, but also for other applications, such as COM objects or Java Servlets. Initially, SOAP was the acronym for Simple Object Access Protocol but now it is used as a name. SOAP specifications [9] are developed by the W3C and OASIS organizations. The requests between service client and the service are transmitted as XML messages in SOAP format. SOAP comes to support the Web service ideas regarding interoperability between heterogeneous distributed applications and loose coupling between applications. Still it has a drawback regarding efficiency since requests to and responses from the Web services need to be encoded in XML, which costs extra time. SOAP defines the structure of the messages which are transmitted over the network to the Web services with WSDL interfaces. SOAP is usually used together with HTTP as transport protocol, but also other protocols may be employed such as SMTP, FTP or RMI. SOAP is a stateless protocol. The SOAP message is contained into an HTTP request or response.

A SOAP message is built mainly of the following elements:

- **SOAP Envelope:** The envelope represents the root element and contains optionally the SOAP header element and mandatory the SOAP body element. It contains the URI to the SOAP schema that is used for the SOAP message and the encoding rules.
- **SOAP Header:** The header element is optional and contains the specification of extra information such as the message sender or receiver, QoS properties, information about transactions, object references, security with information about authentication, authorization or digital signatures, etc.
- **SOAP Body:** The body contains the application-specific data of the message that is transmitted between the service and the client. The data has to be in XML format and it represents the payload of the message. The request message contains also the name of the called service operation, and the response message contains the operation name suffixed by the *Response* char-sequence to distinguish it from their request. If a fault occurred, then the body will contain the fault message.

SOAP defines two communication styles, the RPC and the document (message) style. With the RPC style, the Web service is viewed as a remote object by the client. The client passes the parameters to the service and the service sends the result back to the client. While the RPC style can be used only for synchronous communication, the document style is also applied for asynchronous communication. With the document style, an entire document in XML format may be sent to the Web service.

2.1.3.2 Service Description with WSDL

In order to be accessible for the clients, a Web service needs to expose a service description (interface) which provides all the necessary artefacts. The Web Service Description Language (WSDL) [10] from the W3C organization emerged for specifying such an interface. WSDL [79, 74] is an XML-based language and is independent of the service implementation. A WSDL description file is mainly divided into two parts, the *abstract description* and the *concrete description*. The *abstract description* contains information about *what* functionalities the service is offering, and the *concrete description* targets the technical details *how* the consumer may *bind* to the service. The service description contains the service location and the provided operations which are similar to method signatures of a Java interface. The operations are described together with the corresponding XML-request and response messages that an operation may receive or may return. The WSDL description also contains the binding data, the supported protocols, the URL with the service location, the used messaging style (RPC or document style) and the encoding style. The latest version of WSDL (2.0) can be also used for RESTful Services.

With appropriate WSDL tool support (e.g. JDeveloper, Eclipse with plug-ins for Web services) the WSDL file is usually generated from the Web service implementation (e.g. a Java Class) and can be accessed at the URL of the Web service followed by the char sequence "?wsdl". The generated file can be manually modified according to the requirements. The tools offer support for the service consumer to create automatically proxy classes for invoking the service, on the basis of the WSDL description.

The WSDL file contains the following elements:

- **Description:** This element represents the root element of the WSDL XML file.
- **Documentation:** The service provider may write in this section information about the service and its functionality.
- **Types:** This element specification contains the types of the XML elements which are part of the messages exchanged by the service. The XML element types may be primitive (e.g. int, float, etc.) or complex types, which are built of other elements. Usually, the XML element types are defined in an XML schema which is included in the WSDL file itself or is imported from another location.
- **Interface:** This element definition describes the functionalities provided by the service and contains the list of the offered operations and the exchanged messages. Here, it is also possible to define faults that could be thrown by the service. Therefore, the *interface* element contains the *operation* and *fault* elements.

- **Operation:** The service operations are described inside the *operation* element, which contains the input and output messages that are exchanged with the service client. The *interface* element is part of the newer WSDL version 2.0, whereas in the older WSDL version 1.1 it was called *portType*.
- **Binding:** This definition contains one or multiple transport protocols (like SOAP) that are supported by the service for the message exchange. The most used transport protocol is SOAP over HTTP, which we also use for our Web services. Each transport protocol is specified in an extra binding. The binding also contains some encoding details about the operations.
- **Service:** This element defines one or more *endpoints* where the service operations can be accessed. The *endpoint* element contains the physical address of the Web service operation and the protocol information.

2.2 Business Processes and Workflows

Web services can be aggregated to accomplish a business objective such as delivering goods or producing services for a customer. The set of activities necessary for reaching a business objective build a business process.

A *business process* is defined in [87] as follows: "*A business process is a group of logically related (and typically sequenced) activities that use the resources of the organization to provide defined results. Business processes deliver value in the form of products or services, often to an external party such as a customer or partner.*"

A similar definition was given by Hammer and Champy [59], where a business process is described as "*a set of activities that, taken together produces a result of value to the customer.*" A business process [80, 74] consists of a set of activities that are executed in a defined order having a beginning and an end. A business process may involve humans, resources and materials.

Commonly, an enterprise executes a set of activities which are repeated and can be automated. Business process activities can be performed manually by employees or automatically supported by an information system. For example, a business process for booking a room at a hotel can be performed fully automatically, when the customer books a room on a web site. It is also possible that a process is a combination of manual and automatic business activities. An example for such a process is renting a car, where an employee submits the customer's ordering request to the system and another employee is in charge of delivering the car to the customer or making the protocol of the car damages.

A business process may describe the tasks accomplished inside an organization but also tasks that exceed the boundaries of an organization and sustain the collaboration of multiple organizations [106, 80, 62]. A business process can be triggered by receiving a request from the customer but also by an event (e.g. an event based on time). A process may involve resources, materials or products which are used or may be changed along the execution of the process. Business processes may be graphically represented using *business process models* [106]. Business process models represent the structure and flow of the process; they show how the involved parties interact.

Process models are useful for stakeholders to better communicate about the process and help to detect possible deficiencies in the process behavior. In this thesis, business processes are built by the interaction of multiple Web services that together achieve a business task.

A workflow represents the automation of a business process. Thus, a workflow also consists of a set of activities. The Workflow Management Coalition (WfMC) defines the terms of *workflow* and *workflow management system* [46] as follows.

Definition 3 *Workflow is the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.*

Definition 4 *A workflow management system is a software system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interacts with workflow participants, and, where required, invokes the use of IT tools and applications.*

Workflows may be described using workflow languages, such as Petri-nets, languages based on graphs, event-driven process chains or activity diagrams [41, 106, 61]. With Petri-nets business processes may be described in an abstract way, independent of the execution environment. Petri-nets are employed for modeling dynamic systems and these are represented graphically using places, transitions and directed arcs. Tokens are used for modeling the dynamicity of the system [106]. A workflow language called *Yet Another Workflow Language (YAWL)* was proposed by van der Aalst [102] in order to cover even more workflow patterns that were not supported by Petri-nets. Van der Aalst et al. provided in their work [102] a thorough analysis about workflow patterns. The graph-based languages use nodes and edges to represent process models. The nodes correspond to activities and the edges show the control or data flow [106, 41]. UML activity diagrams may also be used for describing workflows. In [51], the authors highlight strengths and weaknesses of using activity diagrams for specifying workflows and show what workflow patterns are supported by the activity diagrams.

An *Event-driven process chain* (EPC) may be used for modeling business processes and workflows [106, 3]. EPC was proposed by A.W. Scheer et al. and is part of the ARIS framework (Architecture of Integrated Information Systems). The EPCs were adopted in the industry (e.g. by SAP) for describing processes. The core constructs of event-driven process chains are functions, events, connectors (e.g. OR, AND, XOR) and control flow edges. An event-driven process chain starts and ends with an EPC event. Functions are units of work that receive an input and create an output (e.g. a physical product). Functions are triggered by events.

The *Business Process Model and Notation (BPMN)* [6] was proposed by the Object Management Group (OMG) as a graphical notation for visualizing business processes. A BPMN process describes how the process participants collaborate and how the business activities build the process flow. BPMN is addressed to business analysts but also to technical developers. The authors of [6] argue that business people prefer to visualize a business process in a flow chart format, while a language such as WS-BPEL can be understood only by programmers and computer systems. BPMN promises to

bridge the gap between the visualization of a business process and its execution format. Therefore, BPMN provides mapping mechanisms from the business process notation to a process execution language such as BPEL. The core constructs of a BPMN process are *events*, *activities* and *gateways*. An event signalizes the occurrence of a happening, for example, sending or receiving a message. The BPMN process is triggered by an event and ends with another event. A gateway represents flow decisions such as forking, merging or joining paths. The events, activities and gateways are connected by the so called *flow objects* which build the structure of the business process.

2.3 Web Service Compositions

In order to achieve a business goal or to solve a complex task, binding one Web service is not always sufficient and the interaction of several Web services is required. This results in the combination of the functionalities of multiple Web services which put together build a *composite service* [41, 52]. The process of creating a composite service is called *service composition*. The composite service may be exposed in turn as a Web service. The most common way to build a service composition is by specifying its structure using a composition language. The service composition model describes the data flow and how the interaction between the parties takes place. The composition involves several parties (e.g. suppliers, trading partners) which need to exchange messages in a synchronous or asynchronous way. A *service composition middleware* provides an adequate environment for modeling and executing such a service composition.

Depending on the time when it is built, a service composition may be *static* or *dynamic* [52]. The *static service composition* is created at design time when the composition is designed and all its components are compiled and deployed. This is a rather restrictive composition since it assumes that services and partners do not change over time. By its nature, SOA is a dynamic environment where services may change over time or new services appear. Therefore, a *dynamic service composition* is more suited in a SOA. It is necessary to bind new services into the composition without having to redesign the entire system. A dynamic service composition permits the configuration of the composition at runtime. With our BPR framework we target a dynamic service composition where services may be replaced at runtime without having to redeploy the composition.

A service composition may be created *manually* or *automatically* [52]. In this work, service compositions are created manually by a software architect or software developer who specifies the process description with BPEL. Only services are selected automatically, whereas the structure of the composition remains the same. An alternative is that the user himself is involved in composing the services, as it is described in [93, 89]. A tool is proposed by [93, 89] where the end users may compose services by themselves. Since the user does not have any programming skills, the challenge is to keep the composition process as simple as possible.

Another possibility is to create the service composition *automatically*. Usually, this kind of composition assumes the existence of an ontology and semantic annotations of services. More details about the automatic composition of services are provided in Section 2.3.2.

A Web service composition may correspond to an orchestration or a choreography model. The two models differ mostly with regard to the point of view on the interaction between the participants [82, 41].

- *Orchestration* describes the business process flow from the point of view of one single party that participates at the process. The orchestration specifies the flow of interactions between this party and the other participants of the process. The one party controls the entire business logic and how messages are exchanged. A service orchestration may be modelled with different languages, as e.g. BPEL, UML activity diagrams, Petri-nets, state-charts, rule-based orchestrations, Pi-calculus, etc. [52].
- *Choreography* shows the global perspective of the parties that are involved in the conversation. It describes "*peer-to-peer collaborations*" including the common and complementary behavior of the parties which achieve a business goal [14]. Choreography describes the exchange of messages of the collaborating parties and rules for interaction. A choreography is used for a general understanding between the parties and for generating code skeletons for implementing the Web services [21]. Languages such as the *Web Service Choreography Description Language* (WS-CDL) [14] or the *Web Service Choreography Interface* (WSCI) [11] support the description of choreographies.

2.3.1 The Business Process Execution Language (BPEL)

The *Business Process Execution Language* (shortly BPEL) [16, 63] is an XML-based language for specifying Web service compositions. A BPEL process description specifies how several Web services collaborate in order to solve a more complex task. In this thesis, BPEL was employed for describing service compositions. The *Web Services Business Process Execution Language* (WS-BPEL) is an OASIS standard and a successor of the previous language version, BPEL for Web services (BPEL4WS). Service compositions described with BPEL are deployed and executed on a BPEL engine, such as the Oracle BPEL Process Manager [8], the Apache ODE (Orchestration Director Engine) [2], the IBM Business Process Manager [7] or the Microsoft BizTalk Server [5].

A BPEL process itself is exposed as a Web service, thus providing a WSDL interface for its clients. The partner Web services integrated into the BPEL process are called via their WSDL descriptions. The messages exchanged by the BPEL process with its partners are usually kept in variables. Commonly, a process is stateful and its state is kept in the variables. The $< partnerlink >$ elements defined in the process description indicate the partners that the BPEL process interacts with, like other partner Web services or the clients that invoke the BPEL process. Inside the $< partnerlink >$ construct, the roles of the partners are defined.

The BPEL specification distinguishes between *basic* and *structured* activities [16, 63]. Basic activities may have different purposes like calling a Web service (with $< invoke >$), exchanging messages with partners (e.g. $< receive >$, $< reply >$), signalizing faults (e.g. $< throw >$, $< rethrow >$), updating variables (with $< assign >$) or terminating the process (using $< exit >$). The $< wait >$ activity is used for including delays in the process execution for a period of time or until a deadline is reached.

Basic Activities	Description
`invoke`	calls a Web service operation.
`receive`	specifies the `partnerlink` from which the process expects to receive a message.
`reply`	sends the response to a message received by the activities `receive` or `pick`.
`assign`	copies data between variables.
`wait`	specifies a period of time or a deadline for a delay of the process execution.
Structured Activities	**Description**
`sequence`	contains activities that are executed in sequential order, in the order of their specification.
`flow`	contains activities that are executed in parallel.
`while`	repeats the execution of the activities inside it as long as an expression evaluates to true.
`pick`	waits until the expected message is received and executes the activity associated to that message.

Figure 2.3: BPEL Activities

When calling a Web service operation using the $< invoke >$ activity, this contains the targeted partner (in the $partnerlink$ attribute) and operation. In a synchronous call, the $< invoke >$ activity contains input and output variables, for the input message that is delivered to the partner operation and for the message returned by the partner.

In a common scenario, a BPEL instance is started with the arrival of a message from the client by the $< receive >$ or the $< pick >$ activity. After processing the request, the response is sent back to the client by the $< reply >$ activity. A synchronous communication with the client is denoted by a combination of the activities $< receive >$ and $< reply >$ corresponding to a request/response operation. In an asynchronous communication the $< reply >$ activity is missing and if needed an $< invoke >$ activity may be called to send a response back to the client.

The structured activities are applied for defining loops (e.g. $< while >$, $< forEach >$, $< repeatUntil >$), for specifying conditional behavior ($< if >$) or for defining the processing order ($< sequence >$, $< flow >$). Some activities (e.g. $< while >$, $< if >$) have similar meanings as their corresponding constructs from programming languages like C or Java. Activities may be called sequentially by nesting them inside a $< sequence >$ activity. This means that the activities are triggered in the order they appear in the $< sequence >$ activity. In case the activities are required to be executed

in parallel, these are nested inside a $< flow >$ activity.

BPEL provides mechanisms for compensation, fault and event handling. Faults may occur during the interaction with a partner or internally triggered by the $< throw >$ activity. Different kinds of faults are handled using the $< faultHandlers >$ element. In case that activities need to be reverted, a compensation handler (specified with the $< compensationHandler >$ element) may be used. The BPEL process is able to listen to events and handle them when they occur. This is possible by using event handlers defined with the $< eventHandlers >$ element [63].

Figure 2.3 provides examples of BPEL activities. For a complete list of the BPEL activities we refer to [16].

Within a process description, we also consider sub-orchestrations which we call *sections* of a business process. By *section* we refer to a part of a BPEL process which begins with one activity and ends with another activity. The second activity is triggered after the first one. Also, a structured activity like a `while` loop may represent a section, where the section contains all the activities nested inside that structured activity.

2.3.1.1 BPEL Process Example

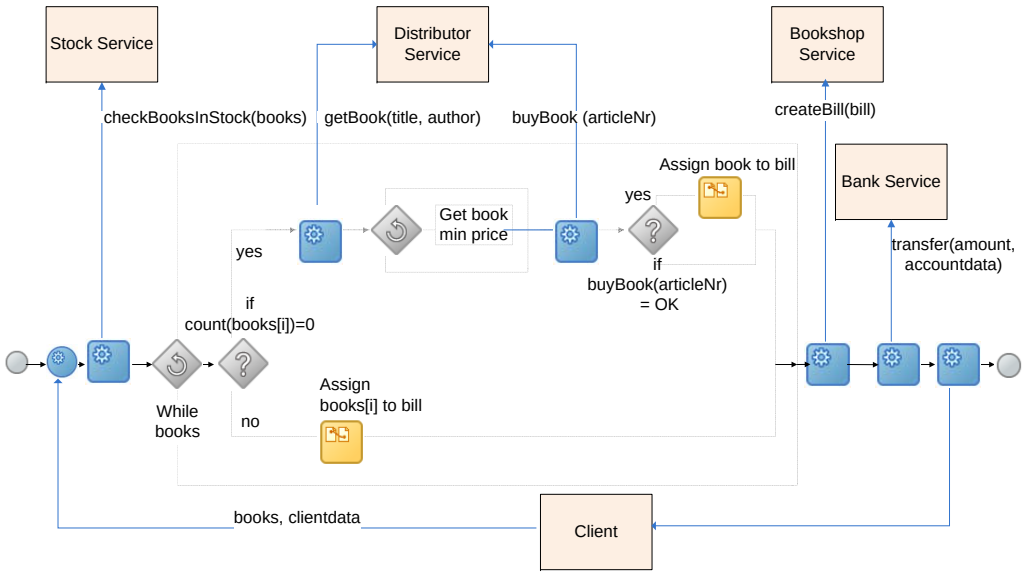

Figure 2.4: Bookshop BPEL Process

As an example, we consider a *bookshop process* for buying books in an online shop, which we implemented using WS-BPEL. A rough structure of the bookshop process is represented in Figure 2.4. Four atomic services are involved in the process: a *stock service*, a *distributor service*, a *bookshop service* and a *bank service*. The *bookshop process* starts with the arrival of a request from the client, which contains the list of books the client wants to purchase at the online bookshop. The *stock service* verifies whether all of the books from the list are in stock. If this is not the case, the *distributor service* is invoked to purchase the missing books from a book wholesaler. If the *distributor*

service returns several books with different prices, the books with the lowest prices are selected to be bought. These books are assigned to the client's bill and the *bookshop service* is invoked for creating the bill. Finally, a *bank service* is invoked to make the withdrawal from the clients' credit card to the online shop account.

2.3.2 Semantic Web Services and Compositions

With WSDL, service descriptions are specified syntactically, but for automatic service discovery and matching, the syntactic description is not sufficient and also a semantic description is needed. With semantic descriptions, the service interface is annotated with semantic concepts so that services may be searched and discovered by their semantic meaning.

When a service from an application or a BPEL process has to be replaced with a newer version of the service or a service from another service provider, a common situation is that the newer service has a slightly different interface and its description does not entirely match with the old interface. This situation may trigger the interruption of the BPEL process execution, causing it to fail. Therefore, a *semantic matching* process is supposed to find out whether the required service interface and the provided interface can be mediated. The differences between the interfaces can be overcome through mediation, and the old service may be replaced with a semantic equivalent service.

Semantic service discovery and description matching is a widely studied research topic. Küster et al. propose in [67] a service description language and matchmaking algorithms for automatic service discovery and composition. In [68], the authors discuss the evaluation of semantic service discovery approaches.

In the ADDOaction project [57, 18], our research group also addressed the subject of semantic service discovery. Since this topic was addressed in the ADDOaction project and proper solutions were developed, the semantic service discovery is out of the scope of this thesis. However, we will give some insights about the solutions from the ADDOaction project. Bleul et al. present in [35, 34, 36] an approach for semantic interface mediation that relies on XSLT transformations. The service operations parameters are annotated with semantic concepts from taxonomies using the Web Ontology Language (OWL) [107]. A mediation is solely possible if the elements of the messages, exchanged by the service operations, of the offered and the requested interfaces are semantically equivalent. The WSDL of a service is extended with semantic annotations. This semantic extension is called Mediation Contract Extension (MECE) and with MECE the service description remains compliant with the WSDL specification. More details about MECE can be found in [35, 34, 36]. A matching algorithm is employed that encompasses a functional and a QoS matching. Functional matching targets message transformations where XSLT documents are generated on the fly. Message transformations include matching of the inputs and outputs of service operations by comparing their semantic representations. The relationship (e.g. specialization) between the semantic representations is considered. The matching algorithm supports data type transformations and also unit transformations for message parameter values that have a quantity (e.g. for a price parameter, a currency transformation between Euro and Yen). These transformations are supported by plug-ins which are software components (e.g. Java Classes) and provide adequate methods for conversion. QoS levels may also

be matched and they are semantically annotated by using a Service Level Ontology [33]. The QoS requirement matching includes a transformation between QoS metrics (e.g. seconds, milliseconds), a transformation between predicates (e.g. greater, greater equals) and QoS dimensions. For a successful matching, it is necessary that all the requested operations are found. The success of the matching process is reflected by a ranking score. The score is computed based on the semantic distance between the concepts and the number of plug-ins used within the transformation. If multiple matching results are found, these are sorted by their ranking score. If a matching is not possible, adequate reports provide information about what caused the matching process to fail.

Another topic that was addressed by the ADDOaction project was the automatic composition of semantic services [105, 29], and three algorithms [104] have been developed to solve this problem: the IDDFS, a Greedy- and a genetic algorithm. These algorithms have proven their performance at the *Web Service Challenge* [29], an international competition that gives researchers the opportunity to meet and compare their research results.

2.4 Terminology

In this section, we introduce the main concepts and terms used in this thesis and describe our interpretation.

- **Abstract service:** An abstract service represents the functionality of a service. The term *abstract* is correlated to the time when the service composition is designed, when it is known what functionality is desired but it is unclear which concrete service will be bound to implement that functionality. An example of an abstract service is a *Shipment Service* which offers the functionality of shipping goods to the customers.
- **Concrete service:** A concrete service represents the realization of an abstract service. We assume that there are several *concrete services (candidates)* that provide the same functionality but have different QoS properties. For example, the Shipment Service may have multiple realizations such as the *DHL-Shipment* Service, *Hermes-Shipment* Service or *DPD (Deutscher Paket Dienst)-Shipment* Service. All these concrete services offer the same functionality of shipping goods. In this example, the three services are offered by different companies. It is also possible that the same provider offers multiple concrete services with different service levels and prices.
- **Business process:** Definitions for a business process were introduced in Section 2.2. In this thesis, we refer to business processes that are realized as service compositions. For example, *Car-Rental* is a business process for renting cars. It is realized as a service composition where a *Car-Insurance* service and a *Financial* service are involved. We use the terms of business process, process, service process, service composition and BPEL process interchangeably.
- **Process instance:** A process instance refers to a single execution of a business process. It represents a concrete execution path of the process. For example,

John Dales wants to rent a car for his trip on Mallorca. Therefore, he enters his request into the Web site and specifies the type of car he wants to rent, the dates and his credit card data. The request order of John Dales is received by the *Car-Rental* business process and a new instance of the process is created which processes the order of John. If the processing was successful, John Dales receives a confirmation message that the car was reserved for him. This corresponds to the last step of the process and thereafter the process instance is terminated.

2.5 Quality Properties

Choosing the services with appropriate Quality Properties (QP) is an important task that contributes to the success of the business process. By Quality Properties we refer to both, QoS and QoE dimensions, which we further introduce in this section. While the QoS are technical parameters that mainly show the performance of the services, the QoE is a reflection of the users' perception about the service. Nevertheless, the QoE is as important as the QoS when it comes to choosing the right service. However, Quality Properties are not part of the BPEL specification and these need to be treated separately.

2.5.1 Quality of Service

QoS are non-functional properties that reflect how well a service is able to perform as expected by the service provider and by the service consumer [79]. A service has to be reliable, to respond in a reasonable time and to fulfill other agreed policies such as security or privacy. QoS for Web services include quality characteristics such as *availability, reliability, response time, throughput, regulatory, scalability, integrity, flexibility, security,* etc. Usually, a service provider advertises his services at different service levels which are agreed with the consumer in a contract, the so called Service Level Agreement (SLA). Services with better performance and quality are usually advertised at higher prices. Not performing at the right service levels as stated in the SLAs may cause penalties for the service providers [75]. QoS of Web services are very important since it can make the difference between multiple services that have the same functionality. Therefore, QoS is a strong argument for choosing between one service or another in a competitive market like the Internet [79].

Different kinds of QoS parameters are defined in the literature [83, 72, 79, 12, 110, 40, 73, 109]. Sometimes QoS are interpreted differently depending on the applications or domain. In our framework, we consider QoS parameters like *availability*, *reliability*, *response time*, *throughput* and *cost*. This list of QoS is not exhaustive. However, the framework is not limited to these dimensions, as it can easily be extended to other QoS properties as well. We distinguish between QoS values that are measured for the atomic services and the QoS computed for the entire service composition. In this section, we present some QoS definitions to clarify how we interpret the QoS in this thesis. In the BPR framework, each service that is part of the composition is automatically monitored by proxies. We consider QoS measurements for one service invocation (called the instance QoS) but also for multiple service invocations. In

Chapter 6, we describe how the QoS are computed for the entire service composition by employing our QoS aggregation algorithms.

2.5.1.1 Definitions

We interpret the QoS of a Web service similar as it is described by Zeng et al. [110], Yu et al. [109], and Jäger et al. [61].

- **Response Time**
  Response time (r) (see [110], [109]) of an operation (op) and invocation (i) of service (s) is computed as the duration of the time when the operation was requested and the time when the result is received by the requestor. It is the sum of the processing time $T_{process,i}$ and the transmission time $T_{trans,i}$ over the network:

$$r_i(s, op) = T_{process,i}(s, op) + T_{trans,i}(s, op) \tag{2.1}$$

  We also call the response time computed in this way as the **instance response time** being the response time for one service invocation.

  The response time of a Web service operation for a given time interval I (e.g. the last n seconds) is computed as the average value of the response time values that were measured for multiple service invocations that completed successfully within the interval I. If the service was not accessible or it failed, this response time value is ignored and will not be considered in the average computation of response time.

- **Reliability**
  The reliability (rel) (see [110], [40], [109]) represents the probability that the request is responded correctly within the maximum expected time frame specified in the Web service description. The reliability is computed as the proportion between the number of service invocations that completed successfully (N_s) and the total number of invocations (N) that were monitored during a given time interval I.

$$rel(s, op) = \frac{N_s(s, op)}{N(s, op)} \tag{2.2}$$

  We define the **instance reliability** as being the reliability of one service operation invocation, which may receive either the values 1 or 0. The value 1 is assigned if the service operation completed successfully within the maximum expected time frame. Otherwise, the reliability is assigned with the value of 0 if the service operation was not accessible, if it returned with a failure or if it exceeded the maximum time frame. The instance reliability is defined in the same way for a section and for the whole process.

- **Availability**
  The availability (a) (see [110], [109]) is the probability that the service is accessible. It is computed as the ratio of the sum of the lengths of the time

intervals $T_a(s, op)$, when the service operation was available, to the length θ of the monitoring interval. The time duration θ is set by the administrator of the service community and depends on the application, on how frequently the service is accessed.

$$a(s, op) = \frac{\sum T_a(s, op)}{\theta} \tag{2.3}$$

We define **the instance availability** as being the availability for one service invocation. The value for the instance availability may be either 1 if the service operation was accessible, or 0 if the service operation was not accessible. The same definition is applicable for the process as well.

- **Throughput**
  Throughput (t) is defined in [61] as "*the amount of processable data per time unit. Usually, the throughput is given in Bytes/sec and is interpreted as an increasing dimension.*" We also adopt this definition.

  In our framework, we compute the **instance throughput** for a service invocation with the following formula, representing the amount of bytes measured during one second:

$$t(s, op) = \frac{b(s, op)}{r(s, op)} \tag{2.4}$$

  whereas b represents the number of transferred bytes and r the response time that was measured for the service invocation.

- **Cost**
  The *cost* of the service is the price of invoking an operation of the service. The cost for a time interval I represents the average of all the cost values for the service invocations during that interval.

The Web service performance can be affected by the messaging protocols (e.g. SOAP) that Web services rely on [72]. The SOAP protocol consumes extra time for the XML parsing of the messages. The other disadvantage of SOAP is caused by the way XML data is represented, resulting in a large size of data in comparison to a binary representation. Performance problems of Web services can be caused by other factors as well, e.g. the response time of the Web server hosting the Web service or by the performance of database or legacy systems used by the Web service.

Other quality aspects mentioned by [72] are *integrity*, *regulatory* or *security* aspects. A list of QoS for Web services was also discussed by Ran [83] who refers additionally to *scalability*, *robustness/flexibility*, *exception handling*, *stability* and *completeness*. The requirements for quality aspects are agreed in the SLA.

- **Integrity** refers to keeping the integrity of data when several transactions operate on this data. It assumes proper mechanisms for Web service transactions where a set of activities has to be treated as a single unit. Transactional processing is quite challenging when it targets long-running and distributed Web services flows.

- **Regulatory** represents the compliance of the Web service with standards (e.g. WSDL, SOAP), rules, law, but also with the SLA.
- **Security** involves the support for different mechanisms such as authentication, authorization, confidentiality, the encryption of messages, accountability, traceability, non-repudiation, etc.
- **Scalability** is related to throughput and represents the ability of the system to increase its computation capacity for handling more requests in a given time period.
- **Robustness/flexibility** refers to the capacity of the service to function correctly even when invalid or incomplete requests are received.
- **Stability** represents the frequency of change regarding the service interface or implementation.
- **Completeness** is the difference between the set of features that are specified and those which are actually implemented.

2.5.2 Quality Property Types

Since the quality properties can be viewed from different perspectives, we will give an overview of the different types of quality properties as they are interpreted in this thesis.

- **QoS promised** (QoS_p) - represents the QoS value that was promised by the service provider. The service provider advertises the service at certain QoS levels for the service clients and these QoS values are stored in the service registry.
- **QoS monitored** (QoS_m) - this value constitutes the QoS measurement that was monitored by the BPR framework. The monitoring is performed automatically by service proxies.
- **QoS experienced** (QoS_e) - this quality property represents the QoS experienced by the service consumer. While the previous QoS_m is monitored on the BPR server, where the service composition is executed, the QoS_e is measured at client side.
- **QoS predicted** (QoS_r) - indicates the QoS value that is predicted by the BPR framework for a certain client or for clients with certain context data. The QoS prediction relies on QoS information of past service invocations.
- **Quality of Experience** (QoE) - the QoE is expressed by a rating score and represents the satisfaction of the human user regarding a service that he used. We view QoE as a subjective value since it is assigned by a human user to the service. In our framework, the QoE can be assigned for the entire process but also for an atomic service. It is a measure for the overall degree of user satisfaction concerning, for example, the GUI, the offered functionality and the performance of the service.

2.5.3 Quality of Experience

QoS properties are technical dimensions which are very important for a good performance of the services. These are measured automatically by service proxies. Thinking from the perspective of the human user, he is expecting that the service behaves well, but this is not the only criteria that he is asking for. Furthermore, the human user is interested whether the service has the expected functionality and fulfills his requirements. Service providers are interested in how users perceive their services in terms of quality, usability and price [54]. Since user satisfaction has a major role for the success of the business, we particularly want to stress that it should be considered for the selection of Web services. The same idea is suggested by other authors [44] who argue that: "*users as consumers of services need to be able to assess whether a (web) service matches their needs in an aesthetic, functional, timely and financial manner*". The term Quality of Experience is defined by ITU-T Rec. P.10/G.100 as "*the overall acceptability of an application or service, as perceived subjectively by the end user*". Another definition was given by Brunnström et al. at the Dagstuhl Seminar [38] and this definition corresponds also to our interpretation of QoE:

Definition 5 *Quality of Experience (QoE) is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and or enjoyment of the application or service in the light of the user's personality and current state. In the context of communication services, it is influenced by content, network, device, application, user expectations and goals, and context of use.*

The QoE is influenced by several factors which were divided by [38] into three categories: *human, system* and *context factors*. The factors may have relations between each other.

- **Human factors** involve characteristics of human users such as psychological, sociological factors, the experience of the user with similar services, the own user expectation or the emotional state of the user [94, 38].
- **System factors** refer to technical quality properties of an application or service. Some examples are properties of the provided content, capabilities of devices involved in the communication and characteristics of the data transmission between the application and the user over a network [38]. Thus, QoS parameters as service properties can be regarded as system factors and may influence the QoE of the users.
- **Context factors** are properties that characterize the "*user's environment in terms of physical, temporal, social, economic, task, and technical characteristics*"[38]. Some examples are the user's location, the date and time, the brand of the service, the costs, etc. In this thesis, we also discuss the influence of the users' context data (e.g. location, age) on the QoE and QoS. These are represented in Figure 2.5.

There are two possibilities for assessing QoE, namely a *subjective* and an *objective testing* [54, 45]. In the *subjective testing* real users are involved for expressing their

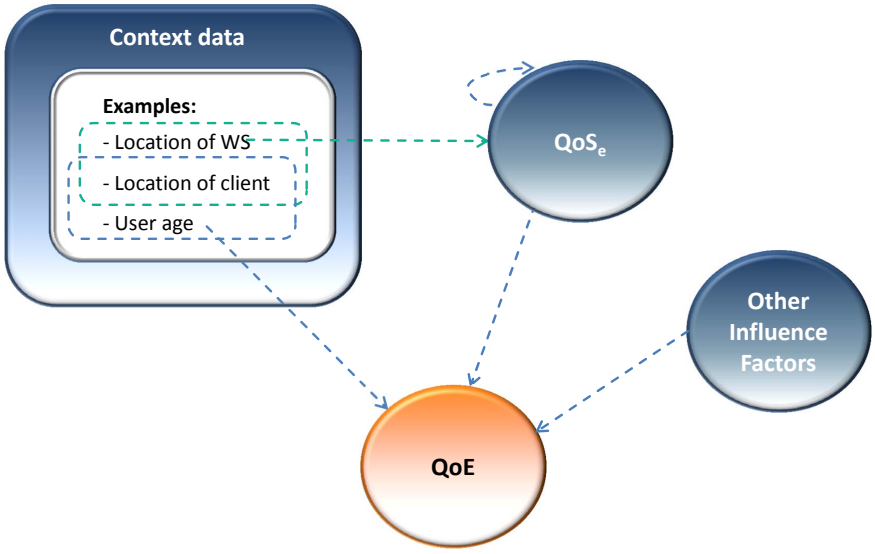

Figure 2.5: Influence Factors on QoE

opinions regarding the service or application. For this type of testing, many users have to participate in order to obtain statistically relevant results [54]. With *objective testing*, an algorithm is used to make predictions of user perceptions on behalf of real users.

While QoS refers to the performance of a service or a network, QoE is a more complex dimension which in addition depends on human psychology, social factors and economic factors. Several studies [54, 91] analyze the relationships between QoS and QoE. In [54], the authors analyze this relationship for a voice over IP (VoIP) service and for Web browsing. The authors show an exponential relation between QoE and network QoS disturbance. In the analyzed scenarios, while the QoS disturbance increased, exceeding a certain threshold value, an exponentially decrease of the QoE value was detected. For the VoIP service, the QoS disturbance was analyzed in terms of jitter and packet loss ratio and reordering. In case of Web browsing the QoS disturbance was simulated by increased response and download times of a Web session.

The work of [91] also tries to infer relationships between network QoS and QoE by analyzing the network traffic. The authors discuss the relationships between throughput, loss ratio, download times of a Web page and the user satisfaction. Thus, it is to be expected that a bad service or network performance would contribute to the decrease of user satisfaction. A bad network performance would even lead to the fact that a user gives up, not using the service anymore [91].

Especially for audio and video streaming it is common practice to measure the QoE based on the Mean Opinion Score (MOS). We consider that this type of score is suited for Web services as well and adopt it in this work. Figure 2.6 represents the possible values of the QoE parameters starting from 1 meaning *bad* to 5 meaning *excellent*. In our framework we assume that users submit ratings for the invoked Web services in the interval 1 to 5.

Quality	Excellent	Good	Fair	Poor	Bad
Rates	5	4	3	2	1

Figure 2.6: QoE Values

2.5.4 Service Level Agreements

A service level agreement (SLA) (see [101], [71], [66], [115]) for Web services represents a contract between the service consumer and the service provider regarding the QoS levels of the Web service. Usually, a SLA contains the following information:

- the **signatory parties** which are involved in the service agreement,
- **validity** containing the period of time when the contract applies,
- **the QoS parameters** to be monitored,
- the **Service level objectives (SLO)** with the QoS constraints that are to be fulfilled,
- **penalties** in case of breaking the SLA,
- **financial agreements** with the price of using the service.

The violation of SLAs usually leads to monetary penalties for the service provider. The service provider may also offer different package levels like *Gold* (e.g. with very good QoS levels), *Silver* and *Bronze* (e.g. with less good QoS levels) for different classes of customers [115].

For the specification of service contracts several languages have been proposed, like the Web Service Level Agreement (WSLA) Language ([71], [66]) proposed by IBM, the Web Service Offerings Language (WSOL) [97] or WS-Agreement [20]. All three languages have XML syntax.

It is also possible to perform a negotiation for the establishment of an SLA. Approaches like [78] [115] propose an automatic negotiation process by exchanging offers between the service provider and consumer. However, the negotiation of SLAs is out of the scope of this thesis. We consider the advertised service levels of the service providers (the promised QoS) which are retrieved from the service registry.

Unfortunately, not always SLAs are fulfilled, thus causing inconveniences for the service consumers. For these reasons, the QoS are permanently monitored in our BPR framework. Specifying the BPR rules appropriately and triggering them at the right moment of time may prevent SLA violations.

2.6 Context-awareness

In this thesis, our target is to select those Web services that are trustworthy, hold their promised QoS and receive a high QoE from the service consumers. Since we assume

that there is a large number of Web services with different QoS we employ prediction techniques to select the 'best' Web services for the users with regard to their contexts. Several approaches are known for predictions, such as neural networks, regression analysis or collaborative filtering.

Prediction techniques are also used in recommender systems in order to recommend certain items or objects to the users from a large set of possibilities. Several recommendation methods have been developed to deliver personalized recommendations to the users so that the user receives the most 'interesting items' for him, e.g. recommending movies in an online shop. This has been adopted successfully also in commercial sites such as Amazon and eBay. Our approach is similar to a recommendation system in the sense that is selects one service from a large number of services by using the predicted values of QoS and QoE. However, there is also a difference between our approach and the recommendation techniques. Recommendation approaches [113] target, in general, one or more items that are recommended for one user, called the *active user*. In comparison, our goal is to predict the QoS and QoE values for all available services for an abstract service. The prediction is performed not for one active user but for an entire user group. A user group contains many users which can be similar with regard to their context data or not. The selection of one service from the service set is performed by the service selection module using an objective function.

In this thesis, one of our goals is to make personalized predictions for the QoS and QoE values of the atomic services which are used for creating the service composition. The personalized predictions are targeted for a group of users with certain similarities based on their context data. The user's context data are used for building context-aware service compositions.

The relevance of context as information for improving the behavior of systems and applications was addressed by many research works [56, 58, 100, 65]. Context is defined in [19] as follows.

Definition 6 *Context is any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object.*

With regard to Web services context information is interpreted by [100] as "*any additional information that can be used to improve the behavior of a service in a situation.*" Without considering additional information the service is also able to work, but with additional information the service performs even more appropriately [100]. The authors from [52, 65] have a similar interpretation of context and regard it as "*information utilized by the Web service to adjust execution and output to provide the client with a customized and personalized behavior*".

Context-awareness has been widely studied in correlation with mobile and ubiquitous computing [58, 65]. However, in this thesis we do not target mobile Web services or mobile users. We focus on services and service compositions which are executed on servers, where the service provisioning is done on the basis of the available context data of services and users. In comparison to the common understanding of context-awareness of mobile applications, where the user is supported in every situation to accomplish his tasks, our goal is to offer personalized solutions for users characterized by certain context data.

The consideration of context data in the prediction brings two important advantages. On one side, taking into account the context data, it improves the prediction quality. On the other side, with context data we may build context-aware service compositions. This composition is created by provisioning services for a specific user group characterized by certain context values. In this thesis, we consider the context data of users and services (e.g. the location) in the prediction of QoS and discuss the prediction of QoE. More details about our context-aware service compositions and the prediction approach are presented in Chapter 8.

3 Solution Overview

This chapter aims to provide an overview of the solution with a focus on the BPR framework, showing how the different parts of the framework work together to achieve an appropriate QP management. A more detailed description of the BPR framework and its implementation can be found in Chapter 9.

3.1 The BPR Approach

In this thesis, we propose a novel solution approach which encompasses the BPR framework, the BPRules language, flexible monitoring techniques, service selection algorithms and QP predictions for services. The BPR framework offers comprehensive support for the QoS and QoE management of service compositions. The mechanisms for monitoring, service selection and QoS and QoE predictions were developed in different modules of the BPR framework: the *QoS Monitor and Aggregator* module, the *Service Selection* module and the *CoFee* module.

The BPRules language is applied to exploit the features of the BPR framework. Rules specified with the BPRules language are called BPR rules and are interpreted and executed by the BPR framework. One may query the QoS monitored by the *QoS Monitor and Aggregator* module or the quality values predicted by *CoFee*. BPRules makes the features of the BPR framework accessible without having to know the internals of the framework. In reaction to undesired QoS and QoE values detected for services, process sections or the composition, management actions may be undertaken. One important action is to trigger the selection of new services for the composition. The selection is done by the *Service Selection* module using one of our selection algorithms (see Chapter 7). If services with better QoS and QoE levels are found, which fulfill the desired QoS and QoE constraints specified in the BPR rules, then the old services are replaced with the new ones.

QoS and QoE values are predicted by the *CoFee* module (see Chapter 8) based on context information of services and users and based on QoS and QoE values that were collected from past service invocations. Context data of users and services (e.g. the distance between the service and the user) may be used for creating context-aware processes, which are personalized for their users.

Several people with different roles are involved in the interaction with the BPEL process and the BPR framework:

- **Software developer:** In the design phase, the software developer designs and implements the BPEL process. Finally, he deploys the BPEL process on a BPEL engine.

- **User:** The user is a human that interacts either with the BPEL process or directly with atomic services. In our scenarios, a user of the BPEL process sends a request for processing. Users may either call directly the atomic services or they may interact with the services as part of the BPEL process. After interacting with the services, the users are asked to submit the QoE, according to their experience with the services.
- **Business analyst:** The business analyst is responsible for the well-functioning of the business processes and the requirements specifications for Web service provisioning. In some cases, the tasks of the business analyst may be undertaken also by a software architect or developer. In the BPR framework, the business analyst is in charge of specifying the BPR rules. The BPR framework offers the possibility to the business analyst to be informed of every important event concerning QoS and QoE. Thus, the BPR framework provides several reports (which are described in the next section) about the behavior of the business process for different time periods. The business analyst is able to act upon possible QoS or QoE violations by specifying adequate BPR rules. He has also knowledge about the business strategy of the company, like the target user groups of a business process and the context data that characterize the users. This context information is necessary when the business analyst provisions services for a personalized service composition which is addressed to users with a specific context. The roles of service provider and service consumer are interpreted as described in Section 2.1.1.

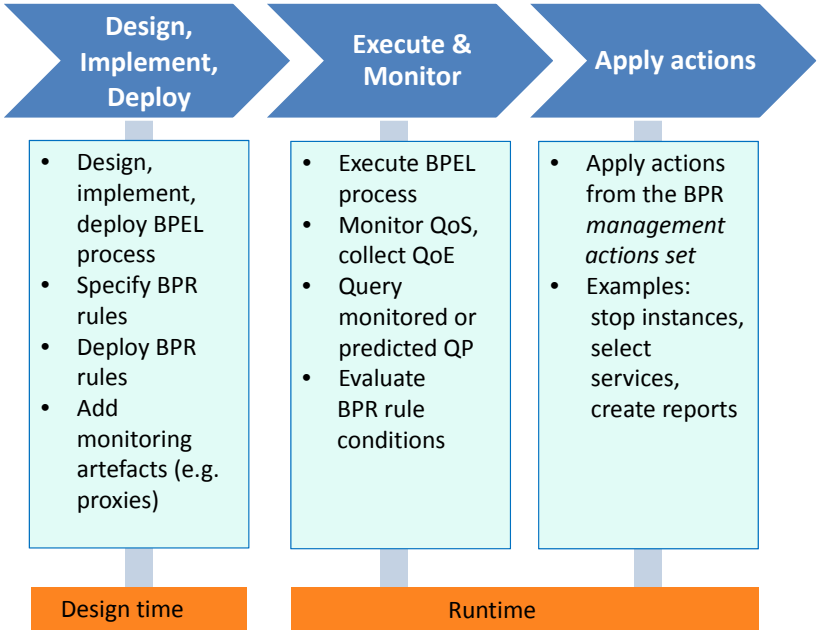

Figure 3.1: Managing a BPEL Process

Figure 3.1 represents the phases that the BPEL process traverses during design time and runtime and how the process is managed by the BPR framework. The business

analyst specifies the BPR rules in the BPR document associated to the BPEL process. The BPR document is stored in the BPR repository (as represented in Figure 3.2) and it is interpreted by the BPR framework. During process execution, the QoS of the services are monitored automatically by proxies and the QoE values are collected from the users that invoked the services. The *BPRules Manager*, as the core module of the BPR framework, retrieves the monitored, predicted or estimated QoS and QoE values. With these values the conditions of the BPR rules are evaluated upon compliance. In case the conditions evaluate to *true*, the corresponding actions (e.g. service selection, message notifications) are triggered and the BPEL process is managed.

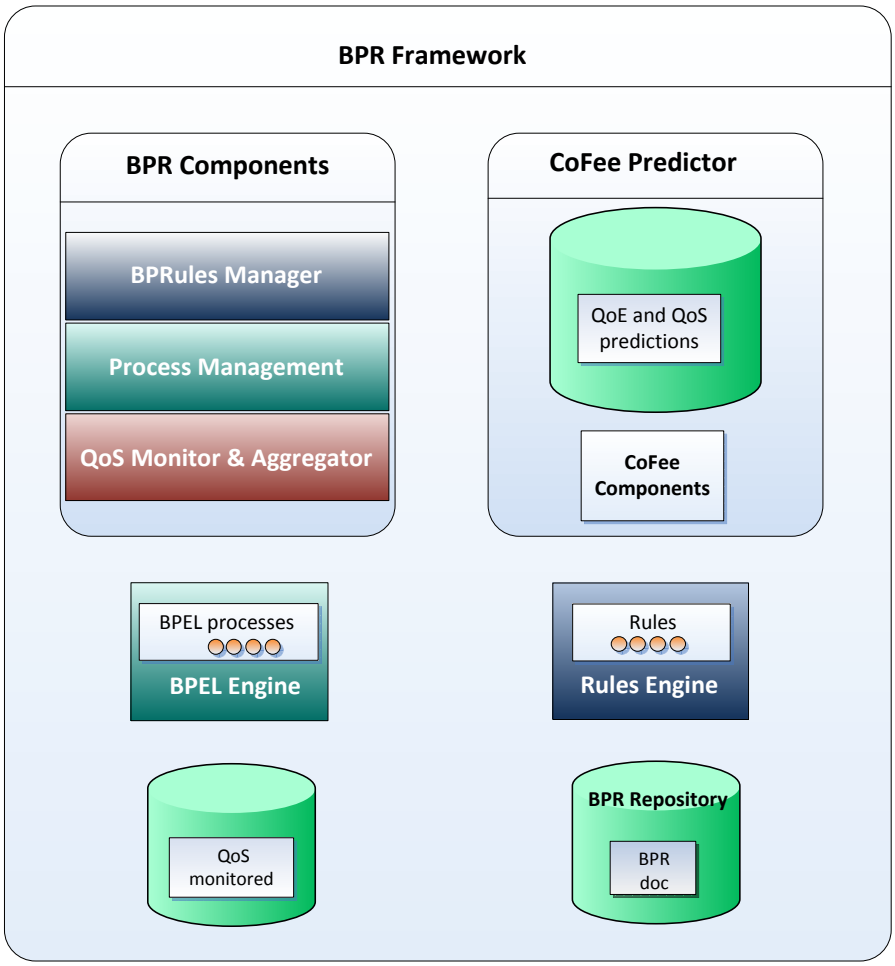

Figure 3.2: Architecture Overview

We give a short overview of the modules and components that build the BPR framework. These are represented in Figure 3.2.

- **BPR modules:** The *BPRules Manager* is responsible for coordinating the communication with the other BPR modules in order to execute the BPR rules. CoFee is a module that may also work independently of the BPR framework. Its role is to predict the QoS and QoE for atomic services. These values are used for the selection of services. More details about the modules of the BPR framework and

how they interact with each other can be found in Chapter 9.

- **BPEL engine:** The BPEL processes are deployed and executed on the BPEL engine. We employed the Oracle BPEL Process Manager to run the processes. The BPR components communicate with the BPEL engine by using the API provided by Oracle.
- **BPR repository:** The *BPR repository* stores the BPR documents. The BPRules Manager loads the BPR documents from the BPR repository.
- **Rules engine:** The Drools rules engine is used to evaluate the BPR rules. The BPR documents are transformed automatically into the Drools format in order to be interpretable by the rules engine.

4 Related Work

In this chapter, we discuss related works that target topics such as QoS Management and Monitoring of Workflows, Service Selection and Service Recommendations. We compare the related works with our approach and highlight our contributions to the state of the art.

4.1 QoS Management

By addressing QoS requirements for services, our BPRules language has similar goals as the two languages *Quality of Service Language for Business Processes (QoSL4BP)* [22, 23] and the *Web Service Requirements and Reactions Policy (WS-Re2Policy)* [85] language. All three languages have a similar structure by means of specifying actions to be undertaken upon QoS violations. Even though, BPRules, QoSL4BP and WS-Re2Policy differ in the provided features and syntax.

BPRules offers various additional features like *instance-set handling*, *dynamic rule set change* and the specification of *rule sets applied on instances from different time periods*, which are not supported by the other languages. Also BPRules provides increased flexibility for the *QoS data retrieval* by monitoring of past and/or running process executions and allows to estimate QoS values for other potential service candidates. Similar to the *section control* feature from BPRules, the authors from [22] are able to query structured activities for QoS. However, they cannot relate QoS parameters from different sections like in BPRules (e.g. the *response time* from the *distributor section* is less than 1/2 of the *response time* of the bookshop process). A crucial action for managing QoS is the service selection action. The *service selection* is supported by all of the three languages but the used selection algorithms are different. In [23], it is mentioned that a constraint programming and a backtracking algorithm are used, but no further details are provided. BPRules may employ our $OPTIM\_S$, $OPTIM\_PRO$ or $OPTIM\_HWeight$ algorithms. We can use $OPTIM\_S$ for a *local*, a *global* or a *heuristic* search, thus having the possibility of choosing between the solution quality and the execution time.

In the following paragraphs, we provide more details on WS-Re2Policy and QoSL4BP, and also discuss further languages and approaches for QoS management of Web service compositions.

4.1.1 WS-Re2Policy

Nicolas Repp et al. propose in [85, 84] WS-Re2Policy. The language is based on XML, it is compliant to WS-Policy and is intended to allow specifications of QoS requirements for business processes. From these specifications monitoring and management

needs are derived. The corresponding architecture named Automated Monitoring and Alignment Services (AMAS.KOM) [85, 84] is used to perform the automatic monitoring and deviation handling. The approach of Nicolas Repp also includes reasoning mechanisms to determine the distribution of the monitoring and control units in the service eco-system.

Just as BPRules, WS-Re2Policy allows to specify actions to be performed in case of QoS deviations or SLA violations based on a simple Event-Condition-Action scheme, with a predefined set of actions like restarting services or triggering service selection for a single service or the whole business process. Although WS-Re2Policy facilitates the specification of QoS requirements for business processes, it remains unclear how these specifications interact with a concrete service selection approach. In [85], just some hints are available that WS-Re2Policy can be incorporated with the service selection of the WSQoSX system proposed by Berbner et al. [31, 30]. Here, BPRules goes beyond WS-Re2Policy as it enables specification not only of QoS constraints, but also of objective functions to be used for service selection. BPRules also allows to clearly distinguish between promised/negotiated, monitored and predicted QoS values for services within its rule definitions, which we consider as indispensable for a flexible approach for QoS management of business processes. WS-Re2Policy is also missing a number of other features of BPRules, as e.g. context-aware service selection, dynamic rule-set change at runtime and specification means to determine the instances, the QoS values of which determine the need of management actions.

4.1.2 QoSL4BP

In [22, 23], Baligand et al. propose QoSL4BP and the ORQOS platform. QoSL4BP is a domain-specific language which facilitates specification of QoS requirements for business processes and corresponding management policies in a declarative way. It is distinguished between static requirements, which can be used for the initial planning steps and do not change during runtime, and dynamic rules, which take into account the runtime behavior of the business process and the involved services. The resulting QoSL4BP models are executed on the QRQOS platform which provides support for monitoring and execution of the defined management actions.

Although QoSL4BP is not based on XML and the definition of QoS constraints and management actions is quite different from BPRules, QoSL4BP and ORQOS inspired many design rationales of BPRules and the BPR framework. One example is separation of concerns with regard to service composition logic and QoS management logic, and both languages are non-intrusive with regard to existing orchestration languages and engines. Thus, with QoSL4BP and BPRules, QoS requirements and management policies are specified separately from the service composition, and existing and well-established orchestration languages and engines, like BPEL and the Oracle BPEL engine, can be used without modifications. Baligand et al. also allow specification of QoS requirements and management actions not only for the whole business process but also for single basic and structured activities. This concept is very similar to the concept of sections in BPRules.

Nevertheless, BPRules goes clearly beyond QoSL4BP. BPRules offers much more flexibility in defining QoS constraints for the business process and the conditions that can act

as trigger for management actions. Whereas the rules of QoSL4BP in most cases just refer to the current or last instance of the business process, BPRules provides elaborate means to select a set of instances as basis for the evaluation of the trigger condition. Furthermore, QoSL4BP lacks a number of other features of BPRules, e.g. definition of objective functions and support of context-aware service selection.

4.1.3 WS-Policy4MASC

The authors of [70] propose a language called WS-Policy4MASC and a framework for adaptation of Web-Service compositions which is able to select the appropriate adaptation strategies for different classes of instances. The strategy selection is not only considering QoS dimensions but also business metrics. With BPRules and the BPR framework we intend to improve the long-term QoS behavior by selecting and replacing services. Thus, in comparison to [70] our focus is much more on service selection algorithms and on specifying rules that define when and how to replace services.

In [98], Tosic et al. describe how WS-Policy4MASC allows the specification of monitoring and control policies for the Manageable and Adaptable Services compositions (MASC) middleware [53]. WS-Policy4MASC extends WS-Policy with new types of policy assertions: *goal policy assertions*, *action policy assertions*, *utility policy assertions*, and *meta-policy assertions*. Goal policy assertions facilitate specification of requirements and/or guarantees that have to be fulfilled by the service composition, whereas action policy assertions define the management actions to be performed if a requirement or guarantee is not met. Whereas goal policies impact the monitoring performed on the service composition, actions policies drive the adaptation of the service composition. In this respect, a rule in BPRules combines goal policy assertions and action policy assertions of WS-Policy4MASC. Utility policy assertions allow to define monetary values for specific situations. Meta-policies can be used to define a set of alternative action policy assertions applicable in a situation and corresponding selection strategies.

In contrast to WS-Policy4MASC, BPRules does not allow to assign monetary values to certain situations, instead it facilitates the specification of objective functions to guide service selection. BPRules also does not support definition of alternative actions and selection strategies. In general, WS-Policy4MASC provides a quite general framework for monitoring and control with a rich set of monitoring and management capabilities, which is not only focused on QoS management. This expressivity also causes WS-Policy4MASC specifications to become really verbose, which Tosic et al. tried to mitigate by defining an according UML profile [99].

BPRules clearly concentrates on QoS monitoring and management of service compositions with service selection and replacement as the most important management action. Thus, in comparison to WS-Policy4MASC, BPRules covers many aspects in more detail, as e.g. elaborate support for selecting instances as base for the monitoring, distinction between monitored, promised and predicted QoS values, which is neglected in WS-Policy4MASC. BPRules also offers unique features like context-aware service selection. In WS-Policy4MASC this aspect is not considered at all.

4.1.4 WSCoL

Baresi et al. [27] propose the Web Service Constraint Language (WSCoL) for defining functional as well as non-functional constraints for service interactions within the workflow of a BPEL process. WSCoL is an XML-aware language, which uses WS-Policy as baseline and extends it with domain-independent support for monitoring assertions. The original WSCoL proposal focuses on defining pre- and post-conditions for basic activities in a BPEL process, like invoke, receive and pick, leading to local constraints for basic service interactions. Although WSCoL introduces only a few language concepts, the approach has to be considered quite powerful as it provides specification means for data collection as well as data analysis. For example, in WSCoL three different kinds of data collection can be specified. It allows to state if internal variables, external variables provided by invocation of other services, or historical data obtained by the monitoring framework should be taken into account when evaluating the constraints. Similar to BPRules, constraints in WSCoL can make use of universal and existential quantifiers and of aggregate functions on collected historical data, as e.g. minimum, maximum, average or sum.

In [28], Baresi et al. extend WSCoL to Timed WSCoL by introducing the concepts of scope and temporal operators. In contrast to WSCoL, where the scope of the constraints is limited to basic activities, Timed WSCoL allows elaborate specification of temporal constraints over several basic or complex activities. For this purpose, Timed WSCoL makes use of concepts from temporal logics. WSCoL as well as Timed WSCoL concentrate on monitoring constraints, whereas management actions in case of constraint violation are not in their focus. BPRules with its elaborate support for management actions like service selection goes clearly beyond monitoring. Thus, the scope of BPRules and WSCoL is quite different. Still, with the authors of [28] we share the insight that different data sources have to be considered in the evaluation of QoS conditions and also elaborate specification means with regard to collected historical data are needed. BPRules allows to distinguish between monitored, promised and predicted QoS values and provides elaborate support for instance handling.

4.1.5 AO4BPEL

AO4BPEL, proposed by Charfi and Mezini [42], is not designed with particular focus on QoS monitoring and management of service compositions. However, Charfi and Mezini recognize the shortcomings of BPEL with regard to a well-modularized and flexible specification of crosscutting concerns such as security, logging, and QoS monitoring, and identify aspect-oriented techniques as most suitable means to compensate these shortcomings. Consequently, AO4BPEL is an aspect-oriented extension for BPEL providing specification means for the main concepts of aspect-oriented programming, which are join points, point cuts and advices. Join points correspond to activities of the BPEL process and thus, point cuts refer to a set of activities which may span over one or several BPEL processes. Advices are commonly specified as sequences of BPEL activities realizing a kind of sub-workflow and can be executed by the underlying BPEL engine.

AO4BPEL does not provide support for expressing QoS requirements and constraints

and corresponding management actions at all. Still it is an interesting approach because most of the other languages in the area of QoS monitoring and management, as e.g. WS-Re2Policy [85, 84], QoSL4BP [22, 23] , WS-Policy4MASC [98] and also BPRules, just support to define which QoS parameters have to be evaluated in the conditions but not how the corresponding monitoring is done. Only WSCoL [28], [27] provides some specification means to refer to Web services for data collection. BPRules relies on the BPR framework to collect and monitor all the needed parameters. For this purpose, the BPEL process is instrumented with appropriate sensors which fire at the start and the end of activities. It could be beneficial to combine BPRules with the ideas of AO4BPEL, however this has to be investigated in future work.

4.1.6 Other Related Approaches

In [69], the authors describe an approach for preventing SLA violations by a dynamic substitution of fragments (equivalent to our sections) at runtime. Thereby the candidate fragments have to be modeled at design time. This approach can be considered as an improvement for a more dynamic substitution of a section, which we plan to adopt in our future work. However, the authors have addressed only this particular substitution aspect in [69]. With our framework, we aim to provide a comprehensive support for managing QoS of service compositions that includes monitoring but also a rich set of management actions as well as efficient service selection strategies.

The WSLA [71] language can be used to specify a contract (SLA) between the service provider and the service consumer. However, WSLA does not address the specifics of service compositions and the monitoring of certain activities or sections from the composition is not possible. The language does not offer support for controlling a service composition or for improving the QoS behavior, like it is possible with BPRules.

The WS-Policy framework provides a more general approach for specifying requirements for different entities like WSDL elements, services, message parts, operations. In comparison to WS-Policy, our BPRules language and the BPR rules are tailored to monitoring and managing QoS in service compositions.

4.2 Service Selection

4.2.1 Integer Programming

Zeng et al. describe in [113] a *QoS-aware Middleware for Web Services Composition*. For the service selection the authors describe two approaches for local and global optimization. In their global planning approach, the authors propose an Integer Programming (IP) solution, assuming that the objective function, the constraints, and the aggregation functions are linear. The multiplicative aggregation functions for availability and success rate are linearized by applying the logarithm. In this way, the service selection problem is mapped to an IP problem and standard IP solvers can be employed to find the optimal solution. However, optimization is performed for each execution path of the workflow separately. Thus, the optimal selections for the single execution paths have to be merged, in order to determine the optimized selection for

the whole workflow. For this purpose, Zeng et al. identify the most frequent path for each of the tasks in the workflow, which they call *hot path*, and use the selected service of the hot path also for the overall solution.

Another problem of the approach of Zeng et al. is the need to unfold loops to sequences. If a loop contains n cycles, each abstract service appears n times in the corresponding unfolded sequence. Thus, loops can increase the required computation time for optimization significantly. Acceptable runtime performance can only be expected for workflows with a limited number of execution paths, loops and concrete services. The approach quickly becomes unfeasible with an increasing number of tasks and concrete services.

In contrast to Zeng et al., our algorithms $OPTIM\_HWeight$ and $OPTIM\_PRO$ can also work on non-linear objective functions, do not need to consider different execution paths separately, and do not require unfolding of loops. Still, they are also applicable to problems with a larger number of abstract services and concrete services per abstract service, as shown in Section 10.1.

4.2.2 Linear Programming Relaxation and Backtracking

In [31], Berbner et al. propose an improvement of the IP approach of Zeng et al. [110]. First, they solve a linear programming (LP) relaxation of the IP problem, which results in an indication how likely a concrete service is part of the optimal solution. Afterwards a backtracking algorithm is employed to calculate the concrete services to be selected. Here, the indications of the first step are used to sort the service candidates for an abstract service and therefore to guide the backtracking algorithm. Furthermore, the number of *likely* service candidates for an abstract service also determines the order in which the abstract services are considered in the backtracking algorithm.

Berbner et al. also investigated to which extend the found selections can be improved by utilizing further heuristics. The first heuristic randomly replaces services and checks if an improvement of QoS was achieved. If an improvement was achieved the new service is used. They also tested simulated annealing as improvement strategy. However, the test results have shown that only a marginal improvement over the original solution can be expected.

Although the approach constitutes an improvement of the original IP approach of Zeng, it has the same restrictions and addresses only sequential Web service compositions and linear objective functions. With our algorithms, we are able to consider sequential and also parallel execution of activities and non-linear objective functions.

4.2.3 Genetic Approach

Canfora et al. [39] propose a genetic approach for the selection problem. Genetic algorithms are inspired by biology and use meta-heuristics in optimization problems. The genome represents the service variants that realize the service orchestration. The length of the genome is equal to the number of abstract services and each element within the corresponding array contains a reference to the list of the concrete candidate

services that may realize the abstract service. The initial population is built with random individuals. The fitness of the individuals represents their utility as a solution. It is computed based on the objective function as well as a penalty term for violated constraints. This means that those individuals not fulfilling the QoS constraints are penalized with regard to their fitness.

Multiple generations over the population are built in an iterative way by applying mutation and crossover operators. Through the mutation operator, the candidate services are varied randomly and an arbitrary concrete service is selected to realize the abstract service. The crossover operator combines service variants of different individuals. The algorithm stops when during multiple generations there is no improvement to the fitness function value.

Since the authors consider also non-linear objective functions, we implemented their approach to compare it with our heuristic algorithms (see Section 10.1). We used the same aggregation functions as Canfora did. Although Canfora et al. select the genome with a fitness function containing a penalty factor, this penalty factor does not guarantee that the individuals will fulfill the QoS constraints for all possible execution paths. By checking the QoS constraints considering the worst case of QoS values throughout multiple branches, our algorithms ensure that the QoS constraints are met for all of the execution paths. In addition, our $OPTIM\_HWeight$ and $OPTIM\_PRO$ need significantly less time than the genetic algorithm, to find the same result or even a better one.

4.2.4 Tabu Search and Hybrid Genetic Algorithms

Parejo et al. [81] investigated approaches to the service selection problem based on tabu search and hybrid genetic algorithms. Their proposal of a hybrid genetic algorithm can be considered as an amendment of the genetic approach of Canfora et al. described in the previous section. The improvement is achieved by iteratively exploring the neighborhood of each individual of the genetic algorithm and replacing the actual individual with the best or the almost best neighbor. In their original algorithm, the authors explore the complete neighborhood of an individual. However, the computational cost of this exploration reduces the diversification of the population or the number of generations if the computation time shall not be increased. Thus, the authors propose only to explore a certain percentage of the neighborhood of individuals.

In their experiments, the authors revealed that compared to its basic version the hybrid genetic algorithms perform better only with regard to small and medium size problems and that tabu search achieves improvements only for small problem instances and short run times. Our algorithms not only show good performance with small and medium size problems but are also able to outperform the genetic algorithm for large problems.

4.2.5 Heuristic Approaches

As the service selection problem addressed in this thesis is NP-hard [22], approaches which provide the global optimal solution in a deterministic manner are deemed to

become unfeasible with an increasing number of abstract services and concrete services. On the other hand, heuristic approaches, as our $OPTIM\_HWeight$ and $OPTIM\_PRO$ algorithms, can deliver results close to the optimal solution with acceptable computational costs.

In [76], Jaeger et al. evaluated different heuristic approaches including greedy selection, bottom-up approximation, discarding subsets and pattern-wise selection. In particular, pattern-wise selection obtains almost optimal QoS values. However, the runtime performance makes pattern-wise selection and discarding subsets unsuitable for a growing number of tasks. Other heuristics like greedy selection or the bottom-up approximation result in a loss of QoS up to 5% but lead to an acceptable runtime performance.

In this respect, our $OPTIM\_HWeight$ and $OPTIM\_PRO$ algorithms provide an improvement with regard to shorter runtimes than pattern-wise selection and discarding subsets, and deliver better QoS than greedy selection and bottom-up approximation. In general, the results of Jaeger et al. confirm the expectation that there is a tradeoff between obtained QoS and computational costs. The $OPTIM\_HWeight$ algorithm constitutes an instance of the $OPTIM\_S$ algorithm, which allows to fine tune this tradeoff by adjusting the parameter for the number of solution candidates propagated from one node to the next level in the workflow tree.

4.3 Context-aware Property Prediction

4.3.1 Collaborative Filtering

Already in 1994, Resnick et al. [86] applied collaborative filtering approaches in their GroupLens system to recommend net news for users. Since then collaborative filtering has been widely adopted in recommender systems [92] [64]. Here it is distinguished between user-based and item-based collaborative filtering. User-based collaborative filtering is based on the idea that a user's experience or rating with regard to a certain item is likely to be similar to the experiences/ratings of users who have made similar experiences or provided similar ratings on a number of other items in the past. Item-based collaborative filtering is based on the same assumption, but with items instead of users. This means, items which have been rated similar in the past for a number of users, are likely to receive a similar rating for the user under consideration.

4.3.2 Predictive Algorithms for Collaborative Filtering

In [37], Breese et al. analyze different approaches of user-based collaborative filtering to predict user ratings. They distinguish between memory-based methods and model-based methods. Whereas memory-based methods directly work on the collected data for user votes, the model-based methods first try to build a model of the collected data, which is then used to make predictions.

Memory-based methods typically predict the user rating as weighted sum of the ratings of other users, where the weights correspond to the similarity of the users. Thus, the

selection of the similarity measure is crucial for the performance of the algorithm. Simple vector similarity or similarity based on Pearson Correlation Coefficient are widely used measures in this context. Breese et al. have also analyzed extensions of these similarity measures with default voting, case amplification or inverse user frequency.

As candidates for model-based algorithms, Breese et al. evaluated probabilistic cluster models and Bayesian network models. For the cluster models they assume membership in an unobserved class variable, which abstracts certain user groups with similar preferences and tastes. This cluster model is built using Expectation Maximization. For the Bayesian network models, a whole Bayesian network with a node for each item in the domain is learned. The votes correspond to the states of the different nodes.

Breese et al. conclude that Bayesian network models and correlation methods outperform the probabilistic cluster models and methods based on simple vector similarity in most cases. Between Bayesian network models and correlation methods there is no clear winner, as the performance of the algorithms depends on the analyzed dataset. Apart from prediction accuracy, there are significant differences between memory-based and model-based approaches. The model-based approaches require less memory, need less time to make predictions and are likely to show better scalability. However, the analyzed probabilistic model-based approaches require a complex learning step, which is not needed for memory-based approaches at all.

In this thesis, we present a new model-based approach based on linear regression. While only relying on a very simple model-building approach, it shows good prediction accuracy. Furthermore, we show how context information can be incorporated into the prediction. However, context information is not considered at all in the approaches analyzed by Breese et al.

4.3.3 WSRec

Z. Zheng et al. describe in [113] an approach based on collaborative filtering for making QoS-aware Web service recommendations. The QoS values of the Web services are predicted for the active user based on the QoS values that were measured in the past. The presented approach relies on the combination of user-based and item-based collaborative filtering techniques using the Pearson Corellation Coefficient.

The authors bring a significant contribution to the research community by making publicly available a dataset of monitored QoS data for real world Web services [17] [112]. We also used this dataset for evaluating our CoFee module. For the collection of QoS data [113] the authors use one hundred and fifty computers in 24 countries and perform about 1.5 millions Web service invocations for 100 real world services. This is a considerable amount of monitoring work. This gives great opportunities to other researchers (like us) that do not afford evaluations at this scale because most of the real Web services perceive taxes for invocations and are not intended for research and test purposes. Therefore, we highly appreciate the work of the authors also for sharing their QoS monitored data.

With the WSRec approach, the final prediction result is calculated as a weighted sum of the user-based prediction result and the item-based prediction result, where the

weights take into account the respective prediction confidence and a manual tuning-factor λ. Prediction confidence is estimated from the similarity values found in the user-item matrix. As similarity measure the Pearson Correlation Coefficient is used. However, not all users or all items of the user-item matrix are taken into account. Only the top k users/item with highest similarity and only users/item with positive correlation are considered.

Furthermore, in their WSRec method, Zheng et al. propose a two-step approach. In the first step, a part of the missing values of the user-item matrix are predicted in order to enhance the matrix density. This enhanced user-item matrix is then used in the second step to perform the predictions for the actual users under consideration.

The WSRec method improves indeed the predictive accuracy of the pure user-based or item-based approach. Nevertheless, there are still lasting problems owing to scalability and practical suitability. These are mainly related to prediction time and scalability: the greater the matrix and the lower the matrix density, the lower the predictive accuracy and the higher the prediction times. The cause is mainly based on the lack of generalization. Our approach based on linear-regression is able to provide such a generalization. Furthermore, just as most other approaches based on collaborative filtering, the WSRec method does not exploit context information in order to improve prediction quality, as we do in this thesis.

4.3.4 Location Based Regularization

In [103], Lo et al. argue that prediction of QoS properties can benefit from the incorporation of context information. They particularly refer to the usage of location information with regard to prediction of response time and state that users which live near to each other are likely to face similar physical network conditions.

The core concept of their prediction approach is matrix factorization: the user-service matrix is factorized to a user feature space matrix and a service feature space matrix. In this respect, the method of Lo et al. can be considered as model-based collaborative filtering approach. Geographical information is incorporated as additional regularization term.

By applying matrix factorization and their location-based regularization the prediction accuracy concerning the dimension response time is improved compared to user or item-based collaborative filtering. But their approach requires manual fine tuning and is heavily affected by the density of the user-item matrix. Besides, so far only QoS dimensions have been considered which depend on the user location. The methods proposed in this thesis are not limited to geographical information. Actually, relevant context dimensions are identified by using statistical approaches as e.g. ANOVA, or are derived as part of the learning step.

4.3.5 Location-aware Memory-based Collaborative Filtering Approaches

Unfortunately, there is only limited work actually concentrating on QoS or QoE predictions for Web services incorporating context data. Still, there are some other

memory-based collaborative filtering approaches which incorporate location information.

In [95], Tang et al. present a location-aware hybrid collaborative filtering method for QoS predictions. Just like WSRec, their approach builds upon user- and item-based collaborative filtering. The main difference to WSRec is that similar neighbors are first identified via location-related information such as country and IP address. Afterwards these neighbors are searched for candidates which are also similar with respect to the Pearson Correlation Coefficient. However, this approach has a major drawback: compared to WSRec even a higher matrix density is required to find similar users or services in the location-based neighborhood. Thus, the predictive accuracy may increase for those QoS dimensions which correlate with the location, but the approach is likely to suffer from scalability issues and is highly affected by matrix sparsity.

Chen et al. [43] propose an approach which is quite similar to that of Tang et al., but they consider only the user location. Consequently, they face the same problem like Tang and his group. Both did not evaluate their algorithms with regard to the impact of matrix densities less than 10%. Considering a service registry with 1000 Web services, 10% means that each user made use from 100 Web services in average.

Also Xie et al. [108] incorporate location of the users and service providers into a memory-based collaborative filtering method. Their method is only compared to a pure user-based or item-based collaborative filtering approach and they do not specify the density of their matrices. Still, significant improvement in prediction accuracy or prediction time cannot be recognized.

In summary, all these approaches share the drawbacks of pure memory-based collaborative filtering with regard to scalability and prediction time due to the lack of generalization. Furthermore, only performance related QoS dimensions are considered which depend on location information. In this thesis, we present a simple model-based linear regression approach which shows good prediction accuracy and avoids the scalability issues. We also consider the incorporation of other context dimensions and discuss context-aware prediction of QoE.

5 The BPRules Language

In this chapter, we describe the BPRules language that was designed to manage QoS and QoE for service compositions. With BPRules, rules are defined for handling possible quality deviations. The business analyst specifies rules for the service process by stating what actions should be undertaken if specific QoS or QoE requirements are not met. Appropriately chosen rules enable a proper execution of the business process even when unforeseen problems occur (e.g. a service is not accessible). We present the main features of BPRules which we consider essential for a successful QP management of a BPEL process. We show how BPR documents and BPR rules are specified and how they are executed. First insights about the BPRules language were published in [50] and the extended BPRules language and the BPR framework were presented in [48].

5.1 Requirements

QP-management: Appropriate QoS and QoE management is responsible for the success of the business process execution. The violation of SLAs and services that perform at inadequate service levels may lead to unsatisfied customers or even to their loss. Thus, the maintenance of the services' performance at appropriate service levels, have become a major concern in the service community. A BPEL process, being a composition of other services, implies that each service from the composition performs well. When QoS or QoE problems appear, immediate actions should deal with that situation. Appropriate actions for handling the situation have to be specified by the business analyst. These actions need to be undertaken at runtime so that, if possible, undesired situations are avoided even before happening.

A set of management actions is necessary for controlling the process. From the actions set, the action for selecting and replacing new services for the service composition is essential. Replacing a service that causes performance problems with another service with good QP would improve the QP of the entire service composition.

Querying Quality Properties: Detecting quality deviations in time, assumes a permanent monitoring of the process. It is necessary to query the monitored QoS and the assigned QoE in order to trigger appropriate deviation handling. It should be possible to query QP for atomic services but also for sections and the entire composition. Querying the QP behavior of sections may give indications on possible malfunctioning parts of the process.

The business analyst, being responsible for the well-functioning of the process, needs to be informed about the quality behavior of the process but also of other services (not part of the current set of selected services) that could potentially be used for the process.

Service Provisioning: The business analyst has to specify which services should be provisioned for the service composition and what quality requirements they should fulfill. The QP values which are promised by the service providers do not always correspond to the values actually perceived by the clients. Therefore, if QP values are predicted, these values can be used so that inappropriate services would not even be selected at all. It is required to specify how service selection should be performed, based on which quality parameters (promised or predicted) and for which users group.

Specifications: The business analyst has to be able to specify the mentioned artefacts (monitoring, service provisioning, etc.) in an easy way and also separately from the BPEL description file.

5.2 The BPRules Language

The BPRules language addresses exactly these challenges and offers appropriate support for the QoS and QoE management. BPRules is a rule-based language and offers management capabilities with regard to the quality behavior of single Web services and Web service compositions. We call a *BPR rule* a rule that is specified with BPRules and is conform to the BPRules format. BPRules is intended for humans which are responsible for the QP management and it offers them comprehensive support to easily use all the features of the BPR framework without having to know its implementation details.

5.2.1 The BPR Rules

Typically, a BPR rule contains a *QP condition* and the corresponding *action* to be triggered. While the condition contains the QP constraints check, the action states what happens when undesired QP are measured. Management actions might rank from just notifying the interested parties about certain events, over starting or stopping the process to actions like selecting and replacing some services with others that provide better QP. The rules are specified in XML and the syntax is validated against the BPRules XSD schema which can be found in Chapter A.

BPR rules are grouped together into rule sets. Listing 5.1 represents the typical form of a rule set and a rule specified with BPRules.

```
1<ruleset id="rs-green">
2  <evaluate trigger="periodic"  unit="minutes">1</evaluate>
3  <rule id="rulename1">
4    <condition>
5      <!--QoS and QoE constraints -->
6    </condition>
7    <action>
8     <!--management actions from the BPR actions set-->
9    </action>
10 </rule>
11 <!-- rule specifications-->
12</ruleset>
```

Listing 5.1: Rule Set Example

A BPR rule usually contains a QP *condition* which may enclose several QP *constraints* that need to be monitored. QP constraints are described using *expressions*. The condition part of a rule might be missing in some cases (e.g. for report generation), but the rule contains exactly one $< action >$ element. The *action* part in turn may enclose several BPR management actions available in the *actions set* of BPRules. Listing 5.1 states that the rule set is evaluated every minute (in the $< evaluate >$ element), which is also the default setting.

5.2.2 Evaluation of Rules

The rule sets may be evaluated at different times, dependent on the kind of actions which need to be performed. The time when a rule set is evaluated is specified within the element $< evaluate >$ using the attribute $trigger$ (e.g. $< evaluate\ trigger =$ "$periodic$" $>$). In BPRules, we distinguish between two ways of triggering rule evaluation:

- **Evaluating rules *once*:** One possibility is to evaluate the rule set only a single time (attribute $trigger =$ "$once$"). This is usually the case, when an initialization needs to be performed in the beginning. For example, the services are selected, the process is deployed and then started.
- **Evaluating rules *periodic*:** The rules from the rule set are evaluated at a specific interval of time (attribute $trigger =$ "$periodic$"). This type of evaluation is appropriate for actions that need to be evaluated periodically. The attribute $unit$ and the value of the element are used to specify the period of time when the rule set is re-evaluated (every 1 minute, every 1 hour, etc.). For example, in a common case, the quality requirements have to be checked more often in order to detect unusual behavior (like every minute) while the reports are needed at the end of the month.

5.2.3 The BPR Document

The BPR rules are specified in BPR documents. The BPR framework is able to load and execute the rules from the BPR documents. More details about the execution of rules are presented in Chapter 9. We associate a BPR document to each of the BPEL processes that run within the BPR framework. Mainly, a valid BPR document defines the following information:

- **Sections:** for the specification of the monitored sub-orchestrations.
- **Constants:** for defining global constants that may be used throughout the BPR document (e.g. as threshold values for the quality parameters).
- **Rule sets:** containing the collection of rules sets that state how to handle QP deviations.

Figure 5.1 represents the general structure of a BPR document which is attached to a process.

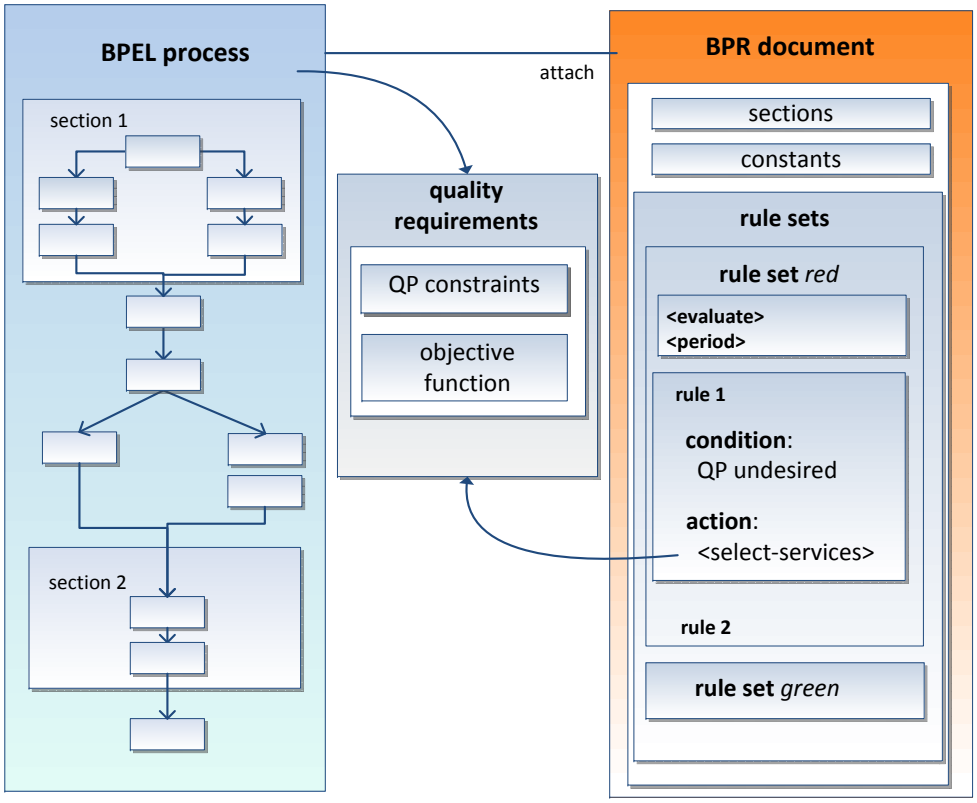

Figure 5.1: The BPR Document

5.2.3.1 Elements Overview

Figure 5.2 represents an overview of the elements from a BPR document together with their cardinalities. The $< bprules >$ element is the root element of the BPR document and it contains the $processid$ attribute that uniquely identifies the BPEL process in the BPR repository. The $processid$ corresponds also to the name of the BPEL process (which is also unique) as it was specified in the BPEL description file.

The $< bprules >$ element contains sub-elements such as $< sections >$ for defining sub-orchestrations, $< constants >$ for specifying constants, and the $< rulesets >$ element which contains one or more rule sets with the BPR rules. Checks for QP constraint violations (e.g. $responsetime < 5$) are specified as part of the $< condition >$ element using expressions, which can be linked by logical operators AND, OR and NOT. The $< service - selection >$ action contains the requirements ($< quality - requirements >$) that need to be fulfilled when new services are searched. These $< quality - requirements >$ are interpreted by the service selection module which searches for appropriate services that fulfill the requirements.

By default, the QoS and QoE constraints and requirements are applied to the entire service composition unless it is specified differently. Another possibility is to apply the constraints and requirements for a single *section* or a single *service*. For these cases the attributes $applysection$ and $applyservice$ are used inside the $< expression >$, $< select - services >$ or $< select - services - context >$ elements.

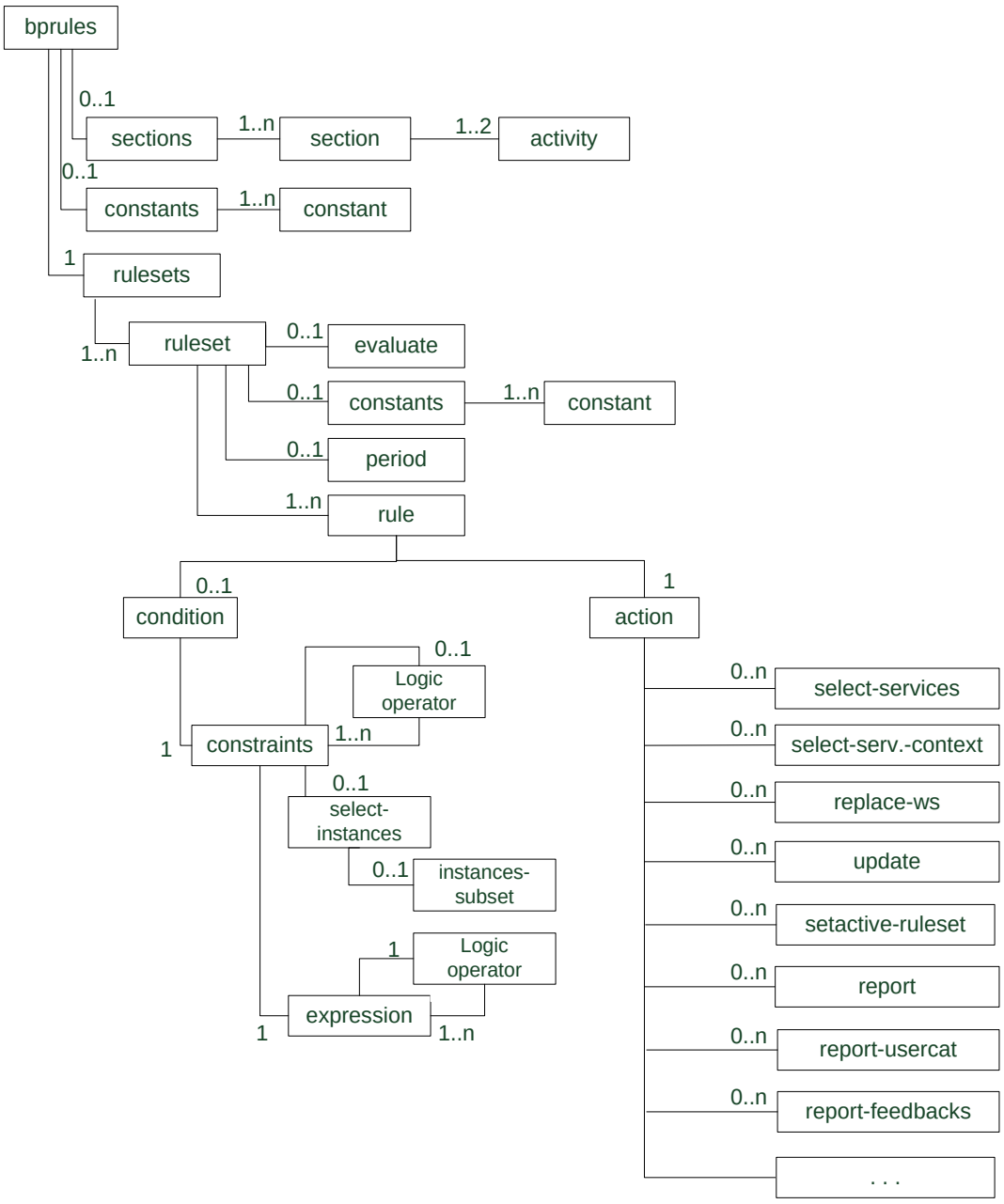

Figure 5.2: BPR Elements Overview

The exact structure of a BPR document and the descriptions of the elements are presented in Chapter A. The BPRules features and actions are explained in more detail in Section 5.3.

5.2.4 Design Rationales

We developed BPRules with the following design rationales in mind: *simplicity, expressivity, reusability* and *separation of concerns*.

- **Simplicity**

  The business analyst, who specifies the rules, is not required to have any programming skills. The BPR documents are human-readable and BPRules is *simply* structured using XML. Also the syntax for specifying *expressions* and *functions* is similar to the syntax of WSLA [71].

- **Expressivity**

  BPRules is *expressive* because it provides various features for monitoring QoS, querying QP and a comprehensive set of actions for the QoS and QoE management. BPRules offers the possibility to combine actions into more complex actions, thus giving the opportunity to build more powerful constructs. BPRules gives the possibility to specify own custom actions to extend the language.

- **Reusability**

  Reusability is supported by the possibility of reusing elements specified inside the BPR document. Several elements (for example $< section >$, $< rulesets >$, $< service - registries >$) are identified by *ids* and can be reused throughout the BPR document by simply referencing the *id*.

- **Separation of Concerns**

  We achieve *separation of concerns* by specifying rules in BPR documents which are stored separately from the business logic. Our goal was to make as few as possible modifications to the BPEL description file and not to interweave monitoring artifacts into the process logic. Thus, the only change we perform to the BPEL file is updating the URLs of the services with those of the proxies. The proxies are supposed to measure the QoS of the services and to facilitate the service replacement.

5.3 BPRules Features

We designed BPRules with several features that we envision as mandatory for the management of QoS and QoE. In the following, we present the features of BPRules:

1. **Flexible QP Data Retrieval**

   BPRules offers the possibility to query the monitored QoS, the assigned QoE and the predicted QP values of services, sections and the entire process. QP are checked for compliance inside the $< condition >$ part and they may be queried for the entire process, its sections or single services. In case of the process or a section, the QP are computed for each process instance (respectively its section) by applying the *measurement algorithm* (see Section 6.4) which aggregates the QP of the process instance (respectively the section) from the QP values of the atomic services.

   Interpreting and processing the QoS and QoE data may be dependent on the period of time when the execution of the process took place. For example, past quality behavior may be retrieved for a report or analysis while current quality behavior has to be monitored to remediate malfunctions by updating the process

at runtime. With BPRules, we can specify rule sets that consider process instances from a specific period of time. For instance, the period may be a time interval in the past or might range from a moment in the past till the actual moment. It can be specified as an absolute time interval (with a begin and end date/time) or as a relative period in the form: *last x time-unit* (e.g.: last 10 hours, last 2 months). Usually, during a period of time there are multiple instances to be considered. In the default case, when QP values are queried, the average QP values of the instances are considered. It is also possible to query the minimal or maximal QP values of the instances. The attribute $applyfunction$ (belonging to the $< qualityParam >$ element) may be used for this purpose.

2. **Section Control**

   For a better control and detection of QoS or QoE deviations, we can divide the process into several parts, which we call *sections*. This provides the possibility to group multiple activities (e.g. all activities inside a *flow*, or a *sequence*) into a single manageable part. We define a section by referring to a structured activity with its nested sub-activities. Another way for specifying a section is by considering all activities between a *start* and an *end* activity inside a *sequence*.

   As an example, in our bookshop process (described in Section 2.3.1.1) we define a section along with its QP requirements. The section consists of several activities, involving the invocation of the *Distributor Service* for checking if the book is available, then choosing the book with the minimal price and buying it. In this section, a low *response time* and *cost* is required. Listing 5.2 shows the QoS requirements for the *distributor section*. The example listings presented in the thesis contain commentaries because of the lengthy XML. However, when BPR documents are specified, these have to contain elements conforming to the BPRules format. When the QoS of the section reaches some risky values (*response time* > 3 or *cost* > 0.25) then the distributor service will be replaced with another one that provides better QoS and whose WSDL description is available at the specified URL. The $< replace - ws >$ action performs a manual service replacement. The *expression* element contains the QoS constraints which can be linked by the logical operators *AND*, *OR* and *NOT* to form more complex conditions. We can specify different QP requirements for different sections.

   With BPRules it is also possible to establish relations between the QP of different sections and the entire process. For example, a query like this is possible: the *response time* from the *distributor section* is less than $1/2$ of the *response time* of the *bookshop process*. Thus the business analyst may be informed if the distributor section consumes too much time in comparison to the entire response time of the process, which can be a good indication for a malfunction in the distributor section. Furthermore, this kind of QP conditions may ensure keeping an appropriate proportion between the QP parameters between process sections and the process.

```
<ruleset id="distributor-normal">
 <period> <!-- time interval --> </period>
 <rule>
  <condition>
    <constraints>
      <instances-subset applyfunction="MIN">10%</instances-subset>
      <expression applysection="distributorsection">
       <or>
          <!-- response time > 3 or cost > 0.25 -->
       </or>
      </expression>
    </constraints>
  </condition>
  <action> <!--from the BPR management actions set -->
    <replace-ws>
      <service name="bookshop/DistributorService">
        <wsdl-url>
          <!--url to the WSDL of the new service -->
        </wsdl-url>
      </service>
    </replace-ws>
  </action>
 </rule>
</ruleset>
```

Listing 5.2: A BPR Rule Example for a Section

3. **Instance-set Handling**

   With BPRules, we can specify a certain set of instances to which the QP constraints apply. This is an important task since, for example, situations when 2% of the instances failed or over 20% of the instances failed need to be treated differently. While the first case could be tolerable, the second case needs to be addressed adequately. In Listing 5.3 it is stated that if *minimum* 20% of the instances failed then a *select services action* should be undertaken to replace the services with others that are predicted to have better QP. The *select services action* performs an automatic service replacement of all the services in the process with services found by the service selection module.

   As described in Listing 5.3, the state of the instances can be queried with the $< property-check >$ element. We distinguish between states like FAULTED for instances with activities that have thrown an exception, RUNNING for instances with activities that are still executed, and COMPLETED for instances where all of the activities are successfully completed. For querying the size of the set of instances that fulfill or violate the QP constraints, BPRules offers a set of functions: FORALL targeting all the instances in the set, EXISTS for at least one instance, MIN nr(%), MAX nr(%), EQUALS nr(%) to refer to a percentage of the total number of instances. With these functions, BPRules makes it possible to trigger appropriate actions according to the runtime behavior of the instances.

```
<rule id="selectAll">
  <condition>
    <constraints>
      <instances-subset function="MIN">20%</instances-subset>
      <expression>
        <property-check select="state">FAULTED</property-check>
      </expression>
    </constraints>
 </condition>
 <action>
  <select-services methodClass="ALG.OptPRO" qualityValues="Pred">
     <service-registries>
      <wsdl-url>http://atlantis:8081/sregistry?wsdl</wsdl-url>
      <!-- other registry URLs to be searched -->
     </service-registries>
     <quality-requirements>
       <expression>
          <and>
            <expression>
              <predicate type="Less">
                <qualityParam>responsetime</qualityParam>
                <Value>3</Value>
              </predicate>
            </expression>
            <expression>
              <predicate type="Greater">
                <qualityParam>availability</qualityParam>
                <Value>0.95</Value>
              </predicate>
            </expression>
            <!-- similar expression for cost < 0.3 -->
          </and>
        </expression>
        <objective-function type="MAX">
          <operation type="DIV">
            <qualityParam>availability</qualityParam>
            <operation type="ADD">
                <qualityParam>responsetime</qualityParam>
                <operation type="MUL">
                  <Value>2</Value>
                  <qualityParam>cost</qualityParam>
                </operation>
            </operation>
          </operation>
        </objective-function>
     </quality-requirements>
  </select-services>
 </action>
</rule>
```

Listing 5.3: A Service Selection Example

4. **Flexible Service Selection**

   BPRules provides extra flexibility for the *selection of services*. The *select-services action* from BPRules may be employed for the entire process or only for some of its sections. It triggers a selection algorithm to search for services in specified service registries and to replace the old services in the process with new ones that provide better QP. The selection algorithms receive as input the quality requirements of the process or the section. The quality requirements consist of the QP constraints and an objective function to be optimized.

   The selection algorithms compute estimations of the QP values for the process (or section) and choose services that may realize the process (or section). The estimated QP are computed by using the *QP estimation algorithm* described in Chapter 6 considering either the promised or the predicted QP values. Therefore, we distinguish between two kinds of estimated QP values:

   - **The estimated-promised QP (Prom)**: a QP estimation of the process (or section) by considering the QP values promised by the service provider for the atomic services
   - **The estimated-predicted QP (Pred)**: a QP estimation of the process (or section) by considering the QP values predicted by the *CoFee Predictor* for the atomic services

   The estimated promised or predicted QP values may be retrieved for single services as well. The estimated QP values are queried using the $qualityValues$ attribute (of the $< select-services >$ or $< select-services-context >$ element) which is set to "$Prom$" for the estimated-promised QP and to "$Pred$" for the estimated-predicted QP, respectively.

   Listing 5.3 represents an example of a *select-services action* defined with BPRules. We consider the QoS dimensions *availability* (avail), *response time* (resp), and *cost*. A set of services is searched for the entire composition and the solution has to fulfill the following QoS constraints and maximize the objective function:

$$f_{obj}(Q) = \frac{avail}{resp + 2 \cdot cost}, \text{maximize } f_{obj}(Q) \tag{5.1}$$

$$resp < 3,\ avail > 0.95,\ cost < 0.3 \tag{5.2}$$

   In contrast to other works [110], our selection strategies are also able to deal with non-linear objective functions and aggregation functions. Our selection action is *customizable with regard to the selection method* (selection algorithm). For example, when searching a few services, like within a section, a trivial brute-force search is sufficient, while in a search at runtime that involves many services (e.g. for the entire process), a more advanced and rapid search is needed. For this purpose, the BPR framework provides three algorithms, $OPTIM\_S$, $OPTIM\_PRO$ and $OPTIM\_HWeight$, that can be employed for the selection of services. $OPTIM\_PRO$ and $OPTIM\_HWeight$ are iterative algorithms that try to improve the found solution with each iteration step. The selection algorithms are described in Chapter 7.

We assume that a service registry is exposed as a Web service and accessible via a URL (Listing 5.3: line 13). The *methodClass* attribute (line 11) is used to specify which of the selection algorithms is employed. In the listing example, the $OPTIM\_PRO$ algorithm is called.

There may be situations when certain services are preferred and it is not desired to replace them during service selection. In this case, we may declare these services in BPRules as *fix*, which means that they will not be searched nor replaced during the selection procedure.

5. **Context-aware Service Selection**

   BPRules offers another type of service selection, by making a personalized service selection for clients that have certain context data. Since QoE and QoS parameters might be in correlation with the context data of the users, a context-aware service selection would offer the possibility to create a service composition personalized for its clients. This kind of context-aware service selection is supported by the $< select - services - context >$ action. A more detailed description of this action and how it is handled is presented in Section 8.3.4.

   For example, the business analyst desires to create a personalized bookshop process for users that live in Germany and work in the medical sector. Therefore, he needs to define the user category which is characterized by certain context parameters (defined as $< context\ param = "name" > value < /context >$) specified inside the $< category >$ element. He modifies the $< select - services >$ action (from Listing 5.3) into a $< select - services - context >$ action described in Listing 5.4. We assume that the context parameters *location* and *profession* of the users are correlated to the assigned ratings. The business analyst specifies also a constraint for QoE (QoE > 4) and introduces this dimension in the objective function to be maximized. Therefore, the service selection module creates a context-aware composition for this user category where as *Distributor Service* a more specialized service (the *MedDistributor Service*) is integrated, which offers a large collection of medical books written in German.

```
1 <select-services-context methodClass="ALG.OptPRO">
2   <quality-requirements>
3      <!-- qoe > 4, resp < 3; avail > 0.95 ; cost < 0.3 -->
4      <!-- fobj = (qoe + avail) / (resp + 2 * cost) -->
5   </quality-requirements>
6   <list-categories>
7     <category name="usercat">
8        <context param="user-location">Germany</context>
9        <context param="user-profession">medical</context>
10     </category>
11  </list-categories>
12 </select-services-context>
```

Listing 5.4: Context-aware Service Selection

6. **The BPR Management Actions Set**

   BPRules offers several management actions which we divided into four categories: actions for *controlling the BPEL process*, actions that are meant to *improve*

the quality of the process behavior, actions which offer *information about the quality behavior*, actions for *retrieving information from CoFee*. Table 5.1 lists the management actions from the four categories offered by BPRules.

a) **Actions for Controlling the BPEL Process**

The actions from this category offer support for controlling the process and its instances. With the $< deploy >$ action, the process is installed from the BPR repository on the BPEL engine, being ready to receive requests for processing. The inverse action is the $< undeploy >$ action. It uninstalls the process from the BPEL engine. However, the process description remains in the BPR repository.

Actions like $< start >$, $< stop >$, $< deploy >$, $< select - services >$, etc. trigger a change in the process state and the process goes into the *managed state*. When the process is in the managed state, other actions that cause state changes may be triggered only sequentially in order to avoid process inconsistencies.

Several actions are provided to control process instances, offering the possibility to stop instances (with $< stop - instances >$), to resume the stopped instances (with $< resume - instances >$) and to cancel instances (with $< cancel - instances >$).

b) **Actions for Improving the Quality of the Process Behavior**

The actions from this category are meant to improve the quality of the process behavior by replacing one or more services with other services that provide better quality properties. The $< replace - ws >$ action allows a manual service replacement where the current service is replaced with the desired one, which WSDL is available at a certain *url*. This action assumes that the business analyst exactly knows which concrete service he wants to integrate into the BPEL process. While the $< replace - ws >$ action is intended for a manual service replacement controlled by the business analyst, the actions $< select - services >$ and $< select - services - context >$ target an automatic service replacement controlled by the BPR framework.

If significant deviations from the desired quality behavior of the process are measured or if several services within a section become unavailable, it may be required to modify the structure of the affected section or of the entire process. For example, it could be useful to fall back to a simplified version of the BPEL process only providing the essential parts of the intended workflow. This kind of structural adaptation is supported by the $< update >$ action, which overwrites the process description file with another file from a given path.

We may *activate* or *deactivate* rule sets at runtime. Active rule sets are those rule sets which are executed, while inactive rule sets are temporarily ignored. We may use the various rule sets for different alarm states analogously to a traffic light system. For example, if the process behaves well, then the active rules may only inform the interested parties about the behavior. In contrast, if the QP of the process gets worse another rule set could be activated with

rules that have more impact on the process, e.g. replacing one or several services. In this way we may adapt the rule sets dynamically at runtime, according to the behavior of the process.

Actions for controlling the BPEL process	
deploy/undeploy	Deploys/undeploys the process identified by a certain *process id* or given by the specified *path*. The process is installed/uninstalled on the BPEL engine.
stop	Stops the process identified by the *process id* and changes its state. All the process instances of this process are stopped. All the requests that are received while the process is stopped are stored into a *request-queue*.
start	Starts the process identified by the *process id*. It changes the state of the process into 'ready', meaning that the process is able to receive requests. New process instances are started for the requests from the *request-queue* if the waiting time in the queue didn't exceed the given threshold (*timeout*).
stop-instances	Stops a set of process instances that started within a given time interval.
resume-instances	Resumes a set of process instances that were previously stopped.
cancel-instances	Cancels a set of process instances that are currently running. It removes the requests from the *request-queue*. The clients receive an exception after cancelling the instances.
Actions for improving the quality of the process behavior	
update	Updates the BPEL process description (or section) from the specified *path*.
replace-ws	Replaces the Web service that realizes a given *abstract service* with a new *concrete service* (or replaces an entire list of services). The *URL* of the WSDL of the new concrete service has to be specified.
select-services	Selects services with better quality properties from the specified service registries and replaces the Web services for the specified section or abstract service, or for the entire process.
select-services-context	Creates context-aware service compositions by selecting appropriate services for the specified user categories.
setactive-ruleset	Activates or deactivates the rule set identified by an ID.
Actions for retrieving information about the process behavior	
report	Makes a report about all the monitored artifacts: the configurations of the BPEL process, the average QoS and QoE values measured for the configurations, the list of exceptions and events encountered during a given time period.
report-rules	Makes a report with the rules that were triggered for a process during a given time period. The report can be done for all rule sets or only for the specified rule sets.
report-error	Makes a report with the exceptions and errors that were encountered during process execution for a given time period.

throw-event	Generates an event and informs the subscribers. Different kinds of events may be triggered, e.g. when a rule is executed, when a rule set is activated or a custom event.
throw-exception	Generates an exception and informs the subscribers (e.g. further propagation of a BPEL exception).
custom action	An interested party may implement a customized action for its own specific needs. Therefore the *path* to the *class* file that implements the corresponding interface from the BPR framework has to be provided.
Actions for retrieving information from CoFee	
report-usercat	Makes a report that contains a list of the context-aware processes that were created for a process. It contains the number of requests delegated to each of the context-aware processes.
report-feedbacks	Makes a report with the feedbacks (including QoS and QoE values) that were collected by CoFee for the entire BPEL process and for all contained Web services during a given time period.

Table 5.1: Management Actions Set

c) **Actions for Retrieving Information about the Process Behavior**

With BPRules, interested parties may be informed about the process behavior during its execution, using the $< throw-event >$ or $< throw-exception >$ actions. BPRules offers different reports for analyzing the process behavior during a longer period of time. Several types of reports are supported by BPRules: a regular report ($< report >$), a report about the rules ($< report-rules >$), a report about malfunctions ($< report-error >$), a report about the feedbacks ($< report-feedbacks >$) and a report about the personalized processes created for different user categories ($< report-usercat >$). The regular report contains the configurations (the pairs of abstract/ concrete services) with the average QoS and QoE values and all the monitored artifacts (events and exceptions). The *rules report* includes an overview of the triggered rules, and the *error report* contains everything that went wrong (e.g. exceptions, errors). The reports contain information about the process or the rules executed during a specific period of time (specified as $< period >$ inside the $< report >$ element). All these reports deliver a good picture of the happenings to the business analyst. In the *rules report* the business analyst may see which rules were executed and this helps him for future rule specifications.

d) **Actions for Retrieving Information from CoFee**

CoFee, as a module of the BPR framework, provides valuable information about how users perceived the services and what has to be expected for the future. Therefore, BPRules provides the two actions $< report-usercat >$ and $< report-feedback >$ for retrieving information from CoFee. These actions may be used by the business analyst or other interested persons that desire to be informed of the feedbacks that the services received. By creating a report with the $< report-usercat >$ action for a process, the business

analyst receives a list with the number of requests that were delegated to each of the context-aware processes that were created for that process. This information helps to find out if the context-aware processes are specified appropriately. For example, if a context-aware process receives only few requests, this process could be removed from the BPR repository.

The $< report - feedbacks >$ action lists all the feedbacks (the QoS and QoE values) that were retrieved by CoFee for a process or service during a period of time. If the feedbacks are queried for a process then the feedbacks are listed for each service of the process and for all the process configurations in the specified period of time. The average QoE is also listed and it reflects whether the users are satisfied or not with the process.

The actions described in Table 5.1 are atomic actions. Usually, for managing the process properly, several actions need to be triggered. For this purpose, the atomic actions can be composed to so-called complex actions.

7. **Advanced Control and Decision Support for the Business Analyst**

   In the BPR framework, the business analyst is responsible for supervising the process behavior and ensuring its proper execution. The BPRules language was created for the business analyst and other persons with similar intentions to support them in accomplishing their goals. The business analyst is in charge of specifying the BPR rules. It is required that the analyst holds a fully control over the process and that he is always able to intervene in the process execution whenever it is necessary. For this purpose, the BPR framework offers adequate support for monitoring and managing the QoS and QoE of the business process. BPRules offers a fine balance between automatic and manual control.

   The different kind of reports supported by BPRules provide valuable information for the business analyst about how the process behaved (e.g. with $< report >$, $< report - error >$) and also show how users have perceived the process and the integrated services (with $< report - feedbacks >$). The rules report (with $< report - rules >$) gives an overview of the triggered rules and shows whether there are rules that were not triggered at all and which may be removed from the BPR document. The business analyst may find out whether he has properly chosen the user categories by looking at the report generated with the $< report - usercat >$ action.

   The business analyst is able to specify his own complex actions which may be reused. The possibilities of composing actions, defining custom actions, or applying manual actions (see the $< replace - ws >$, $< fix >$ declarations) deliver an advanced *control and decision support* for the business analyst.

5.4 Summary and Discussion

QoS and QoE concerns are an important topic for service compositions because the malfunction of one single service inside the composition may spoil the entire behavior of the process. The business analyst is supposed to ensure the well-functioning of the

service composition by specifying proper BPR rules. The main part of a BPR rule is the QP condition upon which the actions are triggered. Thus, specifying adequate reactions to possible quality deviations is an important task for a proper process execution. In order to specify rules (conditions and actions) in the right way, it is helpful to analyze past process executions. These executions provide good indications about measured QoS and QoE values and possible deviations. The BPRules language offers a number of reports (e.g. $< report >$,$< report - rules >$, $< report - feedbacks >$) that may be used by the business analyst to analyze the behavior of the process. Furthermore, BPRules provides several features which are essential for an advanced QP monitoring and management of service compositions, as e.g. *section control* for monitoring the process within sections, *instance-set-handling* for querying the QP for a set of process instances, *service selection* for selecting appropriate services for the process with different algorithms, *context-aware service selection* for creating personalized processes for different user categories characterized by certain context properties.

In the BPR framework, the business analyst has to ensure that the specified rules are not contradictory. One possibility to deal with contradictory rules is to automatically resolve the conflicts. The authors of [96] propose in their architecture a *Policy Conflict Resolution* module based on business metrics.

6 QoS Monitoring and Aggregation

In this chapter, we describe the challenges that QoS monitoring implies and show our solutions to these challenges. We propose the *QoS measurement algorithm* for computing the QoS of a process instance. It can be applied for running and for completed instances. Further, we show how the QoS can be estimated for a process with the *QoS estimation algorithm* that is based on the approach of Canfora et al. [39]. We compare the measurement algorithm and the estimation algorithm, highlight the differences and discuss their usage. The proposed monitoring approach was published in [47].

6.1 Requirements

A Service-Oriented Architecture is a dynamic environment where services and respectively partners are continuously changing. We can describe the interaction between these services with BPEL, but BPEL does not include extensions allowing us to monitor or to ensure the performance of the services or the process. A SOA promotes the ability for flexibility and change, but this is not possible for the assessment of QoS related issues. At any time, new QoS dimensions like cost and throughput have to be introduced, measured and aggregated in order to allow a suitable evaluation for performance. We designed a flexible approach, where QoS parameters can easily be considered and aggregated with minimum effort on manual administration and no effort spent by the business analyst. In this chapter, we tackle the following challenges:

Automated Deployment: A business process includes a set of activities in order to invoke Web services, and each activity performs at a certain QoS. In several research studies (e.g. [26], [25]), the BPEL process description is interweaved with comments and extra activities are inserted into the BPEL process. These artifacts are used to define the required QoS parameters, its monitoring sources and their aggregation functions. Thus, every time the process description changes its behavior or partner services, the process architect has to adjust the whole QoS assessment artifacts. Our goal is not to alter the process description with artifacts for process monitoring.

Aggregation Functions and QoS Parameters: The measurement of QoS for single Web services is different from the QoS computation of business processes, since processes additionally consist of different activities such as if-conditions, loops and parallel invocations of Web services. This is why the measurement of QoS in business processes needs to be treated differently as for Web services. The QoS value of a business process is computed out of the QoS values of the building blocks inside the process.

Usually, the QoS requirements and implicitly the corresponding QoS measurements for business processes vary over time. Thus, the QoS monitoring and measurement

needs to be done as flexible as possible. For example, the business analyst must ensure a certain response time for his business process. If, for some reason, the response time of the process is not the expected one, the business analyst is in charge of analyzing the bad performance of the process. As the response time may be compromised by the throughput, the analyst may also want to introduce the new QoS dimension throughput in the monitoring. In our approach, we introduce a generic QoS assessment algorithm where we must only provide a set of aggregation functions in order to make it work with newly introduced QoS parameters.

Process and Section Measurement: The business analyst should be informed about what sections of the business process may cause problems in the behavior of the entire process. Therefore, performing measurements and monitoring in a section is important and we need to specify QoS requirements and manage the process within sections.

6.2 Service Composition Representation

In our framework, the service compositions are represented as trees. Before the monitoring phase, the BPEL description files are translated into BPEL trees. The BPEL tree structure is necessary for the QoS monitoring and also for the service selection.

The BPEL activities are represented as tree nodes and the tree structure is built from the BPEL XML file structure. The BPEL trees contain the types of elements which are relevant for the execution of the BPEL process, but do not contain nodes for `partnerLinks`, for instance.

In our model, we divide all BPEL elements relevant in the context of QoS computation into two classes:

1. the set of simple element types $S = \{$`receive`, `reply`, `invoke`, `assign`, `throw`, `wait`, ...$\}$ and
2. the set of complex element types C which are used for structuring the control flow, as e.g. `sequence`, `flow`, `if`, `while`, and `foreach`.

Thus, the BPEL activity types are members of the joint set $T = S \cup C$. The instances of a simple element contribute directly to the quality of service of the overall process. They do not contain any child elements from T. Complex elements, on the other hand, may contain arbitrary other complex or simple elements. They specify how these elements are to be executed and their QoS values can be computed by aggregating the QoS of their children.

6.3 QoS Aggregation

In our framework, we measure the QoS for the atomic services and also for the service composition. The QoS values of the service composition (or section) are computed out of the QoS values of the services and activities that are part of the composition. In the BPR framework, we use two kinds of aggregation algorithms: our *measurement algorithm* for the QoS assessment of the process and the *estimation algorithm* as

proposed by [39] for the QoS estimation. Our measurement algorithm aggregates the QoS for each process instance, considering the instance QoS for the atomic services. In comparison, the estimation algorithm considers the average QoS behavior of the atomic services that was monitored during a longer period of time.

The two aggregation algorithms are applied for different purposes. The measurement algorithm aggregates the measured QoS properties of past executions or currently running process executions. It calculates the QoS of the process / or section that was measured during runtime.

In comparison, the estimation algorithm gives an estimation of future QoS behavior of the process, if potential service candidates are chosen. The algorithm is based on the probabilities for the different execution paths of the process. Here, the probabilities are provided by the process designer or derived by monitoring of past executions. Naturally, for the service selection problem the estimation algorithm has to be used.

6.4 The QoS Measurement Algorithm

The QoS measurement algorithm computes the QoS of a process instance and uses a BPEL tree as described in Section 6.2. Each node $qnode$ of the tree has the same structure and contains:

1. the type $qnode.elem \in T$ of the element representing the node,
2. a unique identifier $qnode.id1 \in I1$ of the element in the tree,
3. the list with m quality dimensions $qnode.qDimensions \subseteq Q$ which are monitored or aggregated for this node,
4. the map $qnode.Value$ holding a value $qnode.Value(q)$ for each of the quality dimensions $q \in qnode.qDimensions$ monitored for the element itself,
5. a map $qnode.ChildrenValues \in D^{*\times n}$ which can hold the corresponding QoS values of the n children of the node; we need this map due to the propagation nature of our algorithm, and
6. a reference $qnode.parent$ to the parent node of $qnode$ (or `null` if $qnode$ is the root node of the tree).

Both $qnode.ChildrenValues$ and $qnode.Value$ are initially empty and remain empty in the BPEL tree. For each instance of the business process, our monitoring system creates a new copy of the BPEL tree. In these copies, $qnode.ChildrenValues$ and $qnode.Value$ are filled in by the system. The result of the quality measurement of a process instance is then a tree which contains the QoS values for each element of the business process in the field $qnode.Value$ of the corresponding node $qnode$. We furthermore define the function `getQNode`$(tree, id1)$ which returns the node $qnode \in tree$ with $qnode.id1 = id1$ (or `null` if such a node does not exist in $tree$).

We will call a node $qnode$ a simple node if the element of the node has a simple element type, i.e. $qnode.elem \in S$. Analogously, we call a node $qnode$ a complex node if the element of the node has a complex element type, i.e. $qnode.elem \in C$ (see the

definitions of simple and complex element types in Section 6.2). Since simple elements cannot contain other elements, simple nodes are the leaves of the process trees.

For each element *elem* in a BPEL process specification, a unique identifier $elem.id1 \in I1$, $I1 \subset \mathbb{N}$ will be assigned in the initialization phase of our system. The identifiers *id1* are selected so that they enumerate the nodes in the BPEL tree in the order as they would be visited in a depth-first search. We furthermore assume that *each single execution* of *elem* has an identifier $id2 \in I2$, $I2 \subset \mathbb{N}$ unique in the current process instance. At begin of the execution of a process instance *id2* is set to 1. Each time a node is reached in the execution of a process instance, the tuple $(id1, id2)$ is stored for that process instance and *id2* is increased by one. Please note that at this point the QoS value for the node is not known; it is available when the execution of the node has been finished.

The quality dimensions q which can be measured in our system, are subsumed in the set Q. One example for such a set Q could be $\{\texttt{responsetime}, \texttt{availability}, \texttt{cost}, \texttt{throughput}\}$. For each quality dimension q, there exists a domain d_q which defines the set of possible values of this QoS feature. For $q = \texttt{cost} \in Q$, $d_{\texttt{cost}}$ would be $\{x | x \in \mathbb{R}^+\}$, for instance. We define the domain D as the union of all the domains d_q. The computation of the actual QoS values in our model is based on two functions:

1. $f_{value} : Q \times I2 \mapsto D$ which determines the QoS value of a single invocation of a simple element. It represents the instance QoS (previously defined in Section 2.5.1). When measuring, for example, availability, the QoS value is either 0 or 1, depending if the activity was detected as available or not.
2. $f_{agg} : Q \times C \times D^* \mapsto D$ aggregating all QoS values of the elements nested inside a complex element (where D^* is the set of spaces of vectors of arbitrary dimensionalities over the quality domains).

Whereas $f_{value}(\texttt{cost}, 9)$ returns the single value from $d_{\texttt{cost}}$ which resulted from the invocation of a simple element with $id2 = 9 \in I2$, we could define $f_{agg}(\texttt{cost}, \texttt{sequence}, X)$ as $\sum_{i=1}^{n} \vec{x}_i$, where $X = (x_1, x_2, \ldots, x_n)$ is a vector in $d^*_{\texttt{cost}}$ and n would be the number of elements in this vector. For $X = (0.01, 0.03, 0.08)$, $f_{agg}(\texttt{cost}, \texttt{sequence}, X)$ evaluates to 0.12, for instance.

In Table 6.1, a set of such aggregation functions is listed for representative quality dimensions. We adapted the aggregation formulas from [39] to our approach. Since [39] considers a stochastic model and we perform aggregations on running or completed instances, we take into account only the actually executed path of a switch and have available the quality values of each of the iterations of a loop. As can be seen in the table, the aggregation functions for the dimensions response time and cost are additive while for reliability and availability, the functions are multiplicative.

In the following, we present the generic algorithm for the QoS computation of the business process and its sections. We therefore assume that the values of the f_{value}-function for the simple elements within the process are known. The generic algorithm computes the QoS value of the entire business process and/or its sections. The QoS aggregation of the BPEL process is done in several steps and there are two possible ways for QoS aggregation: A) aggregation on stored monitored data and B) live aggregation. Our algorithm is applicable in both scenarios, QoS aggregation during runtime and also post-processing after the processes have finished their execution.

QoS Dimension	sequence	flow	loop	switch
response time(r)	$\sum_{i=1}^{n} r_i$	$\max_{i \in 1..n} \{r_i\}$	$\sum_{i=1}^{n} r_i$	r_i
cost (c)	$\sum_{i=1}^{n} c_i$	$\sum_{i=1}^{n} c_i$	$\sum_{i=1}^{n} c_i$	c_i
availability (a)	$\prod_{i=1}^{n} a_i$	$\prod_{i=1}^{n} a_i$	$\prod_{i=1}^{n} a_i$	a_i
reliability (l)	$\prod_{i=1}^{n} l_i$	$\prod_{i=1}^{n} l_i$	$\prod_{i=1}^{n} l_i$	l_i

Table 6.1: Aggregation Functions for the QoS Measurement

Monitoring a Running BPEL Process

While the business process is running, our monitoring system records a list $execution$ of records $execElem$, each holding

1. the unique identifier $execElem.id1 \in I1$ of the element which was invoked. This identifier is equal to the identifier of the sensor that was attached to this activity.
2. the unique identifier $execElem.id2 \in I2$ of the invocation itself.
3. the measured quality of service values $execElem.Value$ which provide the results of the f_{value}-function.

   $(execElem.Value(q) = f_{value}(q, execElem.id2)\ \forall q \in Q)$

for a single invocation of an element in the BPEL tree. While the process is running, whenever an activity corresponding to a node in the process tree is finished, a new $execElem$ record is added to the execution list. Whereas the identifier *id2* corresponds to the order, in which the nodes of the BPEL tree have been visited while executing the process instance, the index of the execution list indicates the order in which processing of the nodes of the BPEL tree has been completed.

The generic algorithm provides as a result the aggregated values for m QoS dimensions of an execution path of a BPEL tree or its sections. As input, the algorithm expects the execution list $execution$ and a copy $tree$ of the BPEL tree. The generic algorithm listed below can be applied for any type of QoS dimension if suitable aggregation functions are provided.

The QoS values of the simple elements are determined considering the data obtained by monitoring from the service proxies or the sensors. These QoS values of the simple elements are updated with every execution of the activity.

The QoS values $qnode.Value$ of a complex node $qnode$ are computed from the values of its direct children in the tree. By applying the aggregation functions f_{agg} on the QoS values of the children nodes, we obtain the QoS values of $qnode$. The values of simple nodes are known from the $execElem$-records and given by the value of the f_{value}-function. The algorithm (see Figure 6.1) starts with traversing the list $execution$ of executed activities. Each record $execElem \in execution$ stands for a completed activity. Since the QoS values of an activity can be computed in the moment the activity is finished, each record allows us to derive a set of QoS values. In the case of an

Algorithm QoS measurement

```
1: Procedure aggregateQoS(execution, tree)
2: //computes the QoS of a process instance from the QoS of the simple
   //elements
3: begin
4:  // analyze the complete execution list
6:  for i ←1 to execution.length do
7:     node = getQNode(tree, execution[i].id1)
8:     foreach q in node.qDimensions do
9:        if node.elem in C then
10:          // the node is a complex node
11:         node.Value(q) ← fagg(q, node.elem, node.ChildrenValues(q))
12:       else
13:          // the node is a simple node and fvalue is
             //equivalent to execElem.Value
14:         node.Value(q) ← fvalue(q, execution[i].id2)
15:      endif
16:      //propagate this QoS value of this node to the parent node
17:      addQToChildrenValuesOfParent(node.parent,q,node.Value(q))
18:   endforeach
19: endfor
20: end
```

Figure 6.1: QoS Measurement Algorithm

$execElem$ which denotes completion of an activity belonging to a simple node, the QoS values are the data directly stored in $execElem$ corresponding to the f_{value} function. If $execElem$ belongs to a complex node, its occurrence means that the QoS of this node can be aggregated from its child nodes since an activity can only terminate after all of its children have terminated. In both cases, the new QoS values are propagated to the parent node. In the final step, the root element is the last one that is processed. Its QoS values represent also the QoS values of the entire process computed for an instance.

Because of the propagation nature, the steps 3 to 9 of the algorithm can also be executed online while the process is running. In other words, the quality of service of the process tree can be built on the fly. If this is done, the algorithm is particularly useful for components which supervise or enforce policies such as Service Level Agreements (SLA) or for management components like our BPR framework for business process management.

6.4.1 Example

Figure 6.2 represents an example of a BPEL process execution. It is an example of QoS aggregation for *response time*. On the left side of the figure, the monitored values are represented. These are the *ids* of the executed activities, in the order of execution, and the monitored value of *response time* (which represent *fvalue*) for the simple elements (e.g.: `receive`, `reply`, `assign`, `invoke`). These represent the input data for the QoS aggregation algorithm.

The `while` element (*id1=3*) performs two iterations. The activities inside the `while` have two values except for the `if` element *(id1=5)*. In the first iteration, the first

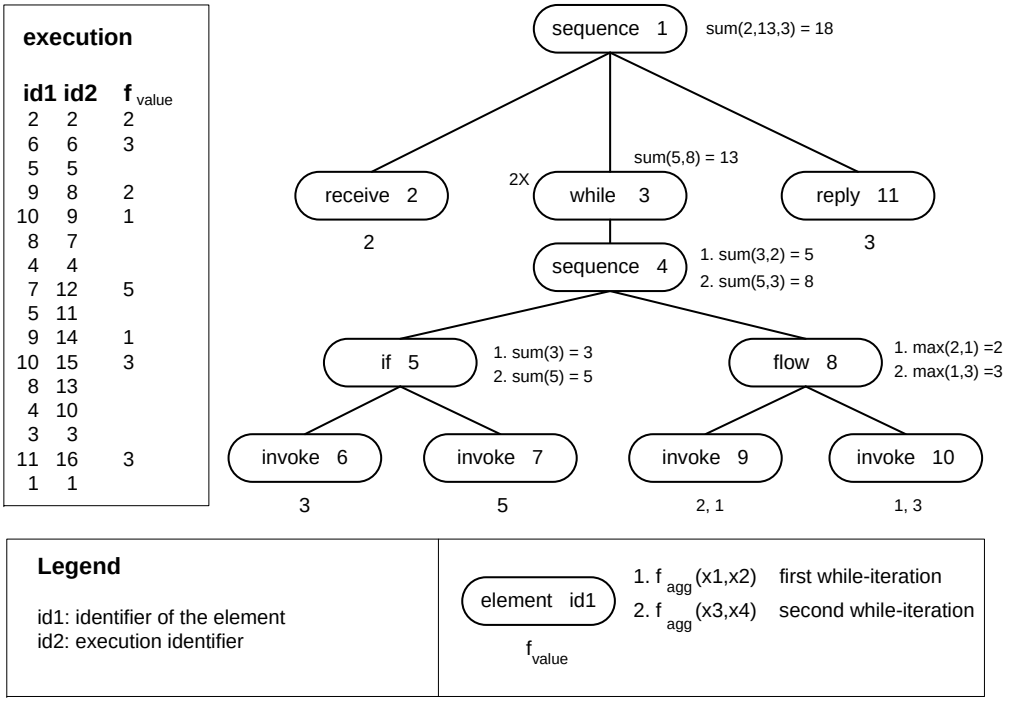

Figure 6.2: Response Time Aggregation

branch of the `if`-element is executed. The computation starts with the `receive` element, *id1=2, f_{value}(responsetime,2)=2*, which is the first element that is completed. Then this value is added to the *ChildrenValues* map of the parent node (`sequence`, *id1* = 1). The process is continued with every completed element in the *execution* list. Finally, the last completed element is `sequence` with *id1 = 1*, which is also the root of the tree. By applying the aggregation function on the values of the children (the *ChildrenValues* map) of the root node (f_{agg}*(responsetime, sequence, X) = 2+13+3=18*), we determine the aggregation value of the *response time* for the entire tree.

6.4.2 QoS of the Service Composition

While the process instance QoS represents the measured QoS for a single process instance, it is also desirable to measure the process QoS for a longer period of time *I*. In this case, the process QoS is computed as the average value of all process instances that were completed in the time interval *I*. Analogously, we compute the section QoS.

In case a process instance (or activity) terminated with an error, this instance is considered only in the computation of availability and reliability. For the computation of response time and throughput of an instance (respectively section) only the successful instances (respectively sections), where no errors occurred, are considered.

6.5 The QoS Estimation Algorithm

The estimation algorithm returns the estimated QoS values for the process based on the probability that certain execution paths are executed. We developed the estimation algorithm based on the approach proposed by Canfora [39]. In their work, Canfora et al. describe some aggregation functions for the computation of QoS for service compositions.

In our framework, the estimation algorithm can be triggered with different QoS values for the atomic services. The following values are possible:

- the QoS values *promised* by the service provider, which are retrieved from the service registry
- the QoS values *monitored* in the BPR framework
- the QoS values *predicted* by the CoFee Predictor

If the estimation is computed with the promised QoS values for the atomic services, the accuracy of the estimated QoS relies on the preciseness of the QoS values advertised by the service provider. This means, that if the service provider advertises QoS values that are close to the runtime behavior, the algorithm makes a reliable estimation. There is still the risk that the service provider promises QoS values which he can not hold but only for attracting clients. However, if the monitored or predicted QoS values from CoFee are used in the estimation, this means that also the experience of other service clients is taken into account. This has the advantage that the BPEL process provider will profit from the experience of other service clients. The disadvantage is that only services may be considered which are registered with our CoFee Predictor.

The estimation algorithm uses the BPEL tree described in Section 6.2. After being created, the BPEL tree has to be initialized with certain probabilities p for conditional structures and the estimated iterations k for the loops. These probabilities and iteration numbers are further used in the aggregation functions for computing the QoS of complex activities. The QoS values (q) of a complex activity (e.g. sequence, flow, etc.) are computed from the QoS values q_i of the activities which are nested inside it. For the simple activities we may consider the promised, monitored or predicted QoS. The estimation algorithm traverses the tree from the bottom to the top, loads the QoS of the simple nodes and computes the QoS for every complex node in the tree. In the final step, the QoS values of the root node are computed which represent the estimated QoS values of the entire process.

The QoS value for a complex node (with children) is computed using the aggregation functions shown in Table 6.2 (taken from [39]). The table is not complete but it provides examples for the structured activities *sequence*, *switch*, *flow* and *loop*. The probability p_i appears for conditional activities like *switch* and corresponds to the probability for executing the branch i of the activity and $\sum_{i=1}^{n} p_i = 1$. For example, when computing the response time of a switch activity, this is the sum of all the response times of the switch branches multiplied with the probability of executing the corresponding branch. Nodes representing a loop (e.g. *repeatUntil*, *while*) receive an iteration number k that represents the average number of iterations monitored for that

loop. For deriving the execution probabilities of branches and the average iteration numbers for loops, the estimation algorithm considers instances executed in a certain period of time.

QoS Dimension	sequence	flow	loop	switch
response time(r)	$\sum_{i=1}^{n} r_i$	$\max_{i \in 1..n} \{r_i\}$	$k \cdot r$	$\sum_{i=1}^{n} p_i \cdot r_i$
cost (c)	$\sum_{i=1}^{n} c_i$	$\sum_{i=1}^{n} c_i$	$k \cdot c$	$\sum_{i=1}^{n} p_i \cdot c_i$
availability (a)	$\prod_{i=1}^{n} a_i$	$\prod_{i=1}^{n} a_i$	a^k	$\sum_{i=1}^{n} p_i \cdot a_i$
reliability (l)	$\prod_{i=1}^{n} l_i$	$\prod_{i=1}^{n} l_i$	l^k	$\sum_{i=1}^{n} p_i \cdot l_i$

Table 6.2: Aggregation Functions from [39]

6.6 Measurement Algorithm vs. Estimation Algorithm

In this section, we compare the measurement algorithm and the estimation algorithm with each other and highlight the differences and similarities between them. Since the two aggregation algorithms have different goals, one for QoS assessment and the other for QoS estimation, we used them for different purposes in our framework. The measurement algorithm is used for the QoS assessment for the current service bindings of the process. In contrast, the estimation algorithm can be used to estimate the QoS of the process if other potential services would be selected. Therefore, particularly the estimation algorithm is also used by our service selection algorithms.

For the *QoS values of the atomic services and simple nodes* the two algorithms consider different values. The measurement algorithm uses the instance QoS for the simple nodes and the current service bindings and computes the QoS on each of the process instances. The estimation algorithm can be triggered with different values for the atomic services: the monitored, the promised or the predicted QoS. With the measurement algorithm, the QoS values of the atomic services and simple nodes may receive for availability and reliability either the values 0 or 1, for a single execution. Using the estimation algorithm, this QoS value would be a value in the interval between 0 and 1.

While the measurement algorithm determines the QoS behavior of each concrete process instance, the estimation algorithm considers the "*average*" *QoS* behavior of the process. Thus, the measurement algorithm computes the QoS value for a single *execution path* of the process, and the estimation algorithm computes the QoS considering several execution paths with certain probabilities. The measurement algorithm uses the execution list containing the order of execution for the activities. In contrast, the estimation algorithm considers the probabilities of executing certain branches within a conditional structure and the average value of the number of iterations for loops. The assessed QoS values of the process instances computed by the measurement algorithm are further used to compute the average QoS of the process or section for the required time interval. In comparison, the estimation algorithm considers the average QoS

values on the simple nodes (atomic services), monitored during a time interval. The measurement algorithm may be applied for running and terminated instances. Thus, it may compute the QoS of a section while the process is still running.

6.7 Automated Deployment

One important task of our framework is the automated monitoring of BPEL processes. For the monitoring purpose, sensors need to be associated to the BPEL process previous to the deployment. The monitoring task is supported by the utilization of sensors which is a feature offered by the Oracle BPEL Process Manager engine, where our business processes are deployed. A sensor is associated to a BPEL activity and is fired at the beginning and end of an activity and on the occurrence of certain events.

Before process deployment, we dynamically associate sensors to each activity in the process. The sensors are declared apart from the BPEL process description inside separate XML files. We automatically generate these sensor files from the BPEL description. They are then interpreted by the Oracle BPEL engine for firing the sensors. The sensors provide valuable information, such as the timestamp when the associated activity was activated, completed or faulted. The BPEL process and its Web services may be monitored by different parties and the monitored data is stored into different databases.

Even if our generic algorithm takes advantage of the sensors offered by the Oracle BPEL Process Manager, it could also run with another BPEL engine as well. If the BPEL engine used does not support sensors similar to those the Oracle engine provides, these software components should be additionally implemented. The premise that our algorithm runs on another BPEL engine is that its input data is delivered properly. The direct impact of sensors is on the creation of the *execution* list, which contains the identifiers of the activities in the order that they were triggered.

The main component of our assessment framework is the *QoS Aggregator* which performs the aggregation of the business process and its sections. Figure 6.3 illustrates the QoS aggregator component as well as its input and output data.

The flexibility of our monitoring framework is sustained by the use of plug-ins, which are software components. The advantage of using plug-ins is that they may be easily added. As the aggregation and value functions (f_{agg} and f_{value}) are subject to modifications, we will use for each aggregation and value functions plug-ins. Plug-ins are reusable components. The f_{agg} function plug-ins can be reused over several instances since they only depend on the type of BPEL activities. The advantage of the f_{value} function plug-ins is that they can retrieve monitoring data from other sources as well, regardless where the monitoring actually takes place. An example for this is a situation where Web services inside the BPEL process are monitored by the Web service providers themselves and the monitored data are stored into a database different from the database where the data from the monitored BPEL *process* are stored.

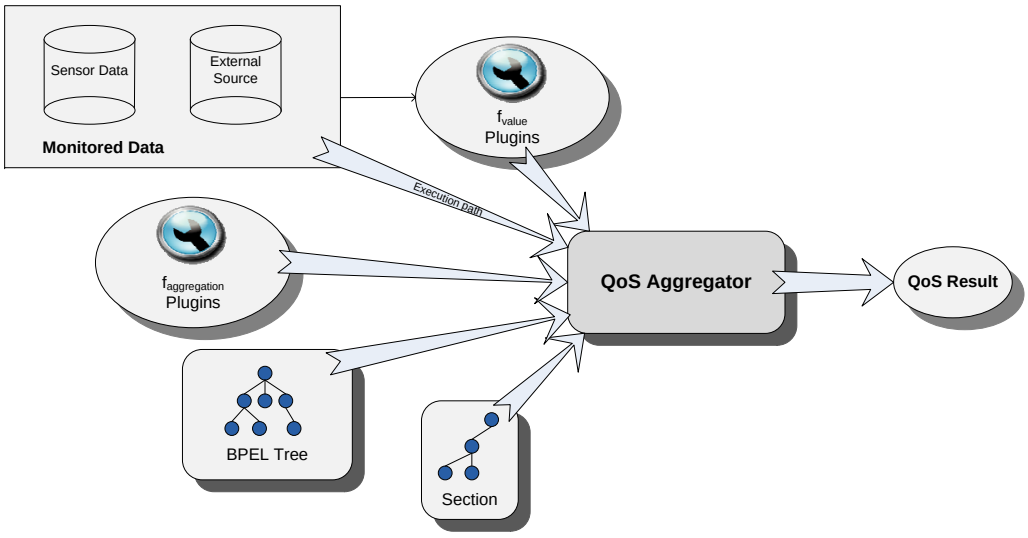

Figure 6.3: The QoS Aggregator Component

6.8 Discussion

The measurement algorithm and the estimation algorithm are applicable for numeric QoS dimensions. The QoS of the service composition is obtained through aggregation, by applying aggregation functions. If new QoS dimensions need to be monitored, then the corresponding aggregation functions have to be provided. Most of the BPEL activities can be mapped to the control structures provided in Table 6.2. If some other activities need to be considered, these need to be checked whether they should be treated as simple or as complex elements. The simple elements should be monitored by proxies or sensors and the QoS should be computed. Depending on the desired purpose, the QoS of the complex elements can be computed as a QoS assessment using the QoS measurement algorithm or as a QoS estimation using the estimation algorithm.

An improvement to our monitoring would be to increase or decrease the monitoring at runtime as it was proposed by [25]. Our monitoring approach is flexible when new QoS dimensions need to be considered. However, we perform the monitoring continuously, for every activity and section of the composition, independently whether the process behaves well or not. It would also be possible to perform a detailed monitoring only when the process behavior gets worse. This would have the advantage of reducing the overhead that is produced by monitoring. In our framework, we could increase the monitoring when the rule set changes, for example, when a *red* rule set is activated.

7 Web Service Selection

In this chapter, we present three heuristic algorithms, *OPTIM_S, OPTIM_HWeight* and *OPTIM_PRO*, for the selection of services in service orchestrations. The OPTIM_HWeight algorithm is an extension of OPTIM_S and uses a special heuristic function. The OPTIM_PRO algorithm considers priority factors for performing the service search. In Chapter 10, we evaluate and compare the algorithms OPTIM_HWeight and OPTIM_PRO with each other and also with the genetic algorithm proposed by [39], which we call GA_CAN. Our experiments revealed that our OPTIM_PRO and OPTIM_HWeight algorithms need less computation time than the genetic algorithm GA_CAN and reach optimization values at least as good as GA_CAN. Our selection algorithms were published in [49].

7.1 Requirements

The QoS of the entire process is dependent on the QoS of the services that build up the service composition. Finding the optimal solution for a service composition means selecting those services that satisfy the QoS requirements, including the QoS constraints, and optimizing an objective function for the entire orchestration. Selecting the right concrete services for the composition is known as the *service selection problem* and is a crucial issue for the successful execution of the process. Assuming that we have n abstract services and each abstract service may have m concrete service realizations, we get a total of m^n possible combinations. The selection problem is known to be NP-hard [22], but the selection task needs to be performed in reasonable time. This problem has received much attention from the research community [110, 39, 31].

7.2 Service Selection

The goal of the service selection algorithms is to select exactly one concrete service for an abstract service. Each abstract service from the composition will be bound to the concrete service that was selected by the service selection algorithms. As shown in Section 5.3, the request for service selection can be specified in the BPR rules. Since several types of service selection may be requested, these are all supported by the service selection module. The selection algorithms may be triggered on the promised QoS values or on monitored or predicted QoS and QoE values. When the promised QoS values are requested, they are obtained from the service registry. The predicted QoS and QoE values are retrieved from the CoFee module, which we present in more detail in Chapter 8. These QoS and QoE values of the concrete services are the inputs for the

selection algorithms. The selection algorithms perform in the same way independently of the QoS types.

We define the set S_a that contains the abstract services, the set S_c containing the concrete services, and the set QD of QoS dimensions. The set V represents the set of service variants, meaning the possible combinations of service candidates. The vector $Q = (a, l, r, c) \in \mathbb{R}^4$ contains the QoS values of the QoS dimensions *availability* (a), *reliability* (l), *response time* (r), and *cost* (c) computed for the service variants $v \subseteq V$. As an example, we consider the following QoS requirements for the orchestration:

$$f_{obj}(Q) = \frac{k_1 \cdot a + k_2 \cdot l}{k_3 \cdot r + k_4 \cdot c} \tag{7.1}$$

$$\text{maximize } f_{obj}(Q) \tag{7.2}$$

$$a > b_1,\ l > b_2,\ r < b_3,\ c < b_4, \text{where } b_i \in \mathbb{R} \tag{7.3}$$

The factors k_i, $i = 1 \ldots 4$, represent the weights for the $q \in QD$ variables depending on the user's preferences, and the b_i represent the bounds for the q variables. This example shows that within our approach we address also non-linear objective functions.

7.2.1 Tree Representing the BPEL Process

As input for our service selection algorithms we create a tree out of the BPEL description file. The example in Figure 7.1 only serves to illustrate how our algorithms find a selection for a composition with four abstract services S_A, S_B, S_C, and S_D which can be realized by different concrete services. It does not show a realistic BPEL tree as this would be too complex here. The nodes of the BPEL tree contain the activities from the BPEL description file which are relevant for the execution of the BPEL process. It does not contain nodes for `partnerLinks`, for instance. We define S as the set of *simple element types* that contains simple BPEL activities like `invoke` and `assign`. These activities are represented as leaves in the tree. The set C of *complex element types* contains structured BPEL activities like `sequence`, `while`, and `switch`, which we represent as inner nodes.

Each node of the tree has the same structure and contains:

- the type *node.elem* $\in S \cup C$ of the node, the reference *node.parent* to the parent node, and the references *node.children* to the child nodes,
- the set of variants *node.V* containing combinations of the service candidates (e.g. $V = \{[v_1 = (S_2, S_4)], [v_2 = (S_2, S_5)]\}$),
- the set of QoS dimensions *node.QD* (e.g. $node.QD = \{a, l, c, r\}$),
- the set of QoS values *node.vQ* of variant v, where $v \in node.V$,
- the set *node.VQ* for all the variants $v \in node.V$ (e.g. $VQ = \{[v_1(c = 7, r = 3)], [v_2(c = 9, r = 4)]\}$), the objective values $node.VF_{obj}$ computed for all the variants $v \in node.V$ having the QoS values *node.VQ* (The objective value of variant v is vF_{obj} and is equivalent to $f_{obj}(vQ)$, analogously we define the heuristic value $f_{Heu}(vQ)$).

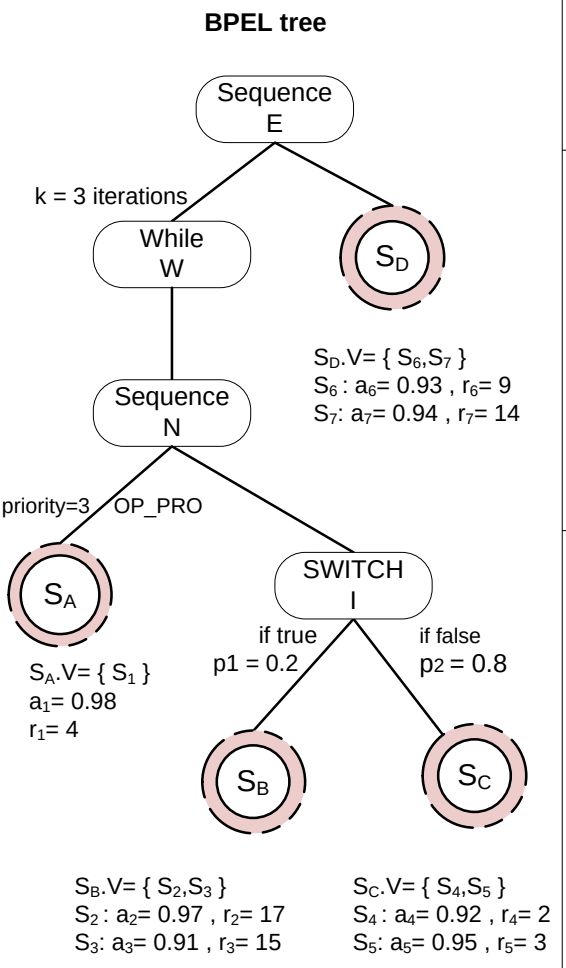

Figure 7.1: BPEL Tree Example

7.2.2 QoS Aggregation and Constraints Checking

The QoS of the service composition depends on the QoS values of the services that build up the composition. The QoS value for a complex node (with children) is computed by the aggregation functions, which are also used with our QoS estimation algorithm (see Section 6.5) and which have been shown in Table 6.2. These aggregation functions take into account the execution probabilities of the different branches to calculate the expected value for the quality dimensions.

Still, the QoS constraints need to be fulfilled for every execution path. Thus, for the *switch* activity we have to consider the worst case of the QoS values over all branches of the *switch*. Therefore, for constraint checking we use modified aggregation formulas for the *switch* activity. Table 7.1 contrasts the original formulas used in our QoS estimation algorithm (column *switch*) with the modified formulas for constraint checking (column *switch worst*). For all other complex activities the formulas for QoS aggregation are used for constraint checking unmodified.

Table 7.1: Constraints Checking

QoS Dimension	switch	switch worst
response time(r)	$\sum_{i=1}^{n} p_i \cdot r_i$	$\max_{i \in 1..n} \{r_i\}$
cost (c)	$\sum_{i=1}^{n} p_i \cdot c_i$	$\max_{i \in 1..n} \{c_i\}$
availability (a)	$\sum_{i=1}^{n} p_i \cdot a_i$	$\min_{i \in 1..n} \{a_i\}$
reliability (l)	$\sum_{i=1}^{n} p_i \cdot l_i$	$\min_{i \in 1..n} \{l_i\}$

The QoS values of the candidate services are normalized to map the values to the $[0, 1]$ interval. This is done with the formula adopted from [110]:

$$q' = \begin{cases} \frac{q - q^{min}}{q^{max} - q^{min}} & \text{if} \quad q^{max} - q^{min} \neq 0; \\ 1 & \text{if} \quad q^{max} - q^{min} = 0. \end{cases} \tag{7.4}$$

7.3 Selection Algorithms

7.3.1 OPTIM_S Algorithm

In this section, we introduce the OPTIM_S algorithm which can perform a heuristic search, a local search, and a global (brute-force) search by only modifying its parameters. Since our OPTIM_HWeight algorithm is an extension of our OPTIM_S algorithm we will first describe the OPTIM_S algorithm. We developed the OPTIM_S algorithm with the intention to allow for easy adaptation to different situations, depending on the number of services that are available at runtime.

The OPTIM_S algorithm permits different types of search (local, global, and heuristic search) for the service selection by only changing its parameters. Thus, the algorithm

Algorithm 1: OPTIM_S(tree, qosConstraints, f_{obj}, nr_v, *optional* f_{Heu})

Input:
tree- the BPEL tree,
q*osConstraints* – the QoS constraints,
f_{obj} -objective function,
f_{Heu} heuristic function,
nr_v the maximal number of variants selected for a node
Output:
the selected services

```
1:  Begin
2:   //initialize the tree nodes with probabilities (p) and iterations (k)
3:   initializeTree( tree)
4:   //initialize the set C with complex nodes and Sa with the abstract services
5:   init(C,Sa)
6:   //traverse the tree from bottom to top
7:   repeat
8:        node ← treeTraverseBottomToTop(tree)
9:        if node.elem in Sa // Sa- is an abstract service, a call to a service
10:           node.V ← node.getConcreteServices()
11:         else
12:          //if node is a complex element with children
13:           if node.elem in  C
14:              children ← node.children
15:              childrenV  ← { }
16:              foreach child in children
17:                 child.VQ ← aggregateQoS(child.V)
18:               //sort the variants with f_Heu and cut those that are too many
19:                  child.V ← sortAndCut(child.V, child.VQ, f_Heu, nr_v)
20:                  childrenV  ← childrenV U child.V
21:               endforeach
22:               node.V ← combine(childrenV)
23:              //compute QoS for each variant and
24:              //eliminate those that don't meet QoS
25:               foreach v in node.V
26:                 //aggregate with formulas for constraints check
27:                  node.vQCons ←aggregateQoSCons(v)
28:                  if  not (checkQoSConstraints(node.v, node.vQCons, qosConstraints))
29:                        node.V ←node.V \ v
30:                 endif
31:               endforeach
32:            endif
33:         endif
34:    until node.parent = null
35:    node.VQ ← aggregateQoS(node.V)
36:    node.VFobj  ←  computeFobj(node.V, node.VQ, f_obj)
37:      //sort the variants by the value of f_obj
38:    node.V ← sort( node.V, node.VFobj)
39:  return node.V[1]
40:  End
```

Figure 7.2: OPTIM_S Algorithm

allows for choosing between shorter computation time of the algorithm and better solution quality. The choice of the search strategy should depend on the number of abstract services, for which the search has to be performed, and the number of services available for the abstract services. For example, when the selection targets only few abstract services (like in a sub-orchestration) a brute-force search is sufficient. In contrast, when the search is performed for the entire process which contains many abstract services, a suitable optimization algorithm is needed. The choice of the selection algorithm may differ between set-up time and runtime. At runtime, a quick and effective solution is usually preferred to an optimal but slow strategy. All these requirements have been considered in the development of our selection algorithms.

As inputs, the algorithm receives the BPEL tree, the QoS constraints, the objective function f_{obj} to be minimized or maximized, the maximum number of selected variants (nr_v) for a node, and optionally a heuristic function f_{Heu} used within the selection process. The output of the algorithm will be the service selection fulfilling the QoS constraints and having the best value found for the objective function.

The basic idea of the algorithm is that, starting with the leafs, for each of the nodes in the tree (*node.V*) we select only a subset of the variants of the children of this node so that the size $|node.V|$ of the set of variants is at most nr_v. For each variant the QoS is computed by using the aggregation functions in Table 6.2. With this QoS value (vQ) the heuristic value is computed by applying the heuristic function $f_{Heu}(vQ)$. The variants are sorted according to this heuristic value and those variants with better values are selected and propagated to the parent node. An example for a heuristic function is the objective function itself.

The different search types *heuristic*, *local*, and *global search* can be switched by modifying the parameters of the OPTIM_S algorithm nr_v and f_{Heu}. The *global search* is a simple brute force search without any heuristic function. The algorithm is invoked by calling it with $nr\_v = \infty$ (in Java we take `Integer.MAX_VALUE`). After sorting the variants on the root node with f_{obj}, the variant $node.V[1]$ will be the global optimum. This kind of search is suited when the search is performed on a small number of services. The *heuristic search* is triggered by calling the algorithm with $\infty > nr\_v \geq 2$ and a given heuristic function. The discovered solution is not necessarily the optimal solution, but a good heuristic could provide a near-optimal or even an optimal solution. The *local search* is triggered by calling the algorithm with $nr\_v = 1$ so that only one variant is selected at each node. While this is the fastest way, the solution quality is expected to be worse than from the heuristic search.

We describe the algorithm on the basis of the pseudocode (see Figure 7.2). The service selection starts with traversing the tree from the bottom to its root node (line 8). We distinguish between nodes that represent a service call and complex nodes. If the node represents a call to a service, the variants of the node *node.V* are initialized with the concrete services (lines 9-10). The number nr_v represents the maximum number of variants that are selected for each node. This allows us to restrict the size of the search space considerably. For each of the non-leaf nodes in the tree, the variants are selected from the variants of the child nodes (lines 13-21) so that the number of combinations of the child variants is at most nr_v. The heuristic function helps us to select those candidates which are likely to perform better in the process (line 19). At selection time, the child variants are already sorted by their heuristic values computed with

f_{Heu}. Furthermore, the selected child variants are combined with their siblings (see the *combine* method in line 22) on their parent node. The QoS values of the variants are computed (line 27) using the formulas in Table 6.2. The variants are checked against the QoS constraints and those which do not fulfill the constraints are eliminated (lines 28-29). In the final step, the variants of the root node are sorted with regard to the objective function and the variant on top of the sorted list (*node.V[1]*) represents the service selection that has won the evaluation.

7.3.2 OPTIM_HWeight Algorithm

The OPTIM_HWeight algorithm makes use of the OPTIM_S algorithm, providing a specific heuristic function $f_{HWeight}$ which is used to rank and sort the candidate services/variants. OPTIM_HWeight is a probabilistic iterative algorithm with the heuristic function $f_{HWeight}$ at its heart. By virtue of this function, the candidate variants are sorted at each node, ascending from the leaves to the root, and only the best rated variants are kept. The heuristic function $f_{HWeight}$ is defined for an arbitrary node N as follows:

$$\begin{aligned} f_{HWeight}(\vec{W}_N, q_N^c, q_N^s) &= \vec{W}_N \cdot (\vec{q_N^c} - \vec{q_N^s}) = \nabla f_{obj} \cdot (\vec{q_N^c} - \vec{q_N^s}) \\ &= \left(\frac{\partial f_{obj}}{\partial q_1^N}, \frac{\partial f_{obj}}{\partial q_2^N}, \ldots, \frac{\partial f_{obj}}{\partial q_n^N} \right)^T \cdot (\vec{q_N^c} - \vec{q_N^s}) \end{aligned} \tag{7.5}$$

with $\cdot$ being the scalar product, $\vec{q_N^s}$ the QoS vector of the current variant selection, ∇f_{obj} the gradient of the objective function at the current variant selection, $\vec{q_N^c}$ the QoS vector of the candidate variant, and the q_i^N being the QoS value of the node (aggregated value for complex nodes) of the i^{th} QoS dimension. We denote the difference vector $\vec{q}_{dif} = \vec{q_N^c} - \vec{q_N^s}$. Figure 7.3 illustrates these vectors in a three-dimensional QoS space consisting of the dimensions *availability*, *response time* and *reliability*.

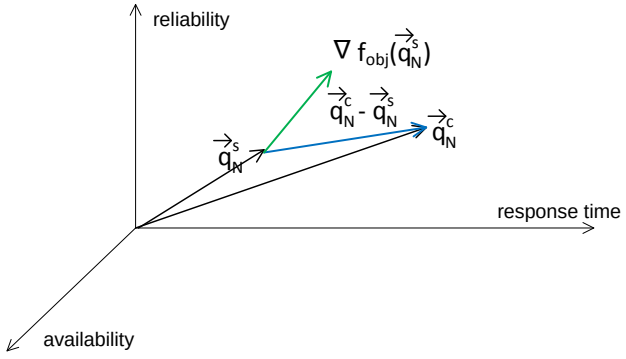

Figure 7.3: Gradient at Point $\vec{q_N^s}$

We take the gradient ∇f_{obj} computed at node N as the weight vector $\vec{W}_N$ considering the QoS values of the current selection. The result of the $f_{HWeight}$ function is used to deliver a score for the candidate service selection versus the current service selection; a higher value is ranked higher. The idea of this approach is essentially a *gradient ascent*.

Given the current point represented by $\vec{q_N^s}$, i.e. the QoS vector of the current selection, we calculate the gradient at that location. We try to find another variant which has "moved" from the point $\vec{q_N^s}$ in the direction of the gradient. As we have only discrete locations in our search space (the service selection variants) we can only choose another point with a minimal error. For this purpose, we compute the heuristic $f_{HWeight}$ as the scalar product between $\vec{W_N}$ and $\vec{q}_{dif}$. For the iteration we use the newly found point in the search space and retry (for n_steps).

In order to compute the derivatives for the gradient, we have to consider that the objective function is a chain of aggregations, so we need to use the chain rule for partial differentiation. This can be explained by the fact that the QoS of a node in the tree is an aggregation of the QoS of its child nodes, and the QoS of each child node is again an aggregation of its child nodes, etc. For instance, in Figure 7.1 the tree has a root node E which has a child node W, and node W has a child node N which has a child S_A (abstract service) being a leaf. For the first quality dimension q_1 we may have the weight factor w_{S_A} computed at node S_A:

$$\begin{aligned} w_{S_A} &= \frac{\partial f_{obj}}{\partial q_1^{S_A}} = \frac{\partial f_{obj}(agg_E^{q_1}(agg_W^{q_1}(agg_N^{q_1}(q_1^{S_A}))))}{\partial q_1^{S_A}} \\ &= \frac{\partial f_{obj}}{\partial agg_E^{q_1}} \frac{\partial agg_E^{q_1}(agg_W^{q_1}(agg_N^{q_1}(q_1^{S_A})))}{\partial q_1^{S_A}} \\ &= \ldots = \frac{\partial f_{obj}}{\partial agg_E^{q_1}} \frac{\partial agg_E^{q_1}}{\partial agg_W^{q_1}} \frac{\partial agg_W^{q_1}}{\partial agg_N^{q_1}} \frac{\partial agg_N^{q_1}(q_1^{S_A})}{\partial q_1^{S_A}} \end{aligned} \tag{7.6}$$

where $agg_X^{q_1}$ denotes the aggregation function for node X with regard to the QoS dimension q_1. The partial derivatives can be efficiently computed when traversing the tree top-down because we only need to reuse the last computed value at the parent node and multiply the inner derivative for the current node.

In the following, we explain the pseudocode of the algorithm (see Figure 7.5). The weight factors of $f_{HWeight}$ differ depending on the nodes in the tree where the heuristic is evaluated. At each node in the tree we need the partial derivatives for each QoS dimension. The weight vector $\vec{W_N}$ is computed (procedure *computeWeight*) by aggregating the QoS values of variant v from the bottom of the tree to the top. Through backpropagation of the influence of the QoS dimensions from top to the bottom of the tree, we compute the weight factors $\vec{W_N}$ for $f_{HWeight}$. This is done by calculating the partial derivatives, which requires the aggregated QoS values of the current selection as input.

$OPTIM\_HWeight$ performs two iterative processes, the external loop (lines 6-24, with n_iter iterations) and the internal loop (lines 13-19, with n_steps iterations). The internal loop can be interpreted in two different ways: (1) it implicitly performs a gradient ascent with regard to the objective function (2) it iteratively adjusts the weight factors $\vec{W_N}$ of the heuristic function. Starting from an initial random variant v_0, the $\vec{W_N}$ vector is computed (procedure *computeWeight*) through bottom-up aggregation of the QoS of v_0 in the tree and backpropagation of the influence of the QoS dimensions

Algorithm OPTIM_HWeight: procedure computeWeight

```
1: Procedure computeWeight(tree, variant)
2: //computes the weights vector w for the heuristic f_HWeight
3: begin
4:   tree.InitToLastLeaf();
5: //Aggregate QoS for the variant from the bottom to the top of the tree
6: repeat
7:     node = treeTraverseBottomToTop(tree)
8:    //get the sub-variant v of the node that coresponds to variant
9:    node.v = node.getSubVariant(variant)
10:    node.vQ = aggregateQoS(node.v) //aggregate QoS values for v
11:  until (node.parent == null)
12: //propagate the QoS values from the top to the bottom of the tree and
13: //compute the weights
14: repeat
15:     node = treeTraverseTopToBottom(tree)
16:     foreach q in QD
17:        if(node == tree.Root)
18:           node.Weight(q) = partialDerivative(f_obj, q, node.vq);
19:        else
20:           node.Weight(q) = node.parent.Weight(q) * partialDerivative(Agg(node.parent, q),
21:                                                    node.siblings.vq, node.vq));
22:        endif
23:     endforeach;
24: until (node == tree.LastLeaf)
25: return tree.Weight;
26: end;
```

Figure 7.4: OPTIM_HWeight Algorithm, Procedure computeWeight

Algorithm OPTIM_HWeight: procedure OPTIM_HWeight

```
1: Procedure OPTIM_HWeight ( tree, qosConstraints, f_obj, nr_v, n_iter, n_steps)
2: begin
3:  //initialize multiple random variants, optimize with OPTIM_S,
4:  //and select the best one
5:  init(v_best)
6:  for i = 1 to n_iter do
7:     v_0 = randomVariants(tree,SC)
8:     //select a random variant from the set of concrete services
9:     tree.Weight = computeWeight(tree, v_0)
10:    v_prev = v_0;
11:    //iterative improvement of weights, iterative steps of the gradient
12:  //ascent
13:    for j=1 to n_steps do
14:       //create the heuristic function f_HWeight
15:       f_HWeight = createFHWeight(tree.Weight, v_prev)
16:       v_s = OPTIM_S(tree, qosConstraints, f_obj, nr_v, f_HWeight)
17:       tree.Weight = computeWeight(tree,v_s)
18:       v_prev =v_s
19:    endfor
20:    //from the found Variants select the one that optimizes fobj
21:    if v_sFobj > v_bestFobj
22:       v_best = v_s
23:    endif
24: endfor
25: return v_best
26: end
```

Figure 7.5: OPTIM_HWeight Algorithm, Procedure OPTIM_HWeight

from top to the leafs of the tree. Knowing the weight factors $\vec{W_N}$ and the QoS of v_0, the heuristic function $f_{HWeight}$ can be created (line 15). The function $f_{HWeight}$ is used inside the $OPTIM\_S$ algorithm (in the *sortAndCut* procedure, see $OPTIM\_S$, line 19) and calculates the scalar product between $\vec{W_N}$ and the difference vector $\vec{q_{dif}}$ $= \vec{q_N^c} - \vec{q_N^{v\_0}}$, i.e. the difference of the QoS vector of the candidate variant at the actual node and the QoS vector of the current selection v_0. Now the OPTIM_S algorithm is called to find the (desired optimal) variant v_s (line 16) that optimizes the objective function. This selected variant v_s is considered in the next iteration step as starting variant to make a further adjustment of the weights $\vec{W_N}$ and perform a further step of the gradient ascent. The computation of $\vec{W_N}$ and of the selection variant v_s starts again and the iterations continue until the n_steps iteration steps have been performed.

In the external loop (line 6 - the for loop) all the steps described above are repeated n_iter times with different random variants as starting points for the inner loop. From the found variants during multiple iterations (i) the one with the best value for optimizing the objective function (lines 21-22) is finally selected.

7.3.3 OPTIM_PRO Algorithm

Our OPTIM_PRO algorithm is the fastest of the presented algorithms. The OPTIM_PRO heuristic algorithm calculates *priority factors* and uses the objective function to sort the variants. It is described in pseudocode in Figure 7.6.

During monitoring of the execution, the nodes in the tree receive an iteration number k and a probability p as explained previously. Each of the nodes that represents an abstract service (Sa, lines 5-8) receives a priority as a product of k and p, and the node is added to the set of abstract services *SaSet*. The *priority factors* state that those nodes which are executed more often should receive a higher priority.

The algorithm proceeds with sorting the nodes from the *SaSet* (line 9) in descendent order of the computed priorities so that the nodes with higher priorities are processed first. OPTIM_PRO is an iterative algorithm which improves the found variant (the objective value) of the root node with each iteration (lines 11-41) step. After selecting a variant for the root node in the first iteration, this variant is going to be improved during the next iterations. A copy of the root is created (*roottmp*, line 14) in order to check if the currently selected service candidate (sc) is an improvement to the objective function. In the root copy variant, the currently selected service candidate replaces the old service candidate. With this new service candidate value, the QoS value of the root copy variant is aggregated ($roottmp.vQ$), checked against the constraints, and the objective function is computed ($roottmp.vFobj$). If the objective function yields a better value, the root receives the value of its copy ($roottmp$), otherwise it remains the same. The variants that no longer can be improved are saved into the list $vlist$. When this is the case, the root variant receives new random candidate services and the improvement process starts again. After reaching the maximal iteration number (n_iter), the iterative process stops. The list $vlist$ that contains the found variants is sorted due to their objective values. The first element in the list is returned as being the best variant that was found for optimizing the objective function.

Algorithm OPTIM_PRO (tree, qosConstraints, f_{obj}, n_iter)

```
1:Begin
2: // initialize the tree nodes with probabilities (p) and iterations(k)
3: initializeTree( tree, p, k)
4: init( root.v, C, Sa, vList)
5: foreach node.elem in Sa // node is an abstract service
6:      node.priority ← computePriority(tree) //compute priority = k * p
7:      SaSet ← SaSet U node
8:  endforeach
9:  SaSet ← sortByPriority(SaSet , 'descendent')
10: i ← 0
11: NEXT while (i < n_iter)
12:     i ← i + 1;
13:     foreach sa in SaSet
14:        c ← 0; roottmp ← root
15:        foreach sc in sa.V
16:          //sc is a concrete service that realizes the abstract service sa
15:          c ← c + 1
17:          //sc replaces the old candidate service of sa, in the root copy
18:           //variant
19:          roottmp.v(sa) ← sc
20:          if (checkQoSConstraintsAggregate2 (roottmp.vQ, qosConstraints))
21:                roottmp.vQ ← aggregateQoS1 (roottmp.v)
22:                roottmp.vFobj ← computeFobj (roottmp.vQ)
23:                if ((i==1) AND(c==1)) OR (roottmp.vFobj > root.vFobj)
24:                   //optimizing obj function OR first iteration,
25:                    //first candidate service
26:                    root ← roottmp
27:                endif
28:          endif
29:          //else select the variant with the minimal distance to fulfill
30:           //constraints
31:        endforeach
32:        if (root.v in vList )
33:             //root.v receives random service candidates
34:             root.v ← chooseForAllRandomServices()
35:             continue NEXT
36:        else
37:            //save the root variant in the root variants list, vlist
38:            vList ← vList U root.v
39:        endif
40:      endforeach
41:   endwhile
42:   sortByFobj(vList) //sort the variants list by their objective values
43:   return vList[1] //return the best variant
44:  End
```

Figure 7.6: OPTIM_PRO Algorithm

7.4 Summary

We presented two heuristic algorithms as solutions to the service selection problem in Web service orchestrations: the OPTIM_HWeight algorithm based on weighting factors inspired by gradient ascent approaches, and the OPTIM_PRO algorithm utilizing priority factors. Both algorithms are iteratively improving the found solution by every iteration step. OPTIM_S provides an easy way to trade computation time against the quality of the solution by merely changing its parameters. As our target was to optimize

non-linear objective functions, we compared OPTIM_HWeight and OPTIM_PRO with the genetic algorithm GA_CAN proposed by Canfora [39] (see Chapter 10). Our experiments revealed that our OPTIM_PRO and OPTIM_HWeight are faster than GA_CAN (in average they needed about 22% and 30%, respectively, of the time of GA_CAN) and even achieve better values for the objective function (in our experiments up to 7% better) than GA_CAN in cases with a high number of combinations. The OPTIM_PRO algorithm turned out to be the fastest algorithm. In our future work, we will also consider a combination of both algorithms, like having the solution of OPTIM_PRO as starting point for OPTIM_HWeight.

8 CoFee - The Feedback Collector and Predictor

This chapter describes how response time, throughput and QoE are predicted by the **Fee**dback **Co**llector and Predictor (CoFee) module. The selection of services in the BPR framework is improved by considering the predicted quality values retrieved from CoFee. In turn, prediction quality benefits from taking into account the context data of users and services, as we show in Chapter 10. We present the *QPred* algorithm that incorporates context data in the prediction. Results of the prediction approach were published in [24]. Furthermore, the creation of context-aware processes that offer new opportunities to build personalized processes for certain client categories is discussed.

8.1 Requirements

Service Selection with CoFee: The previous chapter treats the problem of service selection. In reality, the advertised QoS and QoE values are different from the actually monitored or perceived values. Therefore, when selecting services, it is desirable to consider the actually experienced QoS and QoE values. The BPR framework monitors only the Web services which are integrated into the BPEL processes. However, the users of the BPEL processes managed by the BPR framework should profit from the experiences of other service users outside the BPR framework as well. For the selection of services it is desirable to consider the experienced QoS and QoE values of all Web services which are potential candidates to be integrated into the business processes managed by the BPR framework.

Feedback collection: A new system component (CoFee) is required to collect all feedbacks (including QoS and QoE) of the users that invoked the services. It has to be analyzed how a feedback is created, which data it contains and how the feedback data is retrieved.

Quality property prediction: Based on the clients' experiences from past service invocations, the QoS and QoE values should be predicted. For this purpose, an appropriate prediction algorithm is needed. It should be analyzed whether the consideration of certain context variables in the prediction improves the prediction quality.

Integration with the Service Selection module: The predicted QoS and QoE values have to be considered in the selection of services. For this purpose, the service selection module of the BPR framework has to retrieve the predicted values from the CoFee module.

Context-aware Process: It is desirable to offer personalized BPEL processes to different user categories that have similar context data. Therefore, the service provisioning

should consider the targeted user categories for the service composition. It has to be possible to specify for which user categories the service selection should be performed. Having multiple context-aware processes for different contexts, the user requests have to be forwarded to the corresponding process.

8.2 CoFee - an Overview

Our solution to all these challenges is the **Fee**dback **Co**llector and Predictor (CoFee), a module that is part of the BPR framework. CoFee is responsible for the collection of feedbacks and the prediction of QoS and QoE values for atomic services. The CoFee module communicates with the *Service Selection* module from the BPR framework and offers the possibility to include predicted quality values in the service selection. Although CoFee is part of the BPR framework, it is not tightly coupled to the framework and it could also work independently of the framework. CoFee offers a Java interface for all clients that want to receive QoS and QoE predictions. If multiple instances of CoFee are installed on different servers, these are able to cooperate and exchange their feedbacks. The goal of CoFee is to collect as many feedbacks as possible to make accurate predictions.

As its name suggests, CoFee *collects feedbacks* regarding QoS and QoE from different service users, both from users of BPEL processes, which run within the BPR framework, and also from users that use Web services outside the BPR framework. All these feedbacks are gathered by CoFee and are used to make predictions for future Web service consumptions. In order to make predictions, CoFee relies on quality values retrieved from past service invocations. CoFee retrieves feedbacks that contain QoS values measured by proxies, QoE values assessed by service users and context data of the services and their users.

Context information, QoS measurements and QoE values of previous service invocations are used in the prediction of quality properties for future services invocations. For example, the distances between the locations of the Web services and the locations of the service clients, and the average response time values of the services and users are correlated to the response time values. Such relations between variables have to be detected in order to differentiate whether a context variable is relevant or not in the prediction.

Context data is not only used for improving the prediction quality but also for creating personalized service compositions for certain user categories. If context data is correlated with the QoE of users, consequently the creation of context-aware service compositions would additionally contribute to higher service quality. Therefore, the business analyst may create *context-aware business processes* for certain users categories that have certain context data. For example, there is an *OldiesMusic* Web service that plays music from the fifties for the elderly people, and another *ChartsMusic* Web service playing the current charts for the young people. The business analyst may create a context-aware *MusicEntertainment* process that integrates the *OldiesMusic* Web service for the user category of older people and the *ChartsMusic* Web service for the young people. In this case, the context dimension *age* would be used to build the user

categories of older and younger people in order to offer personalized solutions to the costumers.

8.3 Context-aware Service Composition

In this section, we describe all the necessary steps to build context-aware service compositions. We present how CoFee collects feedbacks, how the business analyst specifies requests for CoFee, how services are provisioned and how the user requests are processed.

In our framework, we consider different types of Web services, which are rated by the users. There are some services that offer products to the users, for example, a *Car Rental* service offers cars for renting to its users. A *Flight* service allows to select the *airline company* for transporting the users to the desired destination. Regarding this type of Web services, the user interacts directly with the products offered by the Web services. Therefore, if the user is not satisfied with the products, he will probably be not satisfied with the Web service as well. Our goal is to integrate services that deliver quality products to its users, because a user that is not satisfied with the delivered products will probably not use the BPEL process a second time. Users with different age and profession commonly have different expectations about the service products regarding their quality and price. Therefore, selecting services that correspond to the users' expectations is an important issue that needs to be addressed. In this thesis, we consider as examples for context data the age and professions of users and the locations of users and services.

8.3.1 Collecting Feedbacks

We interpret QoE as the degree of satisfaction of the user with the service regarding the offered functionality, the offered products, the accuracy of the service results, and the promptitude of the service response. Thus, the rating implicitly encompasses also the response time of the service. The user assesses a rating to the service in the interval $[1,5]$, with 5 meaning excellent and 1 meaning bad. The service rating is similar to the rating that users give to a product in an online shop. The rating can be understood by a user in a personal way, it may encompass several criteria that the user considers as important when using a service. The feedbacks collected by CoFee are further used to make predictions.

The **feedback data** collected by CoFee to make predictions encompasses:

- **the context values** of the Web service, client and/or user [optional],
- **the QP values** with the measured QoS values and the assessed QoE rating.

A Web service may be integrated as an atomic service into an application or into a service composition. In this case, the BPEL process or the application represents the client of the atomic services. This is depicted in Figure 8.1. The feedback for a user consuming the BPEL process contains the QoS monitored by the BPR framework, measured on the server where the BPEL process is executed.

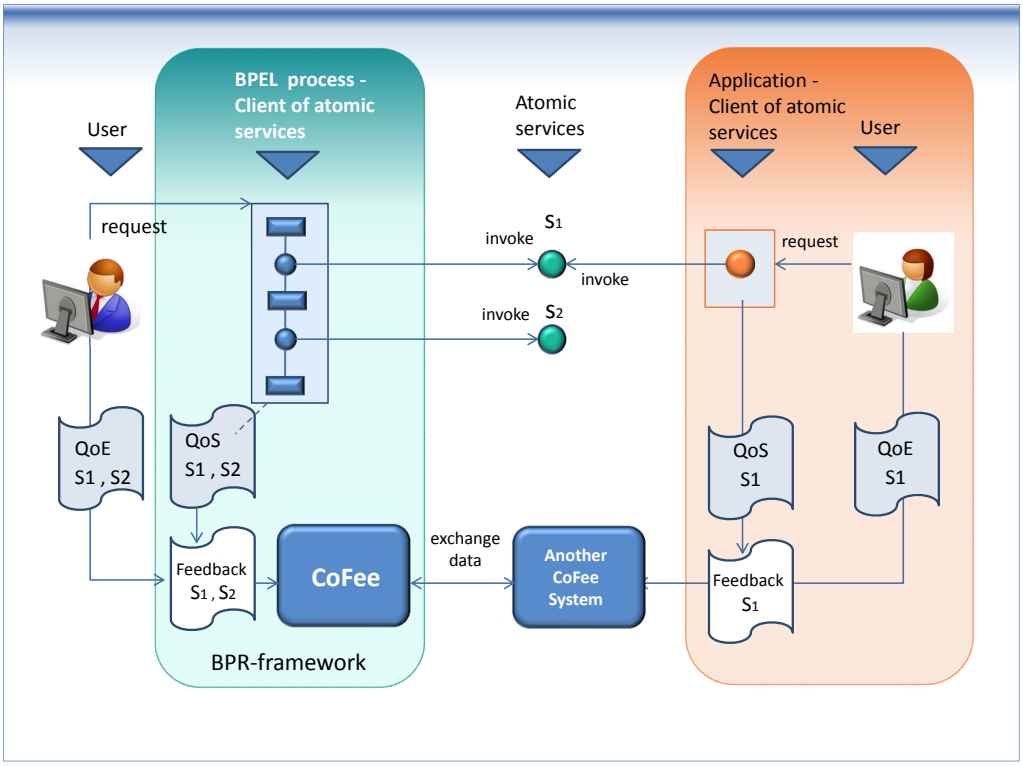

Figure 8.1: Collecting Feedbacks in CoFee

Both the application and the composition can be offered, for example, in an online shop, in a social network or on a Web site. We assume that the user being a customer of an online shop has previously registered to the site and entered his personal profile, like his profession and age. The context data of the user is added automatically to the feedback and this is submitted to CoFee every time the process is invoked. The QoS values measured by the *QoS Monitor* from the BPR framework are also included in the feedback. After using the composition, the user is asked by the BPR framework to submit his feedback, the QoE rating. He assigns his rating for all the Web services he personally interacted with. There may also be a Web service used in the internal logic of the Web service composition so that the user doesn't interact with it directly. In this case, the user is not able to rate it. However, these services are monitored automatically and the QoS values are included in the feedback. If the context data of the user is unknown, then the feedback contains only the QP values.

In Figure 8.2, two types of feedbacks are represented. One feedback is for a single Web service, if the application or composition uses only one Web service. The other feedback is for a service composition and contains the QoS for all the services that are part of the composition and the ratings for all the services with which the user had a direct interaction.

The user is asked to submit his rating for the service via an e-mail that contains the link to the web site with the rating form. The form lists the used services and the products that were delivered to the customer. After the user has entered the rating, the feedback is submitted to CoFee. The time when the user is requested to submit

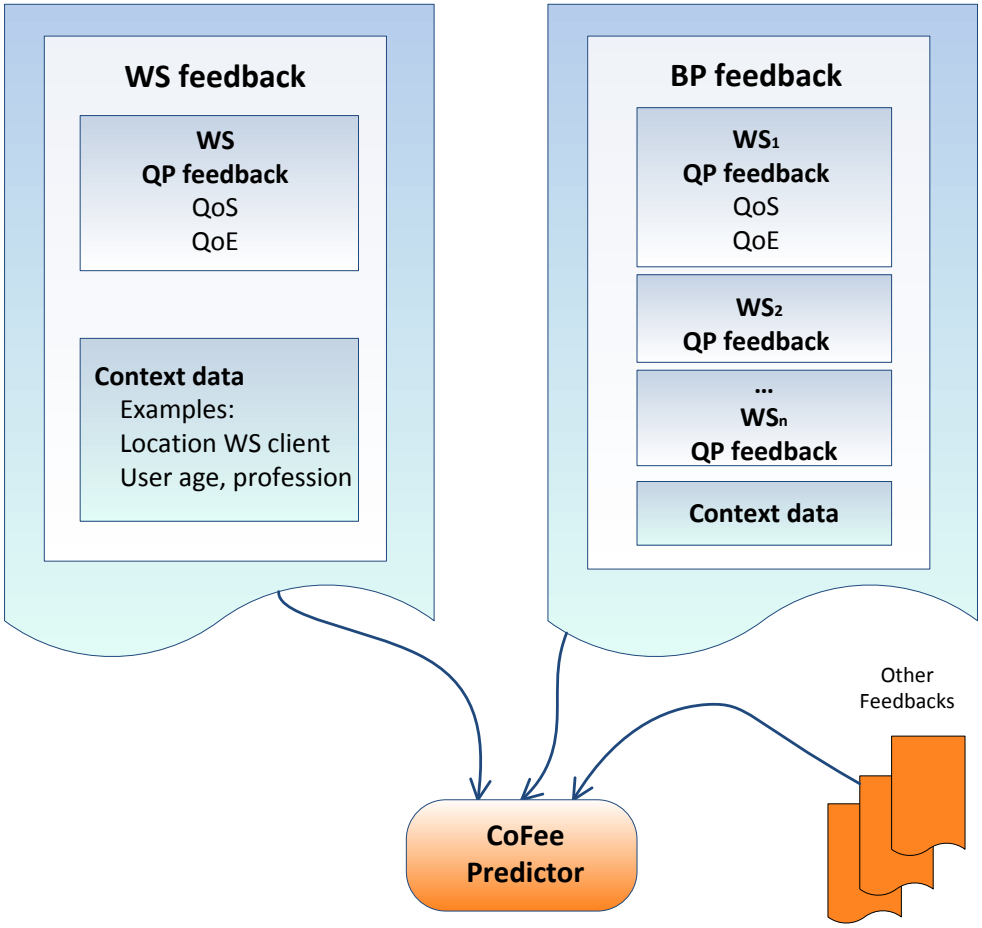

Figure 8.2: Process and WS Feedbacks

his rating depends on the type of the Web service that he consumed. For the Web services that deliver products to their users, the rating request is sent one week after the user received the products. In this way, the user experience with the products can be included in the rating. For the services without products, the user is asked directly after service consumption to rate the services.

8.3.2 Requesting Predictions

To offer personalized solutions to certain user categories of the service composition, the business analyst may specify corresponding requirements in the BPR rules. These requirements are interpreted by the service selection module which extracts the relevant information, creates a request and sends it to CoFee for processing.

Any CoFee client may request QoS and QoE predictions from CoFee. We will call a request to CoFee for service QP predictions a *CoFee-request*. In this thesis, we consider the requests of the business analyst specified in the BPR rules. Other CoFee clients would directly use the CoFee interface.

A **CoFee request** contains the following data:

- **abstract services**,
- **categories** [optional] containing the context values that characterize the user categories.

A CoFee request is created by the service selction module, as a consequence to a BPR condition (e.g. $< constraints\ qualityValues = "Pred" >$) or a BPR action as $< select{-}services >$ or $< select{-}services{-}context >$ where the predicted QP values are required (the $qualityValues$ attribute is set to $"Pred"$). The $< select - services >$ action is triggered either for a single service, for a process section or for the entire process. Therefore, the service selection module detects from the BPEL description file which abstract services are contained inside the section or the process. The list of abstract services and the targeted user categories are added by the service selection module to the CoFee request. Finally, the CoFee request is sent to CoFee for processing. As response, CoFee sends the predicted QoS and QoE values of all the concrete services that realize the abstract services and for all specified user categories back to the service selection module.

We assume that the business analyst has knowledge about the users of the service composition, since the available data about the users is stored automatically by the BPR framework and can be viewed any time by authorized persons, such as the business analyst. The business analyst is also conscious of the business strategy of the process, knowing the target customer groups. We assume that a user category is characterized by certain context values. The business analyst identifies the user categories that the service composition is intended for. When the analyst desires to perform a context-aware service selection for the composition, he specifies one or more target user categories inside the $< select - services - context >$ action. For each user category he declares the context values. If it is not required to perform a context-aware service selection then the context values may be missing and a simple $< select - services >$ action may be triggered instead.

8.3.3 Context-aware Service Composition Example

We consider the *EventPlanner process* as an example for creating context-aware service compositions. The *EventPlanner* process is intended for planning an event where several persons want to participate. For the event, a location is required, the participants have to be provided with food, and music should make the atmosphere at the event more pleasant. Therefore, the *EventPlanner* service composition is created by integrating three Web services: the *EventLocation* Web service, the *Music* service and the *Catering* service. The location for the event can be reserved with the *EventLocation* service, a list of songs to be played are booked with the *Music* service and food is ordered with the *Catering* service. For these abstract services, multiple concrete realizations are available. The *EventLocation_LowCost* service offers locations at low prices to be rented, e.g. a karaoke bar. Another *EventLocation_HighStandard* service offers fancy locations that are more expensive and have a higher standard. Similarly, cheaper food can be ordered with the *Catering_LowCost* service and respectively more expensive and tasty food can be delivered with the *Catering_HighStandard* service. As concrete

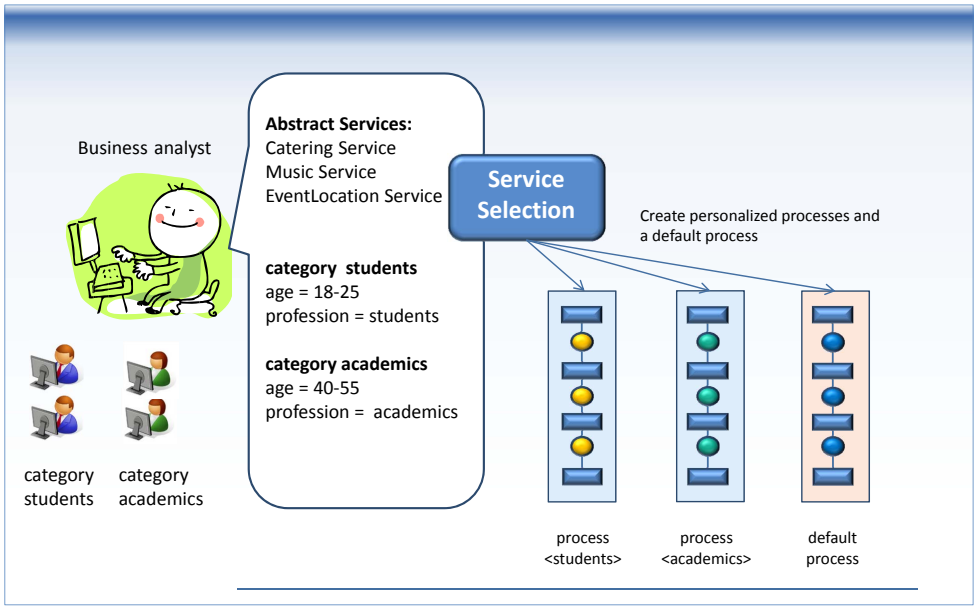

Figure 8.3: Process Replication

realizations for the *Music* service there is the *Music_Pop* service which offers play lists of pop/rock songs, which are currently in the charts, and a *Music_Oldies* service with songs from the fifties to the eighties.

The business analyst wants to create service compositions for two user categories. One category involves students with the age between 18 and 25, and the other category targets academics with the age between 40 and 55. The business analyst entitles the first user category "*students*" and the second category "*academics*". On one side the profession of the users provides information about the special domain where the users work, and on the other side it may be an indicator of the budget that the users have. Services may be intended to customers of certain age, like it is the case for the *Music* service.

We assume that the users' age is in correlation with the ratings of the *Music* services and the professions correlate to the ratings of the *EventLocation* and *Catering* services. We hypothesize that younger people assign a better rating to the *Music_Pop* service and a lower rating to the *Music_Oldies* service. In opposition, the *Music_Oldies* service is preffered by elderly people. A similar assumption is made regarding the users' profession, where the students submit a good rating to the *Catering_LowCost* service and the *EventLocation_LowCost* service, while there are quite few students that provide high ratings for the corresponding *HighStandard* services. The academics are expected to assign good ratings to the *EventLocation_HighStandard* and *Catering_HighStandard* services and bad ratings to the equivalent low cost services. We also assume that the ratings depend on the average ratings of the users and the services.

The business analyst specifies his request in a BPR rule, which is represented in listing 8.1. He desires to create service compositions for the user categories *students* and *academics* and therefore, he triggers a $< select - services - context >$ action. The rule has only an action part (the condition is always *true*) and it is triggered only once,

for the initial creation of the service compositions.

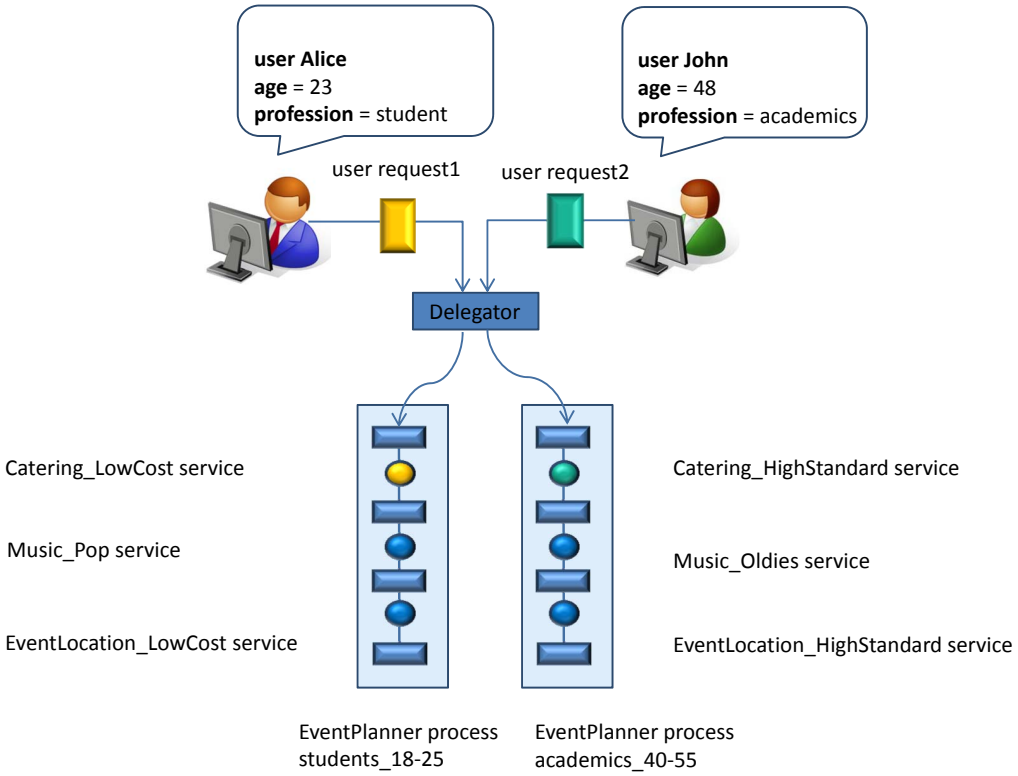

Figure 8.4: EventPlanner Service Composition

```
<rule id="eventPlanner">
  <condition/> <!--condition is always true-->
  <action>
    <select-services-context qualityValues="Pred">
      <quality-requirements>
        <!-- response time < 4, availability > 0.9
             Max qoe -->
      </quality-requirements>
      <list-categories>
        <category name="students">
          <context param="user-age">18-25</context>
          <context param="user-profession">students</context>
        </category>
        <category name= "academics">
          <context param="user-age">40-55</context>
          <context param="user-profession">academics</context>
        </category>
      </list-categories>
    </select-services-context>
  </action>
</rule>
```

Listing 8.1: Context-aware Service Selection

When the rule is interpreted, the BPR framework creates three service compositions by replicating the composition. The BPEL process structure is copied and different concrete services are selected for the abstract services of the replicated processes. The first composition created for the category *students* integrates the low cost services *EventLocation_LowCost* and *Catering_LowCost*, and the *Music_Pop* service. The second composition intended for the category *academics* calls the equivalent high standard services and the *Music_Oldies* service. A default service composition is created for users that are not included in any of the two previous user categories. In Figure 8.3, we show how the *EventPlanner* process is replicated for the two user categories.

Figure 8.4 represents how the user request is delegated to the appropriate service composition. A *delegator* component is automatically created for the *EventPlanner* process. It intercepts the user requests sent to the *EventPlanner* process and delegates the user request to the service composition of the corresponding category. The delegation of requests is described in more detail in Section 8.3.5.

8.3.4 Service Provisioning

As we saw in the example, the creation of *context-aware service compositions* delivers personalized solutions for different categories of users. As context data may influence the values of QoS and QoE, the creation of personalized processes targeting a particular user category can improve the process quality perceived by that category.

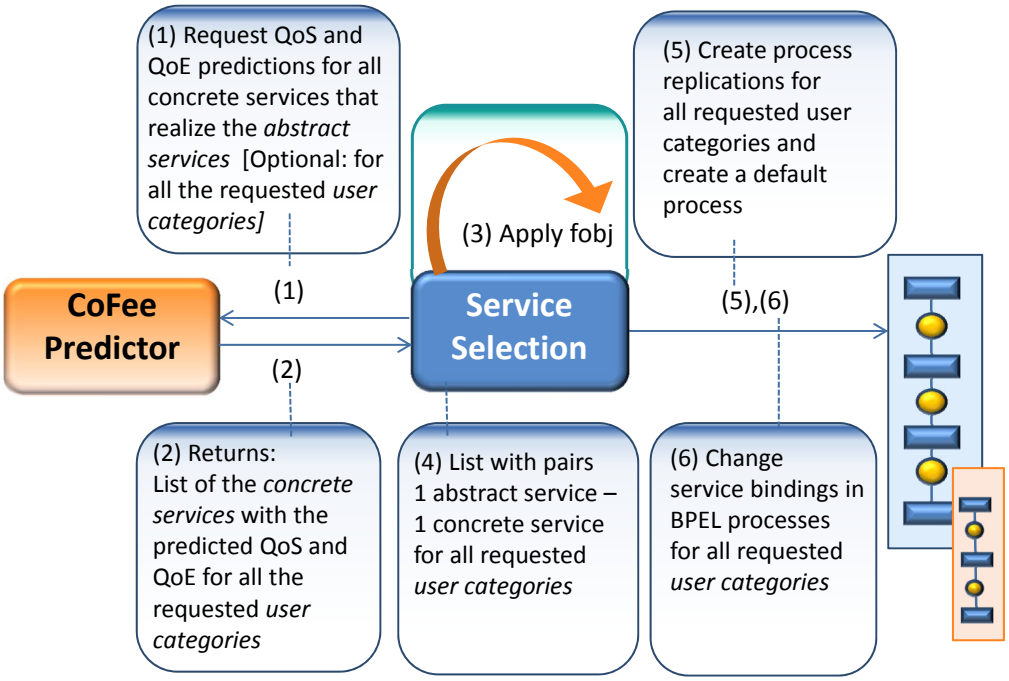

Figure 8.5: Service Selection and Cofee

A *context-aware service composition* is created by selecting for each of the abstract services in the composition a concrete service that has specifically good quality values for a

particular user category. In Figure 8.5, we represent the steps necessary for process creation and service provisioning, starting with the request to CoFee towards the creation of the context-aware service compositions. In the first step, CoFee receives a request, like the one described in the previous section, containing the list of abstract services and one or multiple user categories. CoFee returns the QoS and QoE predictions (the second step) for all the concrete services that realize the requested abstract services for all the requested user categories. From these services, it is necessary to choose only one concrete service per abstract service for a certain user category. Therefore, in step three, the service selection algorithm is triggered for each user category in order to select a service for each abstract service. The objective function is applied to choose only the best service from the multitude of services. The objective function is interpreted by the Service Selection module and it is applied on the predicted values returned by CoFee. After all the concrete services are found with the selection algorithms, they are stored into a list, which we call $serviceList[cat]$ for the user category cat (see step 4). The BPR framework is able to create service compositions for all the user categories provisioning the services from the service list $serviceList[cat]$. This is done by replicating the initial service composition to multiple copies for each of the user categories. The replication of the service composition is done automatically by copying the process in the registry, changing its name and replacing the service bindings with those of the services from $serviceList[cat]$. When a request of a user is received, this is redirected to the service composition with the context that is most similar to the context of the user. A *default process* is created for all the users that do not belong to any of the specified user categories. This is the case when the similarity of the user and the user categories falls below a certain threshold value.

8.3.5 User Request Delegation

Having multiple personalized service compositions managed by the BPR framework, we desire that the user is served by the composition that is most suitable for him. The users of the service composition start the process by sending a request. Typically, the user request contains some entries for the process that have to be processed. Since multiple service compositions are available for different user categories, the question arises which of the service compositions is going to receive the user request. For this purpose, we built the *delegator*, a component responsible for the redirection of the user request to the right service composition. The delegator's task is to compute the similarity between the user context and the contexts of the available processes. Then the process with the highest similarity to the user is chosen and the user request is delegated to this process. In case all the calculated similarities fall below a certain threshold or the user's context data is not available, the request of the user is redirected to the default process for processing. The delegation process is depicted in Figure 8.6.

The similarity between the user context c_u of user u and the process context c_p of process p is computed with Equation 8.1. The distance $dist_{d_i}$ represents the distance between the user context and process context regarding the context dimension d_i. The parameter w_i represents the significance of context variable d_i. If there are two or more processes that have the same similarity to the user, the first of these processes is chosen.

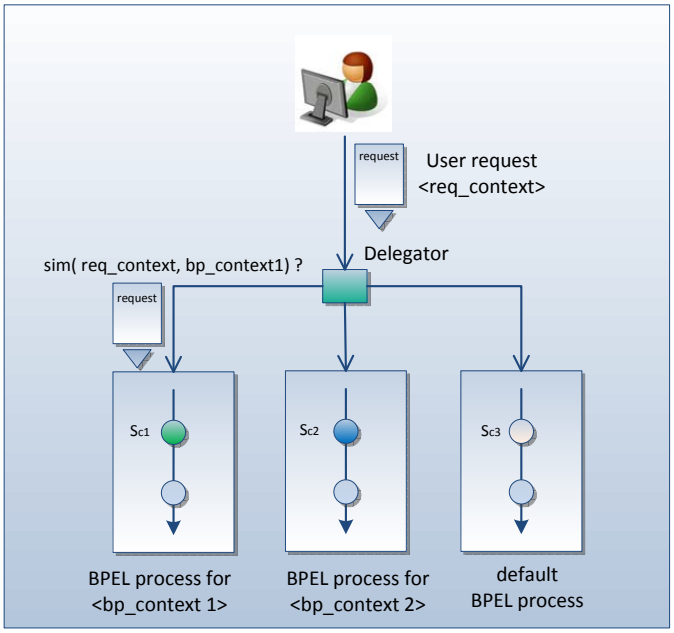

Figure 8.6: Request Delegation

$$sim_{ctx}(c_u, c_p) = \sqrt{\sum_{i=1}^{n} w_i (1 - dist_{d_i}(c_u, c_p))^2} \tag{8.1}$$

8.4 Prediction Approach

This section presents how predictions are made for response time, throughput and QoE of Web services. The QPred algorithm is based on *multiple linear regression* [32],[88], [77] and it is proposed for the prediction of response time and throughput.

8.4.1 Significant Variables

Before executing the prediction algorithm, a preliminary analysis has to find out which of the available context dimensions and which of the monitored or derived QPs are significant in the prediction. This analysis is done automatically and detects the correlations between the variables.

As basis for our analysis with regard to response time and throughput prediction, we used the values from a real dataset of Zheng [112, 111, 17]. The dataset contains also context values such as the locations of the Web services and the users. We represent the locations as geographical coordinates (latitude and longitude) and compute the distance between the locations of users and services. For detecting which variables from the set of context dimensions, QoS dimensions and derived QoS values (e.g. average response time of a service) influence the dependent variables to be predicted

(response time and throughput), we employed the Analysis of Variance (ANOVA) [55]. With ANOVA, the independent variables are partitioned into several levels and it is checked whether the means of the values of the dependent variables differ significantly between the levels. The ANOVA method revealed that the geographic coordinates, the distance d between users and services, the average response times $\bar{r_s}$ of the services, and the average response times $\bar{r_u}$ of the users are in relationship with the response time. The relevance of the average response time of the users may be explained by the connection quality to the Internet that differs from user to user. The different connection quality in several regions and the number of hops between regions may explain the correlation of the distance between users and services and the response time value. The server latency would be an explanation for the influence of the average response time $\bar{r_s}$ of a service.

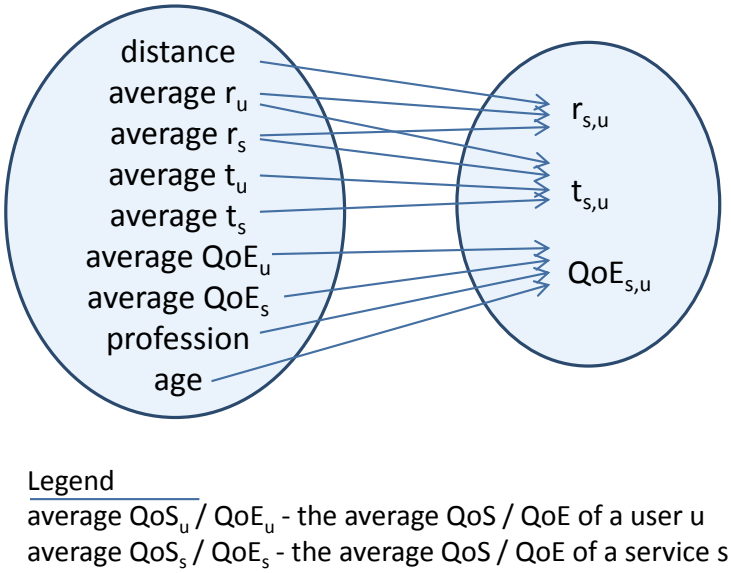

Figure 8.7: Significant Variables

The throughput of a service depends on the average throughput $\bar{t_u}$ of the users, the average throughput $\bar{t_s}$ of the services, the average response times $\bar{r_u}$ of the users, and the average response times $\bar{r_s}$ of the services. The relationship between throughput and response time can be explained by the definition of throughput which reflects that throughput is influenced by response time.

Regarding the QoE dimension, we assume it is dependent on the average QoE ratings of the services and the users, and on context variables such as the user age and profession. If more context data for real Web services would be available for research, this hypothesis could be verified with ANOVA. We expect that the users' profession is in relationship with the assessed QoE value. This is a realistic assumption as we can observe in the following example. For instance, the user is looking for a book that he wants to acquire for his job. We consider a *Bookshop* process that offers books with different subjects for all readers. Another *MedicalBooks* Web service offers medical books. Thus, a physician or a medical assistant probably rates the *MedicalBooks* Service much better than someone that is looking for a novel book.

8.4.2 Prediction of Response Time and Throughput

We developed the *QPred algorithm* for prediction of the response time and throughput of the services. It is not the main focus of this thesis to propose elaborate prediction algorithms for all the different QP dimensions, but to show how the prediction module can be integrated into the BPR framework in order to build context-aware service compositions. Even though, the QPred algorithm is very efficient and provides good prediction accuracy.

The QPred algorithm uses the results from the ANOVA analysis and considers the influencing variables to improve the prediction quality. In the evaluation, a real dataset from Zheng [113] containing response time and throughput was used to evaluate the prediction accuracy. The dependent variables response time and throughput are predicted using the independent variables distance, average response time and average throughput of the services and users. In our approach, predictions are calculated for an entire user category (with similar context values) of a Web service or business process and not only for a particular user. Related approaches [113] make predictions only for a single user.

The QPred algorithm is based on multiple linear regression. As ANOVA revealed, the response time variable depends on the distance between the services and the users, on the average response time $\bar{r}_u$ of the users and the average response time $\bar{r}_s$ of the services. For the response time prediction we defined two regression functions, which we call the *service function* and the *user function*. Analogously, we created two regression functions for throughput.

The *service function* $f^r_{s_z}$, used for predicting the *response time* of user u_x for invoking service s_z, is dependent on the average response time $\bar{r}_{u_x}$ of all the service invocations of user u_x and the distance $d_{u_x s_z}$ between the user u_x and the service s_z. The *service function* is defined as follows:

$$f^r_{s_z}(\bar{r}_{u_x}, d_{u_x s_z}) = c^r_0 \bar{r}_{u_x} + c^r_1 d_{u_x s_z} + c^r_2 \tag{8.2}$$

The coefficients of the regression function c^r_0, c^r_1, c^r_2 are calculated by applying the following equation [88]:

$$C = (X^T X)^{-1} X^T Y \tag{8.3}$$

where Y represents the vector of the predicted response time values and X the matrix containing the observations of the regressor variables, being the average response time values of the users $\bar{r}_u$ and the distances between users and services. A further column with all elments being 1 is added to the matrix X to consider the coefficient c^r_2 for the constant offset.

Similarly, the *user function*, used for predicting the *response time* of user u_x for invoking service s_z, depends on the average response time $\bar{r}_{s_z}$ of the service s_z called by different users and the distance $d_{u_x s_z}$:

$$f^r_{u_x}(\bar{r}_{s_z}, d_{u_x s_z}) = b^r_0 \bar{r}_{s_z} + b^r_1 d_{u_x s_z} + b^r_2 \tag{8.4}$$

The coefficients of the user function b_0^r, b_1^r, b_2^r are computed analogously to the service function with the difference that matrix X contains the values of the average response times of the services $\bar{r}_s$ and the distances.

Similarly to response time, we define two regression functions for throughput. ANOVA showed that the throughput variable depends on the average throughput of the users $\bar{t}_u$, the average throughput of the services $\bar{t}_s$, the average response time of the users $\bar{r}_u$ and the average response time of the services $\bar{r}_s$.

The *service function* for predicting the throughput for service s_z invoked by user u_x is defined in the following way:

$$f_{s_z}^t(\bar{t}_{u_x}, \bar{r}_{u_x}) = c_0^r \bar{t}_{u_x} + c_1^r \bar{r}_{u_x} + c_2^t \tag{8.5}$$

The *user function* is computed as follows:

$$f_{u_x}^t(\bar{t}_{s_z}, \bar{r}_{s_z}) = b_0^t \bar{t}_{s_z} + b_1^t \bar{r}_{s_z} + b_2^t \tag{8.6}$$

Both response time and throughput are finally predicted by combining the predictions of the service and the user function. We define the *combi function fc* as an aggregated weighted value between the user function and the service function. The predictions made by the user and service function are weighted with the coefficients of determination of these functions and combined to the final prediction value. The coefficient of determination reflects the suitability of the regression model, thus using it as a weighting factor is a reasonable choice. The combi function is defined as:

$$fc_{u_x s_z} = \frac{f_{u_x} R_x^2 + f_{s_z} R_z^2}{R_x^2 + R_z^2} \tag{8.7}$$

where R_x^2 represents the *coefficient of determination* of the user function f_{u_x} and R_z^2 of the service function f_{s_z}. The coefficient of determination is calculated as follows [88]:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{8.8}$$

where $\hat{y}_i$ is the predicted QoS value with the regression model and y_i is the measured QoS value.

The evaluation (see Chapter 10) showed that in average the service function has the greatest prediction accuracy, followed by the combi function. In the evaluation, the prediction accuracy is reflected by the MAE computed between the predicted response time (resp. throughput) value and the actual response time (resp. throughput) value of multiple service invocations.

8.4.2.1 Outlier Detection

Linear regression is sensitive to outliers. Few points may significantly impact the calculation of the regression coefficients and consequently affect the prediction. Outliers can be detected by using *Cook's distance* [77]. A large value of the Cook's distance calculated for a point indicates that the point is *influential*. The Cook's distance D_i for the i^{th} point of the set of known points is calculated as follows:

$$D_i = \frac{r_i^2}{p} \frac{h_{ii}}{(1-h_{ii})} \tag{8.9}$$

where r_i represents the *studentized residual* that shows how well the model fits the i^{th} observation y_i. If there are n observations $(x_{i1}, x_{i2}, .., x_{ik}, y_i)$ and k regressor variables, then the distance $h_{ii}/(1-h_{ii})$ reflects how far the i^{th} point is away from the remaining $n-1$ points [77]. The i^{th} point is considered influential if a value of $D_i > 1$ is calculated.

In the following, we explain in detail how the Cook's distance is computed. The value h_{ii} represents the i^{th} element from the diagonal of the matrix H, also called the hat matrix. Matrix H is computed with the following formula [77]:

$$H = X(X^T X)^{-1} X^T \tag{8.10}$$

Matrix X contains the values of the observations of the k regressor variables. The fitted values $\hat{y}$ and the measured values y are linked by the equation $\hat{y} = Hy$.

The studentized residual is computed as [77]:

$$r_i = \frac{e_i}{\sqrt{\sigma^2(1-h_{ii}))}} \tag{8.11}$$

The parameter σ (the variance of the error term ϵ) is estimated by [77]:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} e_i^2}{n-p} \tag{8.12}$$

with $e_i = y_i - \hat{y}_i$ being the residual, i.e. the difference between the observation y_i and the fitted value $\hat{y}_i$, and p the number of regression coefficients.

After computing the studentized residual and the hat matrix, the Cook's distance can easily be computed and the influential points can be detected. However, not all points that are influential should be considered as outliers to be filtered when calculating the regression coefficients. An influential point that is far away from the remaining points, indicated by a large value of $h_{ii}/(1-h_{ii})$, can still be important as it may stabilize prediction in areas where only a few observations are available. Therefore, we applied the following criteria to filter point i when calculating the regression coefficients: $h_i < 0.65$ for response time and for throughput; $D_i > 0.3$ for response time and $D_i > 0.2$ for throughput. These thresholds have been determined experimentally.

8.4.2.2 Prediction Example

In the following, we present an example with 10 users and 10 services and show how response time is predicted. Each user u_x calls a service s_z and the measured response time for each of these invocations is represented in Figure 8.8. We assume that 50% of the values are known and the other 50% values from the table have to be predicted. The known response time values are represented in a blue cell and the ones that need to be predicted are in a gray cell. Some response time values are not available (like some values for the user u_9) and these receive the value of -1. These values are not included in the computation. On the margins of the table, the average response time values for each user and each service are computed. The averages are computed only considering the known quality values.

As an example, we consider the response time prediction for user u_7 and service s_6 and show how the user, service and combi functions are computed to make the prediction.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	$\bar{r}_u$
u_1	0.206	0.798	1.607	0.761	1.008	0.201	0.841	0.135	5.294	4.84	1.619
u_2	0.244	0.59	1.215	0.617	0.425	0.068	0.394	0.251	0.905	0.731	0.47
u_3	0.283	0.698	1.305	0.47	1.169	0.089	0.478	0.3	0.946	0.507	0.821
u_4	0.322	0.5	0.906	0.513	0.528	0.112	0.336	0.337	0.249	0.26	0.471
u_5	0.333	0.549	1.507	0.486	0.352	0.109	0.273	0.309	0.298	0.415	0.507
u_6	0.465	6.046	0.602	5.696	5.842	0.169	5.147	0.425	5.504	5.252	3.692
u_7	3.208	5.741	0.603	1.594	4.238	0.802	0.687	0.353	0.929	0.177	2.961
u_8	0.285	0.974	1.707	0.877	1.1	0.18	0.579	0.29	0.629	0.635	0.794
u_9	0.25	0.453	1.107	0.608	-1	0.052	0.659	-1	-1	-1	0.303
u_{10}	0.407	5.69	0.91	5.681	5.543	0.217	5.117	0.418	5.252	5.211	3.435
$\bar{r}_s$	0.784	4.162	1.165	1.654	2.44	0.125	1.512	0.266	2.388	0.507	

Figure 8.8: Users-Services Response Time Values

Function service is computed as follows:

$$f^r_{s_6}(\bar{r}_{u_7}, d_{u_7 s_6}) = c^r_0 \bar{r}_{u_7} + c^r_1 d_{u_7 s_6} + c^r_2 \tag{8.13}$$

The distance between user u_7 and service s_6 is 5907.1, and the average response time value for user u_7 is 2.961. The regression coefficients of the service function f_{s_6} are $-0.016, 0.039$ and 0.000016, and the coefficient of determination is 0.99. Thus, the response time is predicted as:

$$f^r_{s_6}(\bar{r}_{u_7}, d_{u_7 s_6}) = -0.016 + 0.039 \cdot 2.961 + 0.000016 \cdot 5907.1 = 0.192 \tag{8.14}$$

For the user function f_{u_7} we compute the regression coefficients being -0.62, 0.756 and 0.0002. The coefficient of determination is 0.91 . The average response time value

for service s_6 is 0.125. The response time is predicted with the user function f_{u_7} as follows:

$$f^r_{u_7}(\bar{r}_{s_6}, d_{u_7 s_6}) = b^r_0 \bar{r}_{s_6} + b^r_1 d_{u_7 s_6} + b^r_2 \tag{8.15}$$

$$f^r_{u_7}(\bar{r}_{u_7}, d_{u_7 s_6}) = -0.62 + 0.756 \cdot 0.125 + 0.0002 \cdot 5907.1 = 0.718 \tag{8.16}$$

The combi function $fc_{u_7 s_6}$ represents a weighting of f_{s_6} and f_{u_7} with the coefficients of determination:

$$fc_{u_7 s_6} = \frac{f_{u_7} R^2_{u_7} + f_{s_6} R^2_{s_6}}{R^2_{u_7} + R^2_{s_6}} = 0.442 \tag{8.17}$$

8.4.2.3 The QPred Algorithm

The QPred algorithm computes the predictions for response time and throughput based on the regression functions that were previously presented. For calculating the regression coefficients, the algorithm relies on a set of historical data containing QoS measurements of past service invocations as well as context data for the services and the users. Assuming that QoS measurements for k service invocations are available, these k known points are used to compute the average response time and throughput $(\bar{r}_u, \bar{t}_u, \bar{r}_s, \bar{t}_s)$ of the users and services. Afterwards, the regression coefficients for the user, service and combi functions are calculated from the set of known points. Next, the Cook's distance is used to eliminate outliers, and the coefficients of the service, user and combi function are recomputed. Finally, the three functions are used for computing the predictions.

8.4.2.4 Sparsity Problem

The sparsity problem is encountered when the user-service matrix has not enough entries. In this case, the regression coefficients of the prediction functions can not be computed. Since the prediction is performed using the three functions *service*, *user* and *combi function*, we discuss three cases of matrix sparsity, one for each of the prediction functions. The cases of matrix sparsity are represented in Figure 8.9.

In the first case of matrix sparsity (see Figure 8.9 function service) the quality value $\hat{q}_{u_x s_z}$ of service s_z and user u_x has to be predicted using a "substitute" for the service function f_{s_z} since the regression coefficients for the service function could not be computed. It is the case when less than three quality values are available for service s_z. In case the average quality value $\bar{q}_{u_x}$ of user u_x and the distance $d_{u_x s_z}$ are known (condition(cs_1)), the service function will get as coefficients the average of all the coefficients belonging to the other service functions $(f_{s_y}, s_y \in S, s_y \neq s_z)$. This assumes that at least one other service function could be computed. In case the condition (cs_1)

Function	Sparsity Problem
Function service f_{s_z}	Sparsity Condition $\lvert P_{k,s_z} \rvert < 3,\ SC = \{s_y \in S : \lvert P_{k,s_y} \rvert \geq 3\}$ 1. **If** $\overline{q}_{u_x}$, $d_{u_x s_z}$ are *known* and $\lvert SC \rvert >= 1$ (cs_1) **then** $\hat{q}_{u_x s_z}$ – predicted with the average coefficients of the service functions of the services in SC 2. **Else if** $\overline{q}_{s_z}$ is *known* (cs_2) **then** $\hat{q}_{u_x s_z}$ is predicted as $\overline{q}_{s_z}$ 3. **Else** $\hat{q}_{u_x s_z}$ will not be predicted
Function user f_{u_x}	Sparsity Condition $\lvert P_{k,u_x} \rvert < 3,\ UC = \{u_x \in U : \lvert P_{k,u_x} \rvert \geq 3\}$ 1. **If** $\overline{q}_{s_z}$, $d_{u_x s_z}$ are *known* and $\lvert UC \rvert >= 1$ (cu_1) **then** $\hat{q}_{u_x s_z}$ – is predicted with the average coefficients of all user functions of the users in UC 2. **Else if** $\overline{q}_{u_x}$ is *known* (cu_2) **then** $\hat{q}_{u_x s_z}$ is predicted as $\overline{q}_{u_x}$ 3. **Else** $\hat{q}_{u_x s_z}$ will not be predicted
Function combi $f_{u_x s_z}$	Sparsity Condition $\lvert P_{k,s_z} \rvert < 3$ or $\lvert P_{k,u_x} \rvert < 3$ 1. **If** $\lvert P_{k,s_z} \rvert \geq 3$ and $\overline{q}_{u_x}$ is *known* and $\lvert P_{k,u_x} \rvert < 3$ (c_1) **then** $\hat{q}_{u_x s_z}$ – is predicted with the service function f_{s_z} 2. **Else if** $\lvert P_{k,u_x} \rvert \geq 3$ and $\overline{q}_{s_z}$ is *known* and $\lvert P_{k,s_z} \rvert < 3$ (c_2) **then** $\hat{q}_{u_x s_z}$ – is predicted with the user function f_{u_x} 3. **Else if** $\overline{q}_{s_z}$ is *known* **then** $\hat{q}_{u_x s_z}$ is predicted as $\overline{q}_{s_z}$ 4. **Else** $\hat{q}_{u_x s_z}$ will not be predicted

Figure 8.9: Sparsity Prediction

was not fulfilled and the average quality value $\bar{q}_{s_z}$ of service s_z is known (condition (cs_2)) then $\hat{q}_{u_x s_z}$ is predicted as the average quality value $\bar{q}_{s_z}$ of service s_z. If none of the conditions (cs_1) nor (cs_2) holds, then the value $\hat{q}_{u_x s_z}$ will not be predicted. The second case of sparsity, when a substitute for the user function is searched, is handled in a similar way (see Figure 8.9 function user).

In the third case of matrix sparsity, a substitute for the combi function is searched for predicting the quality value $\hat{q}_{u_x s_z}$. The combi function is tried to be substituted by either the service function f_{s_z} or by the user function f_{u_x}, depending on which one is available. In the first case (condition c_1), it is tested whether the average quality value $\bar{q}_{u_x}$ is known and the service function f_{s_z} is available. If this condition is true then the value $\hat{q}_{u_x s_z}$ is predicted with the service function f_{s_z}. In case the condition

(c_1) is not fulfilled but $\bar{q}_{s_z}$ is known and the user function f_{u_x} is available (condition (c_2)) then $\hat{q}_{u_x s_z}$ is predicted using the user function f_{u_x}. If neither the condition c_1 nor the condition c_2 are fulfilled but the average quality value $\bar{q}_{s_z}$ is known then the value $\hat{q}_{u_x s_z}$ is predicted as $\bar{q}_{s_z}$.

8.4.3 Discussion about QoE Prediction

The QoE may be influenced by several factors, like human, system and context factors [38], and this has to be considered in the predictions of the CoFee module. The approach for QoE prediction presented in this section has not been evaluated with a real-world dataset. Thus, it can only be considered as discussion how the QoE prediction can be handled within CoFee.

We denote with c_1,... ,c_n a set of context dimensions that we assume to influence the QoE of user u_x with regard to service s_y, denoted as q_{u_x,s_y}. The prediction $\hat{q}_{u_x,s_y}$ of the current user u_x for the service s_y is based on the ratings of other users that previously invoked the service s_y. The prediction $\hat{q}_{u_x,s_y}$ relies on:

- ratings of users that assigned similar ratings as user u_x to services that were coinvoked in the past
- ratings of users that have a similar context as user u_x

Based on these assumptions, our QoE prediction approach constitutes a user-based collaborative filtering [86] [113] [92] [64] using two different similarity measures, and the prediction $\hat{q}_{u_x,s_y}$ is computed as follows:

$$
\begin{aligned}
\hat{q}_{u_x,s_y} &= \lambda_1^{s_y} \overline{q}_{s_y} \\
&+ \lambda_2^{s_y} \left(\overline{q}_{u_x} + \frac{\sum_{u_i \in U, sim_{PCC}(u_j,u_x) \geq t_{PCC}} (q_{u_i,s_y} - \overline{q}_{u_i}) sim_{PCC}(u_i,u_x)}{\sum_{u_j \in U, sim_{PCC}(u_j,u_x) \geq t_{PCC}} sim_{PCC}(u_j,u_x)} \right) \\
&+ \lambda_3^{s_y} \left(\overline{q}_{u_x} + \frac{\sum_{u_i \in U, sim_{ctx}(u_j,u_x,s_y) \geq t_{ctx}} (q_{u_i,s_y} - \overline{q}_{u_i}) sim_{ctx}(u_i,u_x,s_y)}{\sum_{u_j \in U, sim_{ctx}(u_j,u_x,s_y) \geq t_{ctx}} sim_{ctx}(u_j,u_x,s_y)} \right)
\end{aligned}
$$

where U represents the set of all users of service s_y for which ratings and context information are available; $\bar{q}_{s_y}$ represents the average rating of service s_y with regard to all users that rated s_y; $\bar{q}_{u_x}$ is the average rating of user u_x provided for all services he used.

The $sim_{PCC}(u_i,u_x)$ function computes the similarity of users u_x and u_i by means of the Pearson Correlation Coefficient considering the assigned ratings to coinvoked services. A second similarity of the users u_i and u_x regarding their context dimensions is computed with the $sim_{ctx}(u_i,u_x,s_y)$ function. The thresholds t_{PCC} and t_{ctx} ensure that only users exceeding a certain similarity are taken into account.

The parameters $\lambda_1^{s_y}$, $\lambda_2^{s_y}$ and $\lambda_3^{s_y}$ are specific to service s_y and are used for weighting the ratings.

With the $sim_{ctx}(u_i, u_x, s_y)$ function, the similarity between user u_i and user u_x based on their context dimensions with regard to service s_y is calculated as follows:

$$sim_{ctx}(u_i, u_x, s_y) = \sqrt{\sum_{i=1}^{n} w_i^{s_y} (1 - dist_{c_i}(u_i, u_x))^2} \tag{8.18}$$

The value of $w_i^{s_y}$ hints at the importance/significance of the context dimension c_i for service s_y. The formula contains the term $(1 - dist)$ to make the transition from distance to similarity. The distance $dist_{c_i}(u_i, u_x)$ represents the normalized distance (mapped to the $[0, 1]$ range) between the user u_i and the user u_x regarding the context dimension c_i.

For example, the distance between two users regarding the context dimension *age* would be $dist_{c_i}(u_i, u_x) = \frac{|age_{u_i} - age_{u_x}|}{|age_{max} - age_{min}|}$.

As the different services may exhibit different characteristics with regard to the influence of context dimensions and QoE assignments in general, the values of λ_i and w_i are selected specific to the service under consideration. Thus, the problem becomes the detection of an appropriate set of λ_i and w_i for each service based on the collected ratings from the users who invoked that service.

The parameters $w_i^{s_y}$ can be derived with the help of ANOVA, testing the influence of context dimension i on the QoE values assigned by the users of service s_y. Having available the parameters $w_i^{s_y}$ the parameters $\lambda_i^{s_y}$ can be obtained by linear regression.

Dealing with missing data

There might be cases when certain context variables for the users are missing and these cases need to be treated adequately. If only context dimensions with small $w_i^{s_y}$ are missing, these context dimensions are simply ignored when calculating $sim_{ctx}(u_x, u_i, s_y)$. If context dimensions with high $w_i^{s_y}$ are missing, but we have information for still enough context dimensions with high $w_i^{s_y}$, the missing context dimensions are simply ignored as well. If all context dimensions with high $w_i^{s_y}$ are missing or no users with $sim_{ctx} \geq t_{ctx}$ are found, the prediction is done just with the average rating of service s_y and the PCC similarity part (the factor of λ_3 is set to 0). If there is not enough data for computing the PCC similarity part, only the average rating of the service s_y is used. If even the average cannot be determined, then the prediction cannot be calculated.

We have also to consider the case when the QoE prediction has to be performed for a whole user group of a BPEL process. QoE ratings from the user group are only available for the services that are used or have been used in the BPEL process. However, the similarity based on the Pearson Correlation Coefficient makes use of the assumption that there exist several coninvoked services, invoked by the members of the user group and other users. It has to be expected that in many cases only a few of such coninvoked services exist, if any at all. In such cases, prediction is only done with the average QoE rating of the service under consideration and the context similarity part from the prediction equation, i.e. λ_2 is set to 0. If we have not available the context values for the important context dimensions of the service, only the average QoE rating of the service can be used.

The QoE prediction approach presented in this section utilizes user-based collaborative filtering with two different similarity measures at its core. Thus, it inherits all the advantages and disadvantages of user-based collaborative filtering approaches as discussed in Section 4.3: its applicability and performance has been proven in many applications, but it requires quite an amount of historical data and predictions can be expensive with regard to computation time. Still, we propse this approach, as we are confident that collaborative filtering provides robust QoE predictions and that prediction accuracy benefits from combining the two similarity measures based on Pearson Correlation Coefficient and based on the context dimensions. The QoE of users with regard to services is most likely influenced by context factors, and, as stated in [113], the Pearson Correlation Coefficient provides accurate similarity computation, but it sometimes overestimates the similarities of service users who happen to have similar QoS experience on a few coinvoked Web services.

The prediction approach can be applied for a single service but in the same way for the entire BPEL process. The determination of the factors w_i for the BPEL process corresponds to the identification of the context dimensions that impact the QoE ratings of users. Here too, high values of w_i indicate significant context dimensions. These values provide hints to the business analyst how to define the user categories helping to set up personalized copies of the BPEL process to deliver a personalized service.

8.5 Summary and Discussion

This chapter presents CoFee, a module that predicts response time, throughput and QoE for services. CoFee collects feedbacks from service users regarding their experience with the services and uses the feedbacks to make predictions. The service selection module from the BPR framework retrieves the predicted values from CoFee and uses them to select services with higher quality. Thus, the service selection is able to consider QoS and QoE values that were monitored and assigned by users outside the BPR framework. We showed exemplarily how response time and throughput can be predicted with the QPred algorithm and discussed an approach to predict QoE. The prediction accuracy can be improved by considering influence factors such as derived QoS values (e.g. average response time) and context data.

The context variables that are identified to impact the prediction of the QP values may also be considered for creating personalized service compositions for certain user categories characterized by those context variables. The business analyst may specify in the BPR rules for which user category he wants to provision services. When a request is received from a user with context variables for which a personalized service composition exists, the user request is delegated to the appropriate composition for being processed. We showed in an example how a context-aware process can be personalized for certain user categories, where the customers receive a service composition according to their budget and age.

We presented two approaches for context-aware QoS and QoE prediction, but our intention was not to provide algorithms that necessarily outperform other related approaches. Rather, our aim was to provide basic methods that are expected to show good prediction performance and to show how the corresponding predictions can be

used in the BPR framework to create business processes tailored to user categories characterized by certain context properties. It remains for future work to investigate for which QoS dimensions these approaches show reasonable prediction performance and for which dimensions improvements of these approaches or even completely other algorithms are needed.

9 The BPR Framework

In this chapter, we present the architecture of the BPR framework and show how a BPEL process is monitored and how BPR rules are interpreted. We also provide some implementation details of the framework.

9.1 Architecture

We have designed and implemented the BPR framework for testing and evaluating how BPR rules specified for BPEL processes impact their quality behavior. The BPEL processes are executed on the *Oracle BPEL Process Manager* engine [8]. We chose this platform as BPEL engine because it provides suitable tools for the development of BPEL processes as well as good online tutorials and documentation. Another advantage is that Oracle provides APIs for controlling BPEL instances and for attaching and querying sensors of the BPEL process, which is important for the monitoring.

The Web services are installed on the Apache Axis2 [4] engine for Web services. We have implemented a service registry using a MySQL database where services can be searched and published.

The BPR framework is implemented in Java and it contains several modules which interact in order to ensure the QoS and QoE management of the process. The architecture of the BPR framework is represented in Figure 9.1 and it mainly contains four modules: the *BPRules Manager*, which represents the core module, the *QoS Monitor & Aggregator* module for QoS monitoring and aggregation, the *Process Management* module for performing the management actions, and *CoFee* for collecting feedbacks and predicting QP. The business analyst specifies BPR documents which are stored into the BPR repository. We distinguish between two execution phases: the *initial* phase, when all the necessary monitoring artifacts are deployed, and the *monitoring phase*, when the actual QP management takes place.

In the *initial phase*, the *BPRules Manager* module loads the BPR documents (see Fig. Figure 9.1, *step Ini 1*) from the BPR repository. The *BPRules Manager* reads from the BPR documents which BPEL processes (identified by the specified process ID), which sections of the process, and which QoS parameters are going to be monitored in the next phase. Before deploying the BPEL process, some preparations for the monitoring need to be done. Therefore, the *BPRules Manager* reads from the BPEL description file which services are required. Then, it triggers the service selection algorithm to select appropriate concrete services for all the services inside the BPEL process. The *BPRules Manager* creates a proxy for each of the abstract services. The proxy is implemented with Java servlets and it contains a reference to the URL of the concrete service. The proxy intercepts the messages and delegates them to the

concrete service that is bound. The information from the proxy (timestamps, number of transferred bytes, number of invocations, errors) is used for the QoS computation of the concrete service. The *BPRules Manager* updates the endpoint references from the BPEL description file with the URL of the proxy (e.g. for the *Stock Service* the URL of the proxy is *http://atlantis:543/buecherservices/StockService*). In this way, when a service replacement is performed, the BPEL process doesn't have to be redeployed, but only the proxy is updated to reference to another concrete service URL.

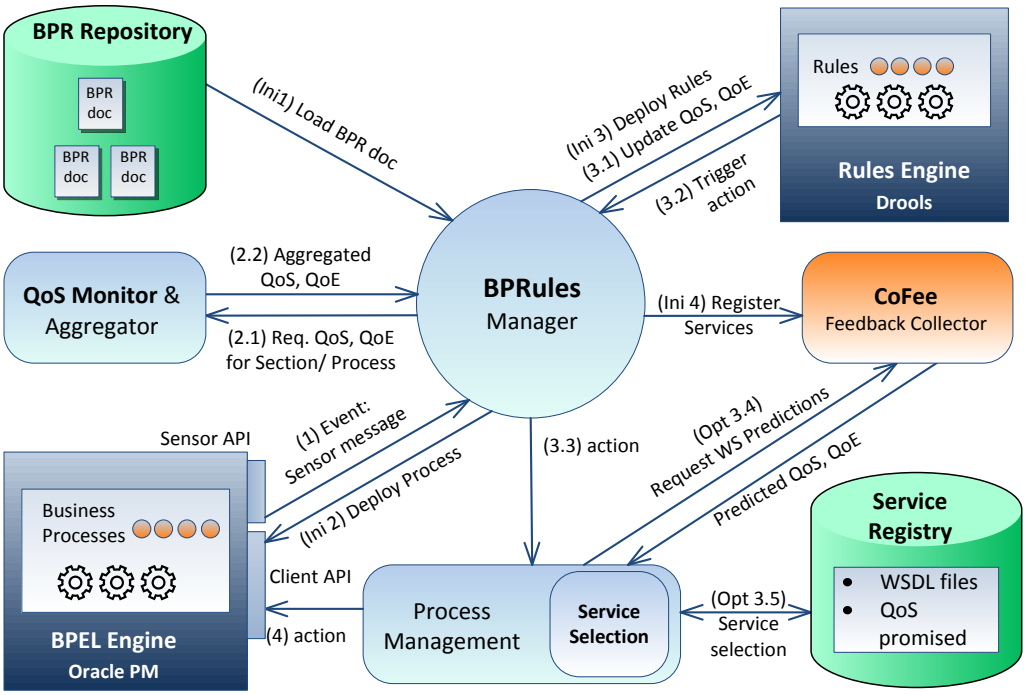

Figure 9.1: The BPR Framework

For the monitoring we use a feature of the Oracle BPEL engine which offers the possibility to attach sensors to the BPEL activities. Such a sensor may inform when a BPEL activity is started/ended or when a failure occurred. The BPRules Manager dynamically attaches sensors to all the activities of the BPEL process. By this, all the monitoring artifacts were created and the BPEL process can be deployed (step *Ini 2*). In the next step (*Ini 3*) the BPR rules need to be deployed on the rules engine.

We employed the *Drools* rules engine from JBoss for executing the rules. Before deployment, the rules are dynamically extracted from the BPR documents and transformed into Drools files. Since BPRules and Drools use similar rule constructions (e.g. condition/action, logic operators), the transformations between the two syntaxes can be done automatically. We also used the possibility offered by Drools to implement customized functions for percentage, minimum and maximum that are applied to the QP objects. After the transformation step, the Drools files are deployed to the Drools engine and the initial phase is terminated.

During process execution, the sensor messages (from each activity) are delivered to the BPRules Manager (see step 1). A sensor message contains the instance ID of the process, the sensor ID, the timestamp, the evaluation time (activation or completion

of the activity) and whether an error occurred. If the sensor represents the end of a section or of the process, the BPRules Manager calls the *QoS Monitor and Aggregator* to perform the QoS computation of the section or the process instance (steps 2.1, 2.2). The QoS of the section or process are computed out of the QoS of the atomic services within the section or process. With these new QoS values the *BPRules Manager* updates the QoS objects from the Drools memory (step 3). The Drools engine permanently evaluates the QoS conditions and in case they are met, it delegates the actions to the *Process Management (PM)* module. Finally, the *PM* module is able to execute the actions on the process. The Oracle BPEL PM engine offers a client API for querying and controlling the BPEL instances (e.g. stopping instances, deploying, undeploying the process) at runtime. Our PM module makes use of this Oracle API and additionally adds other necessary actions (e.g. select-services, replace-ws, etc.).

9.2 Implementation

In this section, we provide some implementation details regarding the service registry and service replacement.

9.2.1 The Service Registry

The service registry is used by the service providers to publish their services. The registry is further queried by the Service Selection module to choose appropriate services for the BPEL process.

The Service Registry is implemented using a MySQL database and it has an interface for accessing it. In the database, we store relevant information from the WSDL interface of the Web services, such as the URL of the service, the namespace of the service and the abstract service name. We also store the QoS and QoE values promised by the service providers in the registry. The registry interface offers the possibility to publish and search for Web services.

9.2.2 Service Replacement

During runtime, the services from the BPEL process may be automatically replaced by other services. We assume that an abstract service that is invoked by the BPEL process can be realized by multiple concrete services that implement the same interface as the abstract service and have the same functionalities.

The automatic service replacement is performed with the help of a proxy, which we call the *ServiceProxy*. The role of the *ServiceProxy* is to forward the messages of the BPEL process to the right concrete services. The communication between the BPEL process, the *ServiceProxy* and the concrete service is represented in Figure 9.2. Our proxy can be used when there is a match between the WSDL interface of the abstract service and that of the concrete service, which means that the service operations have the same signatures. The only differences that may be overcome by our proxy reside in different service names, service URLs and namespaces. Semantic matching is out of

the scope of this thesis, but if this would be desired, then the ServiceProxy could be extended to perform a mediation between semantically annotated interfaces.

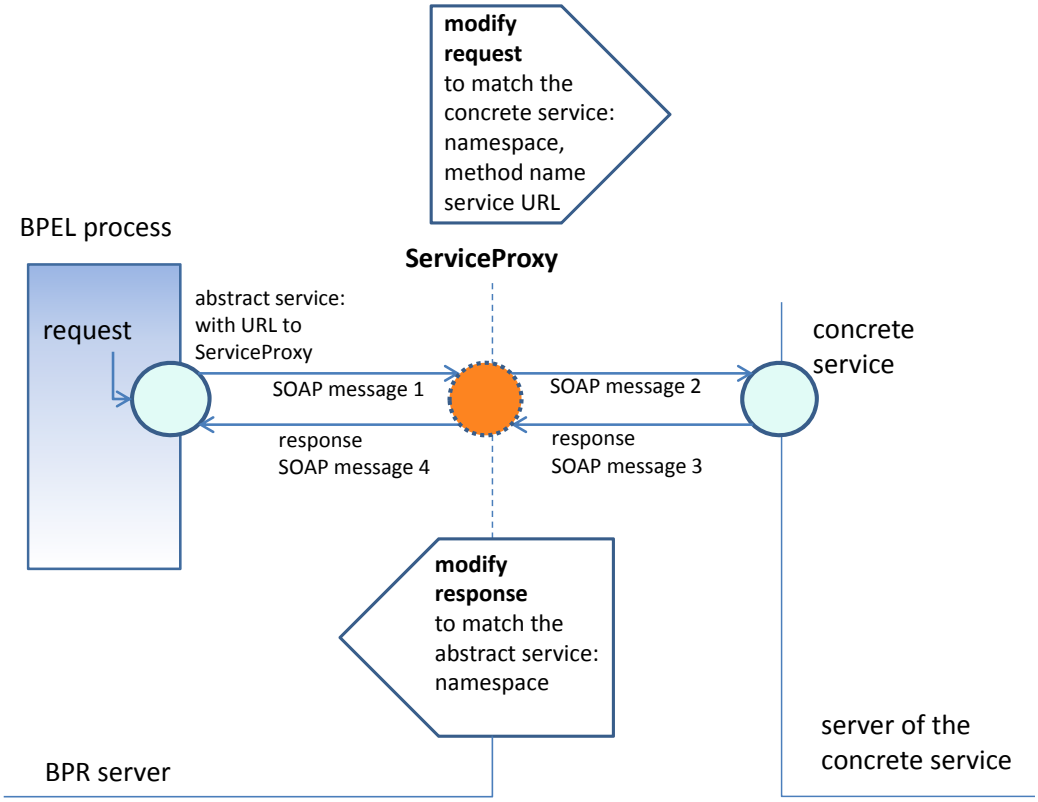

Figure 9.2: Service Replacement

Before the BPEL process is deployed, the service endpoints of the Web services, inside the process, are modified so that they reference the address of the *ServiceProxy*. Thus, whenever a Web service is invoked, the request is sent to the *ServiceProxy*. The *ServiceProxy* is implemented using a *ServerSocket* and it is running on the BPR server.

For example, we have a BPEL process that is calling a *Bank* service. The BPEL process contains a reference to the URL of the *ServiceProxy* having the path of the abstract service *Bank*, for example: $http://192.168.105.4:54321/Services-WServices-context-root/AbstractBankServiceSoapHttpPort$. All the requests for the Bank Service are received by the *ServiceProxy* which delegates them to the concrete Bank Service. The request, in form of a SOAP message, is modified by the *ServiceProxy* before being forwarded to the concrete service. The proxy changes the header and the body of the request so that the namespace and the service URL match the WSDL interface of the concrete service. The concrete service receives the request, processes it and sends the response message back to the *ServiceProxy*. This response is modified again (the namespace) so that it matches the interface of the abstract service, and then it is delivered back to the BPEL process.

A list stores the mappings holding the correspondence between the abstract service name and the binding data of the concrete service. Thus, the *ServiceProxy* knows to which concrete service the request has to be delivered. These mappings are updated by the *BPRules Manager*, whenever it is desired to replace a service with another one, for example, when a new service selection is triggered. Thus, when a service replacement is performed, only the mappings have to be updated and the BPEL process does not have to be redeployed.

10 Evaluation

In this chapter, we present the results of the evaluation of our service selection algorithms and of our context-aware QoS prediction algorithm. We compare our selection algorithms *OPTIM_HWeight* and *OPTIM_PRO* with the *GA_CAN* algorithm proposed in [39], and analyze the computation time and the optimality of the solution. For the QoS prediction algorithm *QPred*, we evaluate prediction quality and stability and compare it with IMEAN (the average QoS values of the Web services). Besides, we measure and discuss the computation time for calculating the regression functions and for performing the actual prediction.

10.1 Service Selection Algorithms

10.1.1 Comparison with the GA_CAN Algorithm

In order to evaluate our heuristic algorithms we implemented a genetic algorithm as proposed by Canfora *et al.* [39] for comparison. Genetic algorithms are inspired by biology and use meta-heuristics to solve optimization problems. The reason why we chose this algorithm is because it can also be applied to non-linear functions and constraints, which is also our target. For more details we refer to [39].

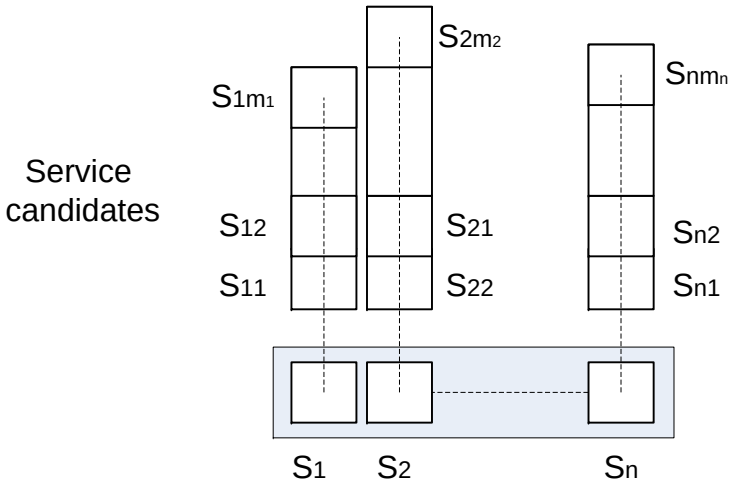

Figure 10.1: Genome

The genome represents the service variants that realize the service orchestration and is encoded as an array. The length of the genome is equal to the number of abstract services. Each element within the array contains a reference to the list of the concrete

candidate services that may realize the abstract service. The initial population is built with random individuals. The fitness of the individuals represents their utility as a solution and is computed using the fitness function defined in Equation 10.2. It corresponds to the sum of the objective function calculated on the genome and the weighted distance $D(g)$ (multiplied with the penalty factor k_5) resulting from constraint satisfaction checking. This means that those individuals that do not fulfill the constraints are penalized with distance $D(g)$. Assuming that there are h missed constraints, the distance $D(g)$ is defined as the sum of all the deviations from each of the missed constraints.

$$D(g) = \sum_{i=1}^{h} dev_i \tag{10.1}$$

The fitness function is computed for the individuals as follows:

$$f_{fit}(g) = f_{obj}(g) + k_5 \cdot D(g) \tag{10.2}$$

We build multiple generations over the population in an iterative way by applying the mutation and crossover operators. Through the mutation operator, the candidate services are varied randomly and an arbitrary concrete service is selected to realize the abstract service. The crossover operator combines service variants of different individuals. The algorithm stops when during multiple generations there is no improvement to the fitness function value.

10.1.2 Evaluation Methodology and Setup

We have evaluated the three algorithms with regard to the required *computation time* and the *optimization of the objective function*.

As baseline for our experiments, we have randomly generated 10 different BPEL trees for each test case with different structures and dimensions. The tree structures have been created in such a way that they contain the relevant BPEL activities, like *while*, *if*, *invoke*, *sequence* and *flow*, with adjustable probabilities.

10.1.3 Experiments

The tests have been performed on a Lenovo R60, 1.83 GHz, 2 GB RAM with Windows XP SP3 and JSDK 1.6. For the GA_CAN algorithm we set the mutation probability to 0.01 and the crossover probability to 0.7. The OPTIM_HWeight algorithm was triggered with $nr\_v = 12$ and $n\_iter = 12$. Since all the algorithms are probabilistic, we executed the algorithms 10 times (for each of the 10 BPEL trees, having 100 test runs in total) and took the average value.

Our experiments A and B are depicted in Figure 10.2, Figure 10.3 and Figure 10.4. We compared the computation time of OPTIM_HWeight and OPTIM_PRO with the computation time of the GA_CAN algorithm (with a population of 100) for reaching approximately the same value (difference less than 0.01 %) for the objective function.

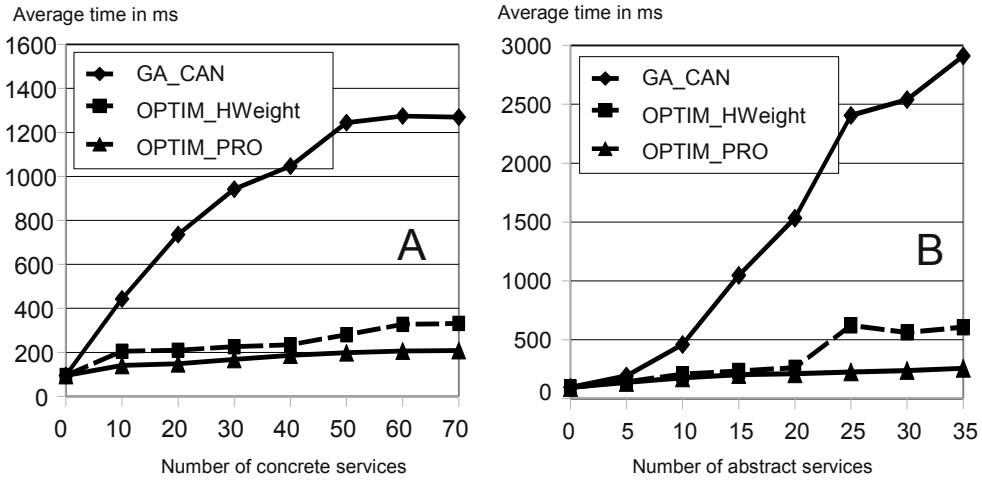

Figure 10.2: Computation Time OPTIM_HWeight, OPTIM_PRO and GA_CAN

In experiment A (see Figure 10.2 left side, Figure 10.3) we used a fixed number of abstract services (15) and measured how an increasing number of concrete services (from 10 to 70 per abstract service) influences the computation time. The results show that OPTIM_PRO is the fastest algorithm, requiring on average about 19% of the time of GA_CAN for reaching approximately the same optimization of f_{obj}. We observed that in average, our OPTIM_HWeight algorithm requires only about 28% of the time of GA_CAN.

Table A

Nr. Con.Serv	GA_CAN(G) Avg. timeG P100 [ms]	HWeight (H) Avg. timeH [ms]	Avg. timeH/ Avg. timeG	O_PRO (P) Avg. timeP [ms]	Avg. timeP/ Avg. timeG
10	443.10	205.10	46.29%	140.80	31.78%
20	736.10	210.10	28.54%	148.50	20.17%
30	942.20	225.80	23.97%	167.80	17.81%
40	1046.50	234.80	22.44%	186.90	17.86%
50	1244.00	280.10	22.52%	198.10	15.92%
60	1274.10	327.30	25.69%	206.50	16.21%
70	1269.00	331.30	26.11%	208.20	16.41%

Figure 10.3: Computation Time of the Algorithms. Varying the Number of Concrete Services

Experiment B is similar to experiment A, but this time we increased the abstract services from 0 to 35 while keeping the number of concrete services constantly at 40. Experiment B (see Figure 10.2 right side, Figure 10.4) shows again that OPTIM_PRO is the fastest algorithm and needed in average about 24% of the time required by GA_CAN, while OPTIM_HWeight needed about 32% of the time of GA_CAN.

In experiment C (see Figure 10.5) we evaluated how well the objective function was optimized by the different algorithms. We computed the value of f_{obj} reached by

Table B

Nr. Abs.Serv	GA_CAN(G) Avg. timeG [ms]	HWeight (H) Avg. timeH [ms]	Avg. timeH/ Avg. timeG	O_PRO (P) Avg. timeP [ms]	Avg. timeP/ Avg. timeG
5	192.10	142.60	74.23%	136.80	71.21%
10	459.30	206.30	44.92%	176.10	38.34%
15	1046.50	234.80	22.44%	203.60	19.46%
20	1532.80	261.20	17.04%	210.40	13.73%
25	2404.70	621.80	25.86%	226.00	9.40%
30	2539.25	561.42	22.11%	238.30	9.38%
35	2909.92	606.25	20.83%	257.20	8.84%

Figure 10.4: Computation Time of the Algorithms. Varying the Number of Abstract Services

OPTIM_HWeight, OPTIM_PRO and GA_CAN, where the computation time limit for all of them was set to 4 seconds. The population size of GA_CAN during the evaluation was varied between 100 and 600 and the best result was chosen. The evaluation shows that our OPTIM_HWeight and OPTIM_PRO provides an optimization value at least as good as GA_CAN. With an increasing number of abstract services (the number of concrete services/possible realizations per abstract service is constantly 100) our algorithms provide even better optimization results than GA_CAN (e.g. above 25 abstract services, f_{obj} is about 7% better). Thus, according to our evaluation, the more possible combinations exist, the better are the optimization results of OPTIM_HWeight and OPTIM_PRO in comparison to GA_CAN.

Table C

Nr. Abs. Serv	Fobj_H/ Fobj_G	Fobj_P/ Fobj_G
5	100.00%	100.00%
10	101.22%	101.22%
15	100.01%	100.01%
20	103.29%	103.36%
25	110.78%	110.94%
30	107.31%	107.31%
35	107.10%	107.31%

Figure 10.5: Optimization of the Objective Function

10.2 Prediction Algorithms

In this section, we evaluate our context-aware prediction algorithm with regard to *prediction quality* and *computation time*. For this purpose, we conducted several experiments based on a dataset provided by Zheng [112, 111, 17] with measurements for response time and throughput for real Web services located in different countries from all over the world. The dataset consists of measurements for 5825 real-world services, which were discovered from the Internet from UDDI or from several Web service portals (e.g. *webservicelist.com*, *webservicex.net*, *seekda.com*). The WSDL files

were downloaded and client-side Java code was generated with *Axis2* [112] in order to invoke the services. In the dataset, invocations from 339 users were recorded.

10.2.1 Evaluation Methodology and Setup

In order to evaluate the performance of our prediction algorithm with regard to *prediction quality*, we first had to pre-process the dataset of Zheng to enrich it with additional information regarding the distances between the locations of the services and the users. In the dataset of Zheng, the user locations are indicated as both countries and GPS coordinates, and the service locations are indicated solely as countries. Thus, the service locations are not specified that precise as the user locations, and this issue may bring some distortion into our predictions. For the locations of the services, we simply used the GPS coordinates of the capitals of the corresponding countries. The GPS coordinates are then used to calculate the distances between services and users.

It has also to be considered that the dataset of Zheng contains a number of data points where for response time or throughput there were assigned values of -1, which we interpret as missing values so that they can not be considered in the prediction. To minimize such entries, we selected in our dataset only services that have less than 30 entries per service assigned with values of -1 for response time or throughput. The dataset for our experiments contains 339 users from 31 countries and 1101 services from 42 countries.

For our experiments we used sub-sets of the enriched dataset from Zheng. Each sub-set is divided into two parts: data for training, i.e. calculation of the regression coefficients, and test data, i.e. to compare predicted values with measured values. We measured the prediction quality by computing the *Mean Absolute Error* (MAE):

$$MAE = \frac{\sum_{i,j} \left| y_{i,j} - \hat{y}_{i,j} \right|}{n} \tag{10.3}$$

where $y_{i,j}$ is the predicted value and $\hat{y}_{i,j}$ the recorded value.

We compared the MAE for response time and throughput of our prediction algorithm with a very simple approach, i.e. taking the average values over the users for a service, which we call in the following paragraphs *IMEAN*. Our intention is not to propose an algorithm that outperforms all other known prediction algorithms, but to show that even with a very simple and efficient algorithm significant improvements with regard to prediction quality over IMEAN can be achieved and that predictions can benefit from considering context information.

It has to be expected that prediction quality is highly impacted by the availability of training data, i.e. number of services and users and the portion of measurements recorded for them. Therefore, we vary these factors and assess their influence on the prediction quality. We choose a number n, indicating the number of services and users, and a number p for the percentage of the recorded measurements. With the help of these two numbers, a matrix of n services and n users is randomly chosen from the dataset, and only p percent of the data are used for training. The remaining $100 - p$ percent of the data are predicted. As the dataset for training is selected randomly, we repeat each experiment 20 times.

We also evaluate our prediction algorithm with regard to computation time in order to show its applicability in real world scenarios. Computation time was measured for calculation of the regression coefficients and for the actual predictions. In the same way as for assessing the prediction quality, we created sub-sets of the dataset of Zheng. This time, however, we keep p constant at 20% and only vary the number of services and users. All measurements have been performed with our Java-based prototype implementation on a Lenovo T510 notebook with an Intel Core i7 processor with 2.67GHz and 4GB RAM. Each experiment has been repeated 10 times.

10.2.2 Experiments

In the following paragraphs, we describe the three experiments that we conducted: experiment 1 and 2 for evaluating the prediction quality and stability, and experiment 3 where we show the measured computation time for the calculation of regression coefficients and the predictions.

10.2.2.1 Experiment 1- Increasing Matrix Density

In the first experiment, we evaluated the impact of the matrix density, i.e. the percentage of values of the matrix available for the training, on the prediction quality of response time and throughput. We compare the MAE of the predictions computed with the function service, function user, the combi function and IMEAN. We kept the number of services and users constant at 150 and increased the matrix density from 10% to 50%. Figure 10.6 shows the average MAE for response time and Figure 10.7 the average MAE for throughput for the 20 runs and the different prediction approaches.

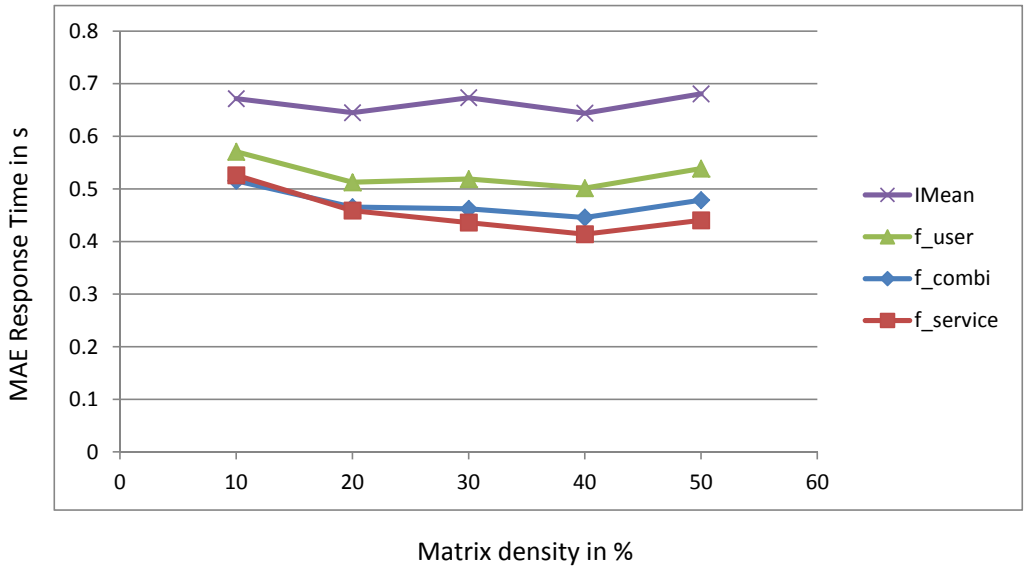

Figure 10.6: Experiment 1 - MAE Response Time

From Figure 10.6 we can observe that the algorithm based on the service function achieves the best prediction results for response time. It achieves an improvement of

prediction quality in terms of MAE of up to 35% compared to IMEAN. It even performs better than the combined function for matrix densities from 20% to 50%. The trend of the prediction quality almost remains the same for matrix densities from 20% to 50% for all prediction algorithms. Only for a small matrix density of 10% the prediction results are slightly worse. The best prediction quality was reached at a matrix density of 40% and the lowest MAE value was reached with the $f\_service$ function. The prediction quality of IMEAN is not affected by the matrix density in the evaluated range.

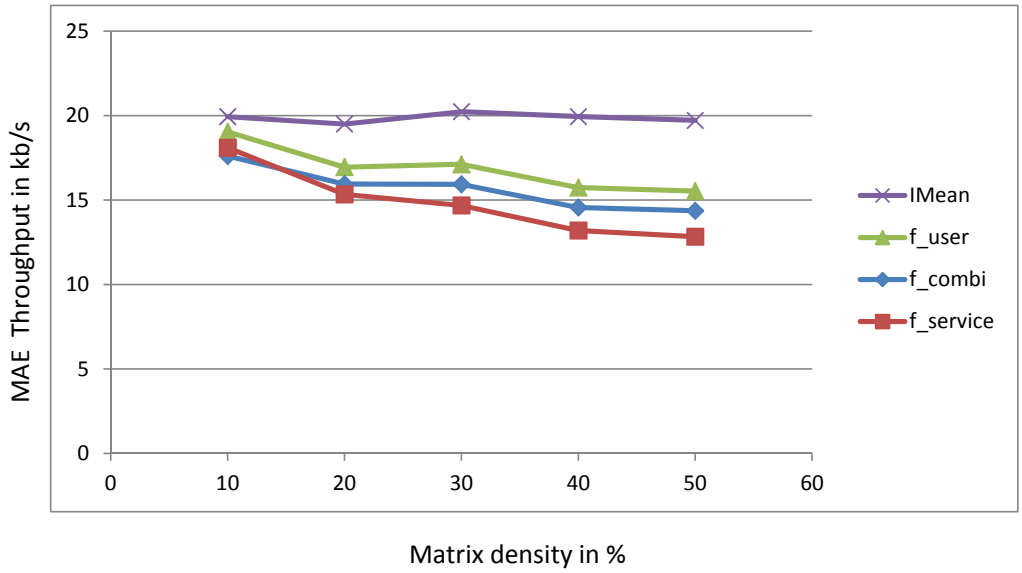

Figure 10.7: Experiment 1 - MAE Throughput

Figure 10.7 shows the average MAE for the different prediction functions and increasing matrix densities for throughput. They look very similar to the results for response time. Here too, the algorithm based on the service function outperforms the other algorithms. Compared to IMEAN it again achieves an improvement of the average MAE of about 35% at matrix density 50%. In contrast to the results for response time, prediction quality improves for throughput with increasing matrix density for all regression-based prediction algorithms.

The matrix density may not only impact the prediction quality with respect to the average MAE over the 20 runs, but also the stability of the predictions. In order to assess this aspect, Figure 10.8 and Figure 10.9 provide some more detailed information on the statistics of the results using box plots. The blue bars indicate the median of the MAE over the 20 runs. The boundaries of the boxes highlight the range from the 25th percentile to the 75th percentile, and therefore correspond to the middle 50% of values. The vertical line shows the range from the minimum to the maximum value.

Figure 10.8 shows that stability of the prediction quality for response time slightly improves for the regression based prediction algorithms with increasing matrix densities. For example, for function service the box for the middle 50% of the MAE values covers a range of length 0.09s for matrix density 10%, whereas for matrix density 50% it only covers a range of length 0.055s.

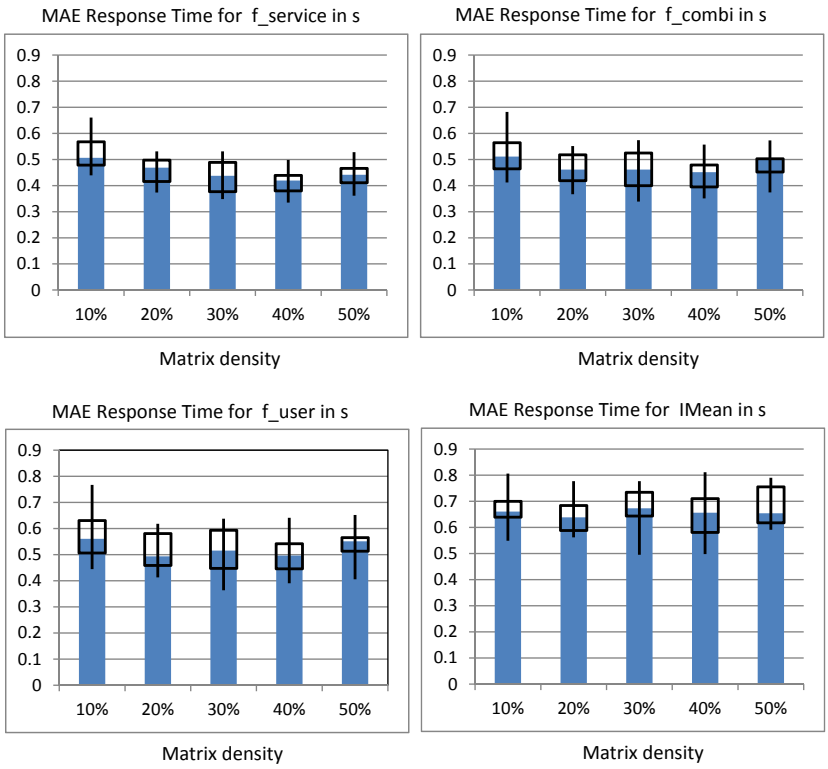

Figure 10.8: Experiment 1 - Boxplots MAE Response Time

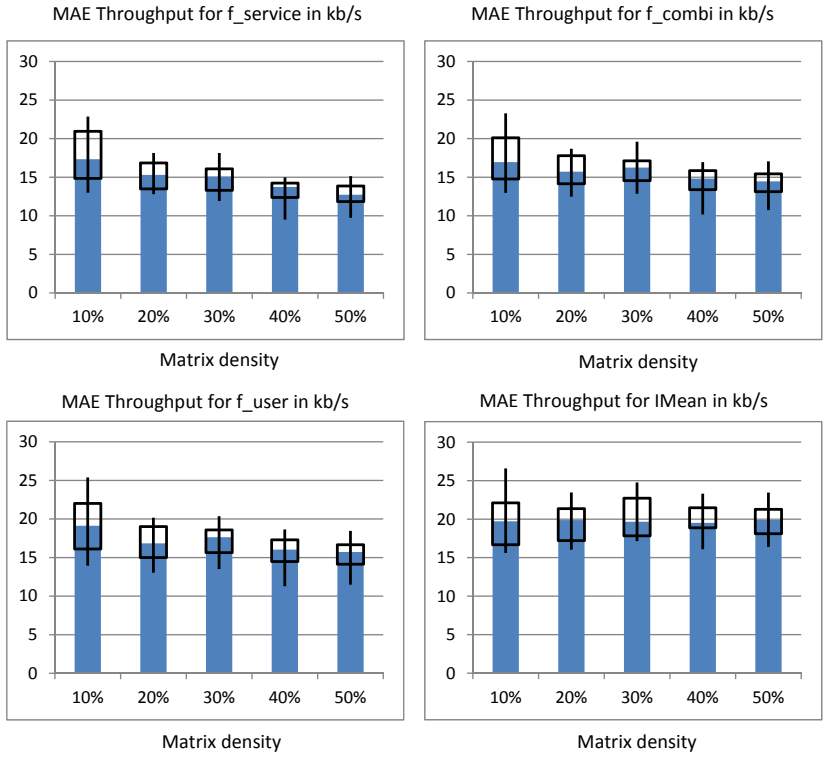

Figure 10.9: Experiment 1 - Boxplots MAE Throughput

From Figure 10.9 it can be observed that also for throughput stability of the regression-based prediction algorithms improves with increasing matrix density. Whereas for matrix density 10% and function service the box covers a range of length 6.1kb/s, it only covers a range of 2.0kb/s for matrix density 50%.

10.2.2.2 Experiment 2 - Increasing Matrix Size

In experiment 2, we analyze the prediction quality and its stability for response time and throughput when increasing the size of the matrix. Similarly to experiment 1, we randomly select entries from the dataset as training data and vary the size of the matrix. For each of the matrix dimensions we make predictions for 20 runs and compute the average MAEs. The matrix density is kept constant at 20% and the matrix size is varied from 70 users and 70 services to 339 users and 350 services.

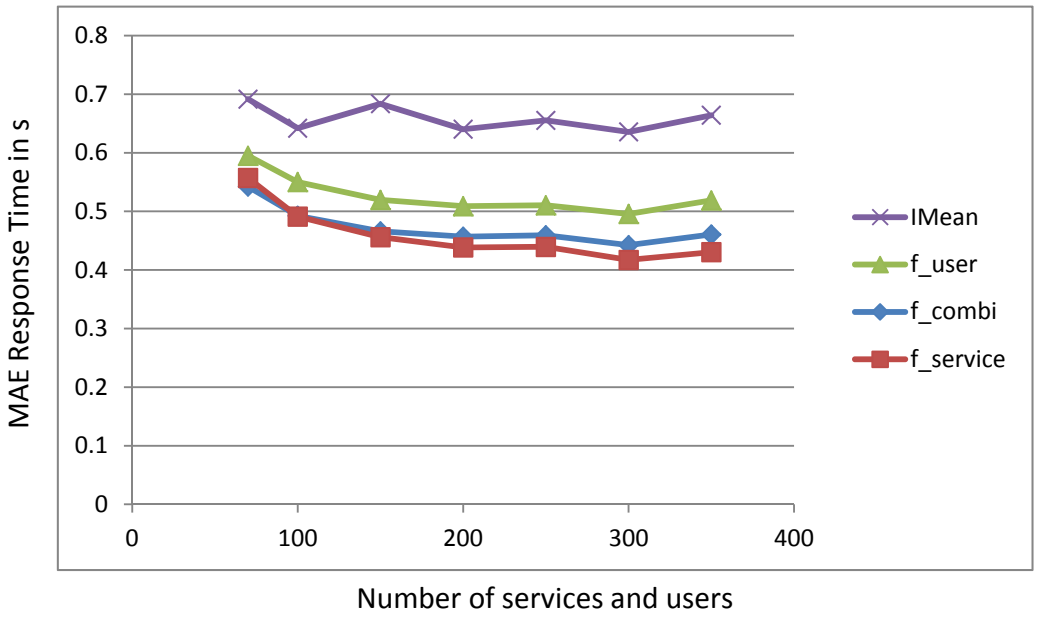

Figure 10.10: Experiment 2 - MAE Response Time

Figure 10.10 and Figure 10.11 present the average MAE values for the 20 runs for response time and throughput for the different regression functions and IMEAN. It can be observed that both for response time and throughput the prediction quality with the regression functions increases (MAE getting lower) with increasing matrix size from 70 to 200 users and services. This is to be expected as with a greater matrix dimension there are more users and services available and more entries are considered for computing the coefficients of the regression functions. Beyond the matrix size of 200 the prediction quality remains quite the same, which means that more data points do not result in a further improvement of the regression functions. Similarly to experiment 1, the MAE for IMEAN is not affected by the size of the matrix and changes for response time only slightly in the interval between 0.63s and 0.69s.

Observing Figure 10.10, the predictions of response time with the service functions are the best among the tested regression functions and improve from a MAE of 0.56s for

matrix size 70 to a MAE of 0.42s for matrix size 300, thus achieving up to 34% better predictions (at matrix size 300). For matrices with a size varying from 70 to 150, the predictions with $f\_service$ and f_combi reach a similar good MAE both for response time and throughput (see Figure 10.10, Figure 10.11).

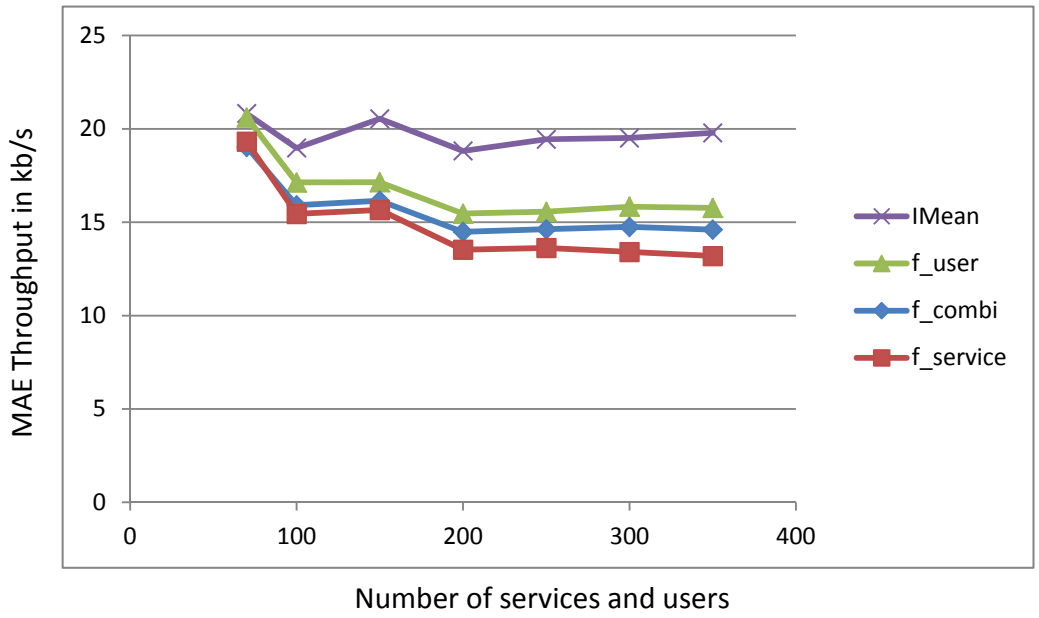

Figure 10.11: Experiment 2 - MAE Throughput

For matrix dimensions greater than 150, the response time and throughput predictions with $f\_service$ are the best from all the regression functions, having slightly lower MAE values than the predictions with f_combi. Regarding throughput, the lowest MAE value of 13.18kb/s was reached with the $f\_service$ function for a matrix consisting of 339 users and 350 services and the value represents an improvement of 33% compared to the MAE value of IMEAN.

Similarly to experiment 1, we observe that the stability of the prediction with the regression functions improves with increasing the matrix size. This trend for response time and throughput is shown in Figure 10.12 and Figure 10.13.

For example, with $f\_service$ the box for the middle 50% of the MAE values for response time covers a range of length 0.10s for 70 services and users, whereas it only covers a range of length 0.031s for 300 services and users. Similarly, the box for the MAE values for throughput with $f\_service$ covers a range of length 4.64kb/s for 70 services and users, whereas it covers only a range of 1.4kb/s for 339 users and 350 services. For both response time and throughput, also the range between the minimum and maximum MAE values gets smaller with increasing matrix size for all regression functions.

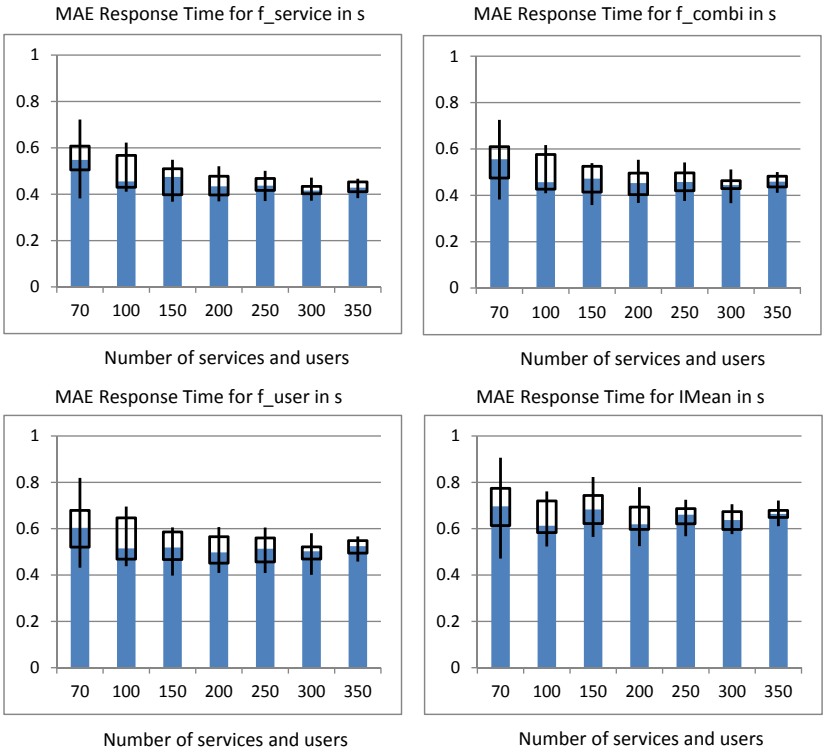

Figure 10.12: Experiment 2 - Boxplots MAE Response Time

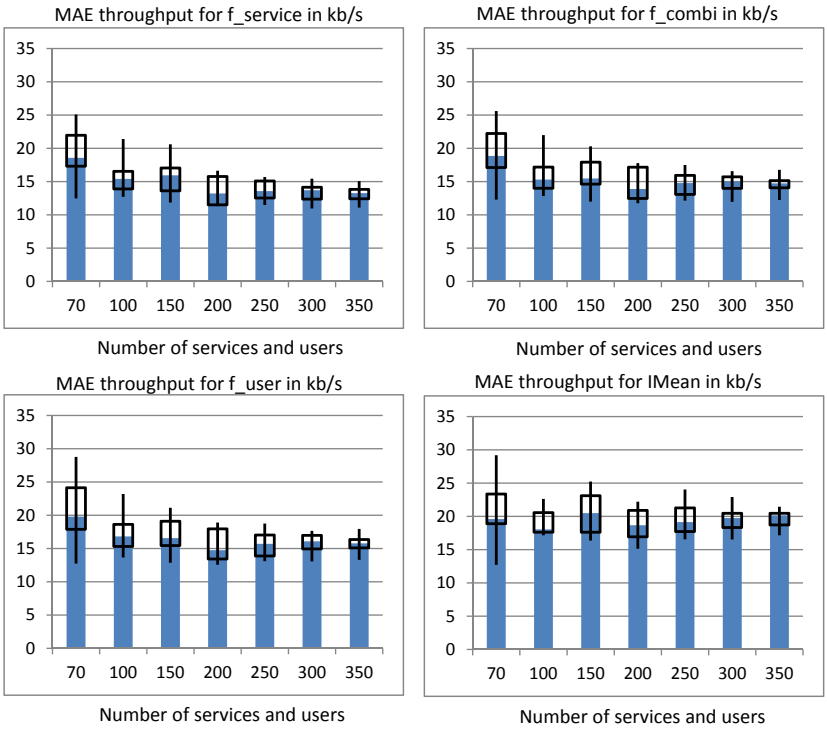

Figure 10.13: Experiment 2 - Boxplots MAE Throughput

10.2.2.3 Experiment 3 - Computation Time

In experiment 3, we measure the computation time for calculating the regression coefficients for response time prediction and measure the time needed for the predictions. We vary the matrix size from 70 services and users to 350 services and 339 users. The matrix density is kept constant at 20%. This means that 20% of the matrix are used for calculation of the regression coefficients and the remaining 80% of the matrix are predicted.

For coefficient calculation, the following steps have to be performed and are included in the measurement of the computation time:

- multiple linear regression with all observations
- calculation of the Cook's distance and filtering outliers
- multiple linear regression without filtered outliers
- calculation of the coefficient of determination

For prediction, only the regression coefficients have to be fetched from the repository, and the service, user or combi function has to be applied depending on the used prediction method. When using the service function, also the coefficients of determination of the user and the service function have to be fetched.

We perform 10 runs for each matrix dimension and build the average computation time of the 10 runs. The measured time values for coefficient calculation are listed in Figure 10.14.

Computation Time for Coefficient Calculation in ms						
	Average			Sum for matrix		
Matrix Size	f_user	f_service	f_combi	f_user	f_service	f_combi
70x70	0.0705	0.0680	0.1091	4.93	4.76	7.63
100x100	0.0925	0.0896	0.1526	9.24	8.96	15.26
150x150	0.1237	0.1277	0.2455	18.56	19.15	36.82
200x200	0.1669	0.1639	0.2901	33.38	32.75	57.92
250x250	0.1923	0.1908	0.3604	48.07	47.69	90.02
300x300	0.2291	0.2261	0.4228	68.71	67.83	126.66
339x350	0.2779	0.2621	0.4475	94.20	91.75	153.53

Figure 10.14: Experiment 3 - Computation Time for Coefficient Calculation

The left side of Figure 10.14 shows the average time required for the coefficient calculations for the different functions with respect to different matrix sizes, whereas the right side lists the times required for calculating the coefficients for the whole matrix. The matrix size (in conjunction with the selected matrix density of 20%) impacts the number of points available for the multiple linear regression. It can be observed that calculating the regression coefficients for function user and function service approximately needs the same time, whereas the calculation of the coefficients for the combi function almost needs twice as much time. This is quite obvious, as the combi function internally consists of a combination of the function user and the function service, and therefore the coefficients for both functions have to be calculated.

We also observe that the average time needed for coefficient calculations linearly increases with the matrix size, i.e. with the number of observations available for performing the multiple linear regression. This is also illustrated in Figure 10.15.

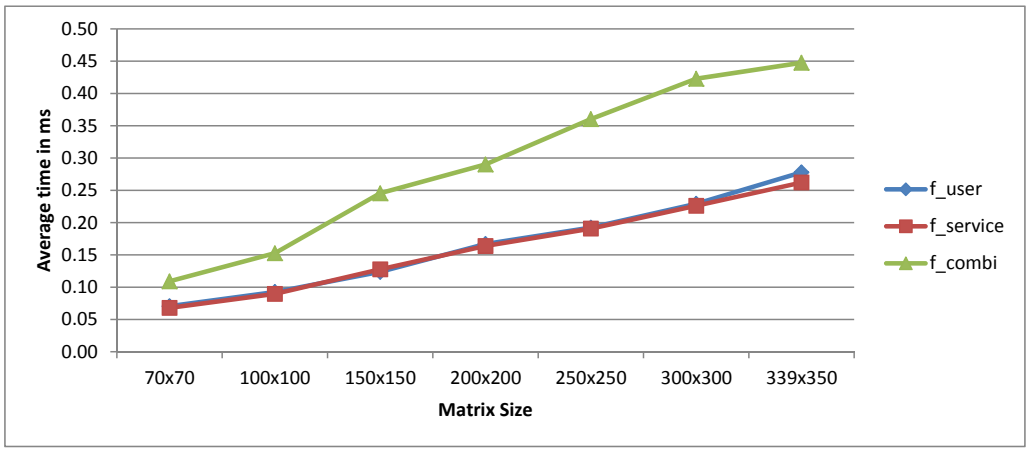

Figure 10.15: Experiment 3 - Average Computation Time for Coefficient Calculation

In the following paragraphs, we review the different steps included in the time measurement for coefficient calculation and show that this observation conforms to the complexity of the different involved steps.

- **Multiple Linear Regression with All Observations**

  This step mainly consists of creating the matrices X and Y containing the observations and performing the matrix calculation $(X^TX)^{-1}X^TY$. Here X is a $(n,3)$-matrix, with n being the number of observations, and Y is a $(n,1)$-matrix. Consequently, (X^TX) is a $(3,3)$-matrix and for each of the nine entries a sum of n products has to be calculated. Therefore, the complexity of calculating (X^TX) is linear in the number of observations. Inverting a $(3,3)$-matrix shows constant complexity. The calculation of $(X^TX)^{-1}X^T$ consists of a multiplication of a $(3,3)$-matrix and a $(3,n)$-matrix which results in a $(3,n)$-matrix. This means that $3n$ entries have to be calculated, each being a sum of three products. Thus, also this step is linear in the number of observations. Finally, a multiplication of a $(3,n)$-matrix with a $(n,1)$-matrix is required to get the three regression coefficients. For each of the coefficients, a sum of n products has to be calculated. In summary, we can conclude that calculating the regression coefficients shows a complexity linear in the number of observations.

- **Calculation of the Cook's Distance and Filtering Outliers**

  The calculation of the Cook's distance involves the calculation of the studentized residual r for each of the points and the computation of the hat matrix $H = X(X^TX)^{-1}X^T$. Examining the formula for the studentized residual (see Equation 8.11) and assuming the availability of the of the hat matrix and of $\hat{\sigma}^2$, calculation of the studentized residual for a point has constant complexity. From the formula for $\hat{\sigma}^2$ (see Equation 8.12), it can easily be verified that its complexity is linear in the number of available observations. Determining the hat matrix

$H = X(X^TX)^{-1}X^T$ consists of a multiplication of X, which is a $(n, 3)$-matrix, with $(X^TX)^{-1}X$, which is a $(3, n)$-matrix. Using the insights from the previous bullet point, the complexity of calculating $(X^TX)^{-1}X$ is linear with respect to the number of observations, but the calculation of the final matrix multiplication results in a (n, n)-matrix with n^2 entries. However, only the n elements on the diagonal of the hat matrix are required. Thus, also the complexity of calculating the Cook's distance is linear with regard to the number of observations.

- **Multiple Linear Regression without Filtered Outliers**

  From a complexity point of view, this step equals the step from the first bullet point but with fewer observations if outliers have been filtered.

- **Calculation of the Coefficient of Determination**

  From Equation 8.8 it can easily be seen, that also the complexity of this step is linear with regard to the number of observations.

In total, we can conclude that the calculation of the regression coefficients can be achieved with a complexity linear in the number of observations, which is inline with the results of our experiments.

Computation Time for Prediction in ms						
	Average			Sum for matrix		
Matrix Size	f_user	f_service	f_combi	f_user	f_service	f_combi
70x70	0.0016	0.0015	0.0026	6.30	5.68	10.31
100x100	0.0017	0.0018	0.0027	13.22	13.91	21.58
150x150	0.0018	0.0019	0.0029	32.37	33.38	52.26
200x200	0.0018	0.0019	0.0030	55.56	60.24	94.24
250x250	0.0019	0.0018	0.0049	91.75	90.65	237.40
300x300	0.0018	0.0018	0.0043	125.51	125.18	302.57
339x350	0.0018	0.0017	0.0036	166.04	162.20	335.17

Figure 10.16: Experiment 3 - Computation Time for Prediction

Figure 10.16 lists the results of our experiments with regard to the computation time needed for the actual predictions. The left side of Figure 10.16 shows the average computation times required for the predictions, whereas the right side lists the times required for performing the predictions for the whole matrix. The numbers reveal that predictions with our approach can be performed very fast, i.e. in a few microseconds. Just as for the calculation of the regression coefficients, the average times for function service and function user are almost the same, whereas predictions with the combi function require twice as much time as the predictions with only the function user or only the function service. It is also obvious that the average prediction times are independent from the matrix size, whereas the prediction times for the whole matrix grow quadratically with the number of users and the number of services (assuming a quadratic user-service matrix). This is due to the fact that the number of entries to be predicted also grows quadratically.

In summary, our experiments reveal that calculating the regression coefficients and making the actual predictions can be performed very fast. Also the complexity of our approach is linear with regard to the available observations. Thus, our context-aware prediction approach based on linear regression is well-suited for real-world applications.

11 Conclusions

Distributed business applications are often realized as service compositions where Web services from different partners cooperate to accomplish a more complex business goal. In order to fulfill clients' requirements, the Web service compositions need to perform at adequate QoS levels. Another important aspect to be considered is the fulfillment of the users' expectations so that the users are satisfied with both performance and functionalities of the composition.

The degree of users' satisfaction is reflected by the Quality of Experience (QoE). Even when undesired situations occur, such as a network failure or services becoming unavailable, the service composition has to maintain adequate QoS and QoE levels. The malfunction of one single service inside the service composition may spoil the behavior of the entire service process.

Thus, the composition has to be properly managed so that its clients do not become aware of such problems. Monitoring and managing QoS and QoE are crucial tasks to ensure the success of the service process. In this thesis, we considered QoS dimensions such as response time, availability, reliability and throughput, and also the QoE. We presented a comprehensive solution which contains adequate mechanisms for managing QoS and QoE of Web service compositions at runtime. In the following, we review the identified requirements and the corresponding solutions.

11.1 Requirements and Solution Summary

1. **Flexible Monitoring**

   In order to be able to react to undesired service levels, the service composition needs to be monitored regarding its quality behavior. It has to be possible to easily add new QoS dimensions to the monitoring. When monitoring a service process, the process description should not have to be altered to include monitoring artefacts.

   Our solution for a flexible monitoring of the service composition was described in Chapter 6. We presented the *QoS measurement algorithm* that aggregates the QoS of the service composition out of the QoS of the building blocks. New QoS dimensions may be added by the use of plug-ins. Monitoring is performed with the help of sensors that signal the begin and end of an activity. These monitoring artefacts are defined in separate files, and thus the BPEL description file has not to be altered for this purpose. We also compared the measurement algorithm with the estimation algorithm and showed for which situation each of the algorithms is suited.

2. **Managing QoS and QoE**

   In case the QP requirements for the Web services or the Web service composition are not met, adequate reactions need to be specified and the deviations have to be handled properly.

   We designed and developed the *BPRules language* for managing QoS and QoE of Web services and Web service compositions. The BPR rules are specified by the business analyst and define what actions should be undertaken when certain QoS or QoE values are detected. BPR rules are defined in extra files separately from the process description and are deployed and executed by our BPR framework. Among the management actions supported by BPRules there are: starting or stopping the process, selecting new services to improve the QoS behavior of the process, and selecting services for user groups characterized by certain context data. BPRules also facilitates the generation of different kinds of reports which help the business analyst to get a more detailed view on the process behavior.

3. **Service Selection**

   In current service markets, multiple services are available that have the same functionality but perform at different service levels. Therefore, an abstract service of the composition can be realized by multiple concrete services. Thus, services with adequate QoS and QoE levels need to be selected so that the service composition fulfills the desired QoS and QoE constraints and optimizes an objective function.

   In Chapter 7 we presented three *selection algorithms*: $OPTIM\_S$, $OPTIM\_PRO$ and $OPTIM\_HWeight$. The $OPTIM\_S$ algorithm allows different types of search, a local, a global (brute-force) and a heuristic search. This is possible by calling the algorithm with different parameters and it gives the possibility to trade computation time off against the optimality of the solution. The $OPTIM\_HWeight$ algorithm is based on the $OPTIM\_S$ algorithm and uses a special heuristic function which realizes a gradient ascent. The $OPTIM\_PRO$ algorithm uses priority factors to find a near-optimal solution.

4. **QoS and QoE Predictions**

   The advertised QP values may differ from the values experienced by the users. Context dimensions of users and services may also influence the QP values. Thus, considering context dimensions in the prediction of QP would bring additional benefits to the selection of services.

   In Chapter 8, we presented CoFee, a module of the BPR framework, which collects QP feedbacks and is able to make predictions for unknown user-service constellations based on the collected feedbacks. It is also able to incorporate context information in its predictions. We exemplarily showed how response time and throughput is predicted. The predictions are based on linear regression and the corresponding functions also consider the distances between the locations of users and services and the average values of response time and throughput. We showed how the CoFee module is integrated in the BPR framework and how it interacts with the Service Selection module in order to select appropriate services.

5. **A QP Management Solution**

   The solutions to the previously mentioned requirements have to be combined into a comprehensive QP management framework. The business analyst has to be supported in his work with adequate tools for managing the service processes.

   For this purpose, we designed and developed the BPR framework. Our framework consists of several modules and integrates the presented monitoring approach, the specification and execution of BPR rules, the selection of services, and CoFee into a QP management solution. In Chapter 9, we described the architecture of the BPR framework and showed how the modules work together in order to achieve the QP management.

11.2 Contributions

The main contribution of this work resides in providing a comprehensive solution for QoS and QoE management for Web service compositions. Our solution includes a flexible monitoring approach, the BPRules language that provides novel features, new algorithms for the selection of services, and algorithms for QP predictions on the basis of context data. The *BPR framework* combines all the new features and contributions to a comprehensive management solution.

Our *monitoring approach* has its focus on *flexibility*. New QoS dimensions may easily be added to the monitoring. This is possible by the use of plug-ins for the aggregation functions and the value retrieval functions. We showed how automated deployment is done, by automatically generating sensors for the process in extra files. Thus, the monitoring artefacts are kept separate from the BPEL description file. We presented the *measurement algorithm* which is applied for the QoS computation of running or completed process instances.

We designed and developed the *BPRules language*, that includes novel features to become aware of and to overcome possible QP deviations. BPRules offers the possibility to query different types of QP: the monitored QoS and the assigned QoE, the estimated QP and the predicted QP. These values may be queried for a single Web service, for sections or for the entire composition. We are able to consider the QP behavior of running instances (instance-set handling, state querying) but also of instances which are already terminated. For managing the service process, BPRules provides a set of actions which are indispensable for a successful execution of a service process. Among the offered actions are: a flexible service selection, the dynamic rule set change and the generation of different kind of reports which provide the business analyst a good picture of the process behavior. BPRules has its focus on the service selection and replacement. Therefore, this action covers many detail aspects. The service selection action permits to seamlessly integrate multiple service selection algorithms depending on the number of abstract services and the number of available service candidates. The business analyst may specify the QP requirements and the objective function to be optimized for the service selection. BPRules offers the possibility to specify user groups with certain context properties. Further, these properties are used for a context-aware service selection tailored for the user groups.

Our QP management solution includes novel and efficient *algorithms for the selection of services*. The proposed algorithms, $OPTIM\_S$, $OPTIM\_PRO$ and $OPTIM\_HWeight$ are applicable also for non-linear objective functions. The evaluation showed that $OPTIM\_PRO$ and $OPTIM\_HWeight$ outperform a genetic algorithm from related work [39]. $OPTIM\_PRO$ was the fastest algorithm and needed in average about 22% of the time of the genetic algorithm. The $OPTIM\_HWeight$ algorithm needed in average about 30% of the time of the genetic algorithm. Both our algorithms even achieved better values for the objective function (up to 7% better) for cases with high number of combinations.

We presented new approaches for *context-aware QP predictions*. In contrast to other related works, we consider not only the location information of users and services but also discuss other context dimensions which may influence the QoE. Relevant context dimensions are identified by statistical approaches. The predictions are used to make a context-aware service selection for specified user categories with certain context properties.

11.3 Outlook and Future Work

Currently, the BPR framework assumes stateless Web services which are bound into the service compositions. The integration of stateful Web services poses some further challenges since also the states of the services need to be considered. Rollback mechanisms for the services may be required in order to appropriately handle sessions when performing management actions. Stateful Web services would also have an impact on the service selection algorithms as several activities related to the same Web service may spread over the business process. Currently, the activities are considered to be independent and thus, also the services realizing the different abstract services are selected independently of each other.

The BPR framework could be extended with the monitoring and negotiation of SLAs. Also it could be investigated whether parts from BPR rules (e.g. the condition part) may be generated automatically from an SLA.

In the BPR framework, we currently monitor each activity of the process. The monitoring artefacts produce extra overhead. One possibility to reduce the monitoring overhead would be to specify different rule sets with different monitoring levels. Thus, if the process behaves well, a monitoring of the root activity would be sufficient. If the process behavior gets worse and the rule set changes, then the monitoring gets more detailed and also single sections are explicitly monitored. Web services can also be integrated into mobile applications. In this case, reducing the monitoring overhead would also be useful since the resources on mobile devices are often limited.

In our approach, the business analyst is responsible for specifying the BPR rules. In particular, he has to ensure that he does not define contradictory rules. Extra tool support should be offered to the business analyst that exhibits warnings when he specifies contradictory rules or even proposes solutions to resolve the conflicts.

The main role addressed by the BPR framework is that of the business analyst. However, besides the business analyst more participants are involved, like the developers of the

service composition or the providers of the Web services. The service providers are also interested in the behavior of their Web services. Thus, e.g. the report functionality of the BPR framework may be extended for other roles as well.

In future work, BPRules could be extended with other adaptation mechanisms not only considering the replacement of services but also supporting more complex adaptations, like the dynamic adaptation of sections of the process. This assumes to change the structure of the BPEL process and would also affect the selection of services.

The business analyst is supposed to know for which user categories he desires to build context-aware service compositions. He declares the context dimensions that characterize the user categories. However, more elaborate tool support can be provided to automatically detect the user categories based on the monitored QoS, the assessed QoE values and the available context data. The development of such algorithms requires the existence of real datasets to verify their viability.

We discussed the prediction of QoE that may be influenced by context dimensions characterizing the users. However, this approach has to be verified with a real dataset. Currently the approach is based on memory-based collaborative filtering. It should also be analyzed whether a model-based approach or an approach utilizing matrix factorization can be used for the prediction of QoE.

List of Figures

List of Tables

A Appendix

A.1 BPRules XSD Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
3    <!--
4      define the root element bprules for the BPR-document
5    -->
6    <xs:element name="bprules" type="bprulesType"/>
7    <!--
8      elements in the BPR-document
9    -->
10   <xs:complexType name="bprulesType">
11     <xs:sequence>
12       <xs:element name="include-bprDocs" type="includebprdocsType" minOccurs="0"/>
13       <xs:element name="constants" type="constantsType" minOccurs="0"
14                   maxOccurs="unbounded"/>
15       <xs:element name="sections" type="sectionsType" minOccurs="0"/>
16       <xs:element name="rulesets" type="rulesetsType" maxOccurs="unbounded"/>
17     </xs:sequence>
18     <xs:attribute name="id" type="xs:ID"/>
19     <xs:attribute name="processid" type="xs:string" use="required"/>
20   </xs:complexType>
21   <!--
22     Attributes that may be used for multiple elements
23       - id - uniquely identifies an element
24       - select - may reference the element identified by the id
25       - applysection - the quality parameters are computed for the section
26       - applyservice - the quality parameters are computed for a service
27       - applysection and applysection are used for the elements condition, expression,
28         qualityparam, select-services, select-services-context
29       - default value - in absence of applysection and applyservice - the quality parameters
30         are computed for the process
31       - applyfunction - is only applicable for the monitored quality values of a set
32         of instances (e.g. average qp of the instances)
33   -->
34   <!--
35     define sections
36   -->
37   <xs:complexType name="sectionsType">
38     <xs:sequence>
39       <xs:element name="section" type="sectionType" maxOccurs="unbounded"/>
40     </xs:sequence>
41   </xs:complexType>
42   <!--
43     define section: A section is defined by one structured activity or is bounded by two activities
44       - the activities are referenced by their names as specified in the BPEL process
45       - attribute id is required for a section. It enables to refer to the section
```

```
46        (e.g. by the applysection attribute within the select −services action)
47      −−>
48      <xs:complexType name="sectionType">
49        <xs:sequence>
50          <xs:element name="activity" type="xs:string" maxOccurs="2"/>
51        </xs:sequence>
52        <xs:attribute name="id" type="xs:ID" use="required"/>
53      </xs:complexType>
54      <!−−
55        include other bpr−documents
56        doc − contains the location and name of the bpr−document to be included
57      −−>
58      <xs:complexType name="includebprdocsType">
59        <xs:sequence>
60          <xs:element name="doc" type="xs:string" maxOccurs="unbounded"/>
61        </xs:sequence>
62      </xs:complexType>
63      <!−−
64        define constants
65      −−>
66      <xs:complexType name="constantsType">
67        <xs:sequence>
68          <xs:element name="constant" type="constantType" maxOccurs="unbounded"/>
69        </xs:sequence>
70      </xs:complexType>
71      <!−−
72        define rulesets
73      −−>
74      <xs:complexType name="rulesetsType">
75        <xs:sequence>
76          <xs:element name="ruleset" type="rulesetType" maxOccurs="unbounded"/>
77        </xs:sequence>
78      </xs:complexType>
79      <!−−
80        define ruleset
81          − period − the period of time for selecting the instances to be evaluated.
82            It targets only the monitored quality parameters.
83          − active − the ruleset is currently active or inactive (temporarily ignored)
84          − the constants which are defined in the ruleset can be queried only within the ruleset
85            (as local constants)
86      −−>
87      <xs:complexType name="rulesetType">
88        <xs:sequence>
89          <xs:element name="evaluate" type="evaluateType" minOccurs="0"/>
90          <xs:element name="constants" type="constantType" minOccurs="0"
91                      maxOccurs="unbounded"/>
92          <xs:element name="period" type="periodType" minOccurs="0" maxOccurs="1"/>
93          <xs:element name="rule" type="ruleType" maxOccurs="unbounded"/>
94        </xs:sequence>
95        <xs:attribute name="id" type="xs:ID"/>
96        <xs:attribute name="active" type="xs:boolean" use="optional"/>
97      </xs:complexType>
98      <!−−
99        define evaluate
100       − defines when the ruleset is evaluated. There are two possibilities:
101       1. Attribute trigger ="periodic"
102         Rules are triggered periodically (e.g. every 1 minute, every two hours)
103         Example: every three minutes
```

```
104         <evaluate trigger="periodic" unit="minutes">3</evaluate>
105     2. Attribute trigger ="once"
106       The rules are triggered only once, after the BPR-document was loaded or at a
107        specified date and time.
108       Example: <evaluate trigger="once">12-08-13T12:00:00</evaluate>
109     If <evaluate> is missing, the default value for evaluation is 'every minute'.
110   -->
111   <xs:complexType name="evaluateType">
112     <xs:simpleContent>
113       <xs:extension base="xs:string">
114         <xs:attribute name="trigger" type="triggerAttType"/>
115         <xs:attribute name="unit" type="unitTimeType"/>
116       </xs:extension>
117     </xs:simpleContent>
118   </xs:complexType>
119   <xs:simpleType name="triggerAttType">
120     <xs:restriction base="xs:string">
121       <xs:enumeration value="periodic"/>
122       <xs:enumeration value="once"/>
123     </xs:restriction>
124   </xs:simpleType>
125   <!--
126     define rule
127     - there are two possibilities:
128       1. define new rule with an id. The condition can be omitted. In this case it
129          is assumed to evaluate to true. The rule contains one action element.
130       2. reference an existing rule by using the select attribute, in this case condition,
131          action and id are omitted.
132   -->
133   <xs:complexType name="ruleType">
134     <xs:sequence minOccurs="0">
135       <xs:element name="condition" type="conditionType" minOccurs="0" maxOccurs="1"/>
136       <xs:element name="action" type="actionType"/>
137     </xs:sequence>
138     <xs:attribute name="id" type="xs:ID" use="optional"/>
139     <xs:attribute name="select" type="xs:string" use="optional"/>
140   </xs:complexType>
141   <!--
142     define condition
143     - the condition can be applied for the entire BPEL process (default), for a section
144       (with attribute applysection) or a single service (with attribute applyservice).
145   -->
146   <xs:complexType name="conditionType">
147     <xs:sequence minOccurs="0">
148       <xs:element name="constraints" type="constraintsType"/>
149     </xs:sequence>
150     <xs:attribute name="applysection" type="xs:string" use="optional"/>
151     <xs:attribute name="applyservice" type="xs:string" use="optional"/>
152   </xs:complexType>
153   <!--
154       define constraints
155       - select-instances - select instances with a certain state (e.g. CLOSED_COMPLETED,
156         FAULTED, etc.)
157       - instances-subset - defines how many instances from the set should fulfill the constraints.
158       - constraints can be linked by logical operators OR, AND, NOT.
159   -->
160   <xs:complexType name="constraintsType">
161     <xs:sequence minOccurs="0">
```

```
      <xs:element name="select-instances" type="processpropertiesType" minOccurs="0"/>
      <xs:element name="instances-subset" type="instancessubsetType" minOccurs="0"/>
      <xs:choice>
        <xs:element name="expression" type="expressionType"/>
        <xs:element name="and" type="BinaryConstraintsLogicOperatorType"/>
        <xs:element name="or" type="BinaryConstraintsLogicOperatorType"/>
        <xs:element name="not" type="UnaryConstraintsLogicOperatorType"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
    <xs:attribute name="select" type="xs:string" use="optional"/>
  </xs:complexType>
  <!--
    define period of time
    - period of time - for selecting the instances to be evaluated.
    - applies for the monitored quality parameters
    - there are two possibilities:
      1. AbsoluteTime - e.g. '23-09-12T10:50', to specify an absolute date
         two special strings can be used:
         'Current' - for the current time
         'ProcessStart' - for the date when the process was started
      2. RelativeTime with a specified unit, e.g. '-4 days' means from current time 4 days
         in the past
  -->
  <xs:complexType name="periodType">
    <xs:sequence>
      <xs:element name="interval" type="intervalType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <!--
    define interval type
  -->
  <xs:complexType name="intervalType">
    <xs:sequence>
      <xs:element name="begin" type="intervalBoundaryType"/>
      <xs:element name="end" type="intervalBoundaryType"/>
    </xs:sequence>
    <xs:attribute name="type" type="timeType"/>
  </xs:complexType>
  <!--
    define time type - distinguish between absolute and relative time period
  -->
  <xs:simpleType name="timeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="AbsoluteTime"/>
      <xs:enumeration value="RelativeTime"/>
    </xs:restriction>
  </xs:simpleType>
  <!--
    define interval boundary
  -->
  <xs:complexType name="intervalBoundaryType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="unit" type="unitTimeType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

```
<!--
  define time unit
-->
<xs:simpleType name="unitTimeType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="minutes"/>
    <xs:enumeration value="hours"/>
    <xs:enumeration value="days"/>
    <xs:enumeration value="months"/>
    <xs:enumeration value="years"/>
  </xs:restriction>
</xs:simpleType>
<!--
  define a constant
  - a constant can be used for example as a threshold value for the quality parameters
-->
<xs:complexType name="constantType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="type" type="Types"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!--
  define select -instances
  - allows selection of instances with a certain state or having a certain instance id
  - if more than one property is defined, then all the instances are selected which
    fulfill at least one of the specified properties
-->
<xs:complexType name="processpropertiesType">
  <xs:sequence minOccurs="0">
    <xs:element name="property" type="propertyType" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID"/>
  <xs:attribute name="select" type="xs:string"/>
</xs:complexType>
<!--
  define instance property
  - for the selection of instances with a certain state
    or having a certain instance id
  - possible process states are:
    - CLOSED
    - FAULTED
    - CLOSED_COMPLETED
    - CLOSED_FAULTED
    - CLOSED_CANCELLED
    - OPEN_FAULTED
    - OPEN_RUNNING
-->
<xs:complexType name="propertyType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="select" type="selectProcessPropertiesAttType"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="propertyCheckType">
```

```
278     <xs:simpleContent>
279       <xs:extension base="xs:string">
280         <xs:attribute name="select" type="selectProcessPropertiesCheckAttType"/>
281       </xs:extension>
282     </xs:simpleContent>
283   </xs:complexType>
284   <!--
285       selection of instances with a certain state or having a certain instance id
286   -->
287   <xs:simpleType name="selectProcessPropertiesAttType">
288     <xs:restriction base="xs:string">
289       <xs:enumeration value="state"/>
290       <xs:enumeration value="instanceid"/>
291     </xs:restriction>
292   </xs:simpleType>
293   <!--
294       refer to the state of instances within an expression
295   -->
296   <xs:simpleType name="selectProcessPropertiesCheckAttType">
297     <xs:restriction base="xs:string">
298       <xs:enumeration value="state"/>
299     </xs:restriction>
300   </xs:simpleType>
301   <!--
302     define instances-subset
303     - how many of the instances should fulfill the constraints. (e.g. MIN 30%, FORALL)
304     - applies only to the monitored quality values
305   -->
306   <xs:complexType name="instancessubsetType">
307     <xs:simpleContent>
308       <xs:extension base="instancessubsetFunctionValType">
309         <xs:attribute name="function" type="instancessubsetFunctionType"/>
310       </xs:extension>
311     </xs:simpleContent>
312   </xs:complexType>
313   <!--
314     define functions for specifying the subset of instances
315   -->
316   <xs:simpleType name="instancessubsetFunctionType">
317     <xs:restriction base="xs:string">
318       <xs:enumeration value="EXISTS"/>
319       <xs:enumeration value="MIN"/>
320       <xs:enumeration value="MAX"/>
321       <xs:enumeration value="EQUALS"/>
322       <xs:enumeration value="FORALL"/>
323     </xs:restriction>
324   </xs:simpleType>
325   <!--
326     value for the instances-subset functions
327   -->
328   <xs:simpleType name="instancessubsetFunctionValType">
329     <xs:restriction base="xs:string">
330       <xs:pattern value="\d%?"/>
331     </xs:restriction>
332   </xs:simpleType>
333   <!--
334     define expressions
335     - expressions may contain quality parameters, constants, values or functions of them,
```

```
336       which may be evaluated using a predicate (e.g. qualityparam < value)
337     - expressions may be linked with logical operators: and, or, not
338   -->
339   <xs:complexType name="expressionType">
340     <xs:sequence minOccurs="0">
341       <xs:choice>
342         <xs:element name="property-check" type="propertyCheckType"/>
343         <xs:element name="predicate" type="predicateType"/>
344         <xs:element name="and" type="BinaryExpressionLogicOperatorType"/>
345         <xs:element name="or" type="BinaryExpressionLogicOperatorType"/>
346         <xs:element name="not" type="UnaryExpressionLogicOperatorType"/>
347       </xs:choice>
348     </xs:sequence>
349     <xs:attribute name="id" type="xs:ID"/>
350     <xs:attribute name="select" type="xs:string"/>
351     <xs:attribute name="applysection" type="xs:string" use="optional"/>
352     <xs:attribute name="applyservice" type="xs:string" use="optional"/>
353   </xs:complexType>
354   <!--
355     predicate to evaluate a qualityParam, constant, value or a function of them
356   -->
357   <xs:complexType name="predicateType">
358     <xs:sequence minOccurs="1">
359       <xs:choice>
360         <xs:element name="qualityParam" type="qualityParamType"/>
361         <xs:element name="Function" type="functionType"/>
362       </xs:choice>
363       <xs:choice>
364         <xs:element name="qualityParam" type="qualityParamType"/>
365         <xs:element name="Function" type="functionType"/>
366         <xs:element name="constant" type="constantType"/>
367         <xs:element name="Value" type="xs:string"/>
368       </xs:choice>
369     </xs:sequence>
370     <xs:attribute name="type" type="predicateAttType" use="required"/>
371     <xs:attribute name="id" type="xs:ID"/>
372   </xs:complexType>
373   <!--
374     type of the predicate for comparing quality parameters, functions, constants, and values
375   -->
376   <xs:simpleType name="predicateAttType">
377     <xs:restriction base="xs:string">
378       <xs:enumeration value="Greater"/>
379       <xs:enumeration value="Less"/>
380       <xs:enumeration value="Equal"/>
381       <xs:enumeration value="GreaterEqual"/>
382       <xs:enumeration value="LessEqual"/>
383     </xs:restriction>
384   </xs:simpleType>
385   <!--
386     define quality parameter
387     - qualityParam - the quality parameter may be computed for the entire process (default
388       value), for a section or a service
389     - applyfunction - applicable only for the monitored quality parameters for a set of instances
390       (e.g. average, min, max ..)
391    -->
392   <xs:complexType name="qualityParamType">
393     <xs:simpleContent>
```

```
394       <xs:extension base="xs:string">
395         <xs:attribute name="applysection" type="xs:string" use="optional"/>
396         <xs:attribute name="applyservice" type="xs:string" use="optional"/>
397         <xs:attribute name="applyfunction" type="applyfunctionsetType" use="optional"/>
398       </xs:extension>
399     </xs:simpleContent>
400   </xs:complexType>
401   <!--
402     define attribute applyfunction for monitored instances
403     - there are three possibilities:
404       1. applyfunction="Average" the average quality values of the instances
405       2. applyfunction="MIN" the minimial monitored quality value of the instances
406       3. applyfunction="MAX" the maximal monitored quality value of the instances
407     If applyfunction attribute is missing, the default value is "Average".
408   -->
409   <xs:simpleType name="applyfunctionsetType">
410     <xs:restriction base="xs:string">
411       <xs:enumeration value="Average"/>
412       <xs:enumeration value="MIN"/>
413       <xs:enumeration value="MAX"/>
414     </xs:restriction>
415   </xs:simpleType>
416   <!--
417     type quality values to be considered in select-services and select-services-context
418     - there are two possibilities
419       1. "Prom" - consider the quality values promised by the provider
420       2. "Pred" - consider the quality values as predicted by CoFee
421   -->
422   <xs:simpleType name="qualityValuesSelectAttType">
423     <xs:restriction base="xs:string">
424       <xs:enumeration value="Prom"/>
425       <xs:enumeration value="Pred"/>
426     </xs:restriction>
427   </xs:simpleType>
428   <!--
429       define allowed types for constants and values
430   -->
431   <xs:simpleType name="Types">
432     <xs:restriction base="xs:string">
433       <xs:enumeration value="int"/>
434       <xs:enumeration value="float"/>
435       <xs:enumeration value="double"/>
436       <xs:enumeration value="long"/>
437       <xs:enumeration value="byte"/>
438       <xs:enumeration value="boolean"/>
439       <xs:enumeration value="string"/>
440     </xs:restriction>
441   </xs:simpleType>
442   <!--
443     logical linking of expressions with binary operator
444   -->
445   <xs:complexType name="BinaryExpressionLogicOperatorType">
446     <xs:sequence>
447       <xs:element name="expression" type="expressionType" minOccurs="2"
448                   maxOccurs="unbounded"/>
449     </xs:sequence>
450   </xs:complexType>
451   <!--
```

```
452     logical expression with unary operator
453   −−>
454   <xs:complexType name="UnaryExpressionLogicOperatorType">
455     <xs:sequence>
456       <xs:element name="expression" type="expressionType"/>
457     </xs:sequence>
458   </xs:complexType>
459   <!−−
460     logical linking of constraints with binary operator
461   −−>
462   <xs:complexType name="BinaryConstraintsLogicOperatorType">
463     <xs:sequence>
464       <xs:element name="constraints" type="constraintsType" minOccurs="2"
465         maxOccurs="unbounded"/>
466     </xs:sequence>
467   </xs:complexType>
468   <!−−
469     logical expression of constraints with unary operator
470   −−>
471   <xs:complexType name="UnaryConstraintsLogicOperatorType">
472     <xs:sequence>
473       <xs:element name="constraints" type="constraintsType"/>
474     </xs:sequence>
475   </xs:complexType>
476   <!−−
477     define actions
478   −−>
479   <xs:complexType name="actionType">
480     <xs:choice minOccurs="0" maxOccurs="unbounded">
481       <xs:element name="deploy" type="deployType"/>
482       <xs:element name="undeploy" type="undeployType"/>
483       <xs:element name="start" type="startType"/>
484       <xs:element name="stop" type="stopType"/>
485       <xs:element name="stop−instances" type="stopinstancesType"/>
486       <xs:element name="resume−instances" type="resumeinstancesType"/>
487       <xs:element name="cancel−instances" type="cancelinstancesType"/>
488       <xs:element name="update" type="updateType"/>
489       <xs:element name="replace−ws" type="replacewsType"/>
490       <xs:element name="select−services" type="selectservicesType"/>
491       <xs:element name="select−services−context" type="selectservicesContextType"/>
492       <xs:element name="throw−event" type="throweventType"/>
493       <xs:element name="throw−exception" type="throwexceptionType"/>
494       <xs:element name="setactive−ruleset" type="setactiverulesetType"/>
495       <xs:element name="report" type="reportType"/>
496       <xs:element name="report−rules" type="reportrulesType"/>
497       <xs:element name="report−error" type="reporterrorType"/>
498       <xs:element name="report−usercat" type="reportusercatType"/>
499       <xs:element name="report−feedbacks" type="reportfeedbacksType"/>
500     </xs:choice>
501     <xs:attribute name="id" type="xs:ID"/>
502     <xs:attribute name="select" type="xs:string"/>
503   </xs:complexType>
504   <!−−
505     define deploy action
506     − deploys a BPEL process from the BPR−repository to the BPEL engine
507       1. deploy the process with the id "processid"
508       2. deploy the process from "path"
509   −−>
```

```
<xs:complexType name="deployType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="type" type="deployOptionsType"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="deployOptionsType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="processid"/>
    <xs:enumeration value="path"/>
  </xs:restriction>
</xs:simpleType>
<!--
  define undeploy
  - undeploys the BPEL process with id "processid" from the BPEL engine
-->
<xs:complexType name="undeployType">
  <xs:attribute name="processid" type="xs:string"/>
</xs:complexType>
<!--
  define start
  - starts the process identified with the id "processid". The process is then ready to
    receive requests.
  - if the process id is missing, the process id is the one defined in the BPR-document
-->
<xs:complexType name="startType">
  <xs:attribute name="processid" type="xs:string" use="optional"/>
</xs:complexType>
<!--
  define stop
  - stops the process identified with the id "processid".
  - all the process instances are stopped.
  - if the process id is missing, the process id is the one defined in the BPR-document
-->
<xs:complexType name="stopType">
  <xs:attribute name="processid" type="xs:string" use="optional"/>
</xs:complexType>
<!--
  define stop-instances
  - stops a set of process instances that started within a given time interval.
-->
<xs:complexType name="stopinstancesType">
  <xs:sequence>
      <xs:element name="period" type="periodType"/>
  </xs:sequence>
  <xs:attribute name="processid" type="xs:string" use="optional"/>
</xs:complexType>
<!--
  define resume-instances
  - resumes a set of process instances that were previously stopped
-->
<xs:complexType name="resumeinstancesType">
  <xs:attribute name="processid" type="xs:string" use="optional"/>
</xs:complexType>
<!--
  define cancel-instances
  - cancels a set of process instances that are currently running.
```

```
568   −−>
569   <xs:complexType name="cancelinstancesType">
570     <xs:attribute name="processid" type="xs:string" use="optional"/>
571   </xs:complexType>
572   <!−−
573     define replace−ws
574     − replaces a web service with that from the specified url to its wsdl
575   −−>
576   <xs:complexType name="replacewsType">
577     <xs:sequence>
578       <xs:element name="service" type="serviceType" maxOccurs="unbounded"/>
579     </xs:sequence>
580   </xs:complexType>
581   <xs:complexType name="serviceType">
582     <xs:sequence>
583       <xs:element name="wsdl−url" type="xs:string"/>
584     </xs:sequence>
585     <xs:attribute name="name" type="xs:string"/>
586   </xs:complexType>
587   <!−−
588     define throw−event
589     − throw−event generates an event and informs the subscribers
590     − the business analyst may see the events while they are happening in the GUI
591       of the BPR−framework.
592     − The event may be of different types:
593         1. execute−rule − to inform that a rule is executed. It lists the id of the ruleset,
594            the id of the rule and the timestamp when it was triggered.
595         2. setactive −ruleset − to inform that a ruleset was activated.
596         3. info − to provide information
597   −−>
598   <xs:complexType name="throweventType">
599     <xs:simpleContent>
600       <xs:extension base="xs:string">
601         <xs:attribute name="type" type="EventAttType" use="optional"/>
602       </xs:extension>
603     </xs:simpleContent>
604   </xs:complexType>
605   <!−−
606     define valueEventAttType that specifies the type of an event
607   −−>
608   <xs:simpleType name="EventAttType">
609     <xs:restriction base="xs:string">
610       <xs:enumeration value="execute−rule"/>
611       <xs:enumeration value="setactive−ruleset"/>
612       <xs:enumeration value="info"/>
613       <xs:enumeration value="custom"/>
614     </xs:restriction>
615   </xs:simpleType>
616   <!−−
617     define throw−exception
618   −−>
619   <xs:complexType name="throwexceptionType">
620     <xs:sequence>
621       <xs:element name="value" type="valueExceptionType" maxOccurs="4"/>
622     </xs:sequence>
623     <xs:attribute name="type" type="exceptionType"/>
624   </xs:complexType>
625   <xs:simpleType name="exceptionType">
```

```
626     <xs:restriction base="xs:string">
627       <xs:enumeration value="custom"/>
628     </xs:restriction>
629   </xs:simpleType>
630   <xs:complexType name="valueExceptionType">
631     <xs:simpleContent>
632       <xs:extension base="xs:string">
633         <xs:attribute name="type" type="valueExceptionAttType"/>
634       </xs:extension>
635     </xs:simpleContent>
636   </xs:complexType>
637   <xs:simpleType name="valueExceptionAttType">
638     <xs:restriction base="xs:string">
639       <xs:enumeration value="message"/>
640     </xs:restriction>
641   </xs:simpleType>
642   <!--
643     define update
644     - updates the BPEL process description (or section) from the specified path
645   -->
646   <xs:complexType name="updateType">
647     <xs:attribute name="frompath" type="xs:string" use="optional"/>
648   </xs:complexType>
649   <!--
650     define select-services
651     - services are searched for the process or section or service that fulfill the
652       quality requirements
653     - attribute method - specifies which selection algorithm is applied
654       (ALG.OPTIM_S, ALG.OPTIM_PRO, ALG.OPTIM_HW)
655     - attribute methodclass - the implementing java class of the selection algorithm
656     - attribute qualityValues:
657         Prom - using an estimation for the process/section/atomic service considering the
658                quality values as promised by the service provider
659         Pred - using an estimation for the process/section/atomic service considering predicted
660                quality values
661   -->
662   <xs:complexType name="selectservicesType">
663     <xs:sequence>
664       <xs:element name="service-registries" type="serviceRegistriesType" minOccurs="0"/>
665       <xs:element name="quality-requirements" type="qualityRequirementsType"
666         maxOccurs="unbounded"/>
667     </xs:sequence>
668     <xs:attribute name="method" type="xs:string"/>
669     <xs:attribute name="methodClass" type="xs:string"/>
670     <xs:attribute name="qualityValues" type="qualityValuesSelectAttType"/>
671     <xs:attribute name="applysection" type="xs:string" use="optional"/>
672     <xs:attribute name="applyservice" type="xs:string" use="optional"/>
673   </xs:complexType>
674   <!--
675     define select-services-context
676     - extends the select-services action by specifying a list of user categories
677       to perform a context-aware service selection
678   -->
679   <xs:complexType name="selectservicesContextType">
680     <xs:complexContent>
681       <xs:extension base="selectservicesType">
682         <xs:sequence>
683           <xs:element name="list-categories" type="listcategoriesType"/>
```

```
684         </xs:sequence>
685       </xs:extension>
686     </xs:complexContent>
687   </xs:complexType>
688   <!--
689     define the service registries to be searched for services
690     - the service registry provides a wsdl interface accessible at the wsdl-url.
691     - the service registry interface is used to search services
692   -->
693   <xs:complexType name="serviceRegistriesType">
694     <xs:sequence>
695       <xs:element name="wsdl-url" type="xs:string" maxOccurs="unbounded"/>
696     </xs:sequence>
697     <xs:attribute name="id" type="xs:ID"/>
698     <xs:attribute name="select" type="xs:string"/>
699   </xs:complexType>
700   <!--
701     define the list of user categories
702     - it is used for the context-aware service selection in the
703       select -services-context action
704   -->
705   <xs:complexType name="listcategoriesType">
706     <xs:sequence>
707       <xs:element name="category" type="categoryType" maxOccurs="unbounded"/>
708     </xs:sequence>
709   </xs:complexType>
710   <xs:complexType name="categoryType">
711     <xs:sequence>
712       <xs:element name="context" type="contextType" maxOccurs="unbounded"/>
713     </xs:sequence>
714     <xs:attribute name="name" type="xs:string"/>
715     <xs:attribute name="id" type="xs:ID"/>
716     <xs:attribute name="select" type="xs:string"/>
717   </xs:complexType>
718   <!--
719     context parameters of the user categories
720   -->
721   <xs:complexType name="contextType">
722     <xs:simpleContent>
723       <xs:extension base="xs:string">
724         <xs:attribute name="param" type="attusercatparamVal"/>
725       </xs:extension>
726     </xs:simpleContent>
727   </xs:complexType>
728
729   <xs:simpleType name="attusercatparamVal">
730     <xs:restriction base="xs:string">
731       <xs:enumeration value="user-location"/>
732       <xs:enumeration value="user-age"/>
733       <xs:enumeration value="user-profession"/>
734     </xs:restriction>
735   </xs:simpleType>
736   <!--
737     quality requirements for the service selection
738   -->
739   <xs:complexType name="qualityRequirementsType">
740     <xs:sequence minOccurs="0">
741       <xs:element name="expression" type="expressionType" minOccurs="0"
```

```
                    maxOccurs="unbounded"/>
      <xs:element name="objective-function" type="objfunctionType" minOccurs="0"/>
      <xs:element name="fix" type="fixType" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="select" type="xs:string"/>
  </xs:complexType>
  <!--
    define fix
    - the services in the list will not be replaced during the service-selection
  -->
  <xs:complexType name="fixType">
    <xs:sequence minOccurs="0">
      <xs:element name="service" type="serviceType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <!--
    define function
    - a function may be used inside expressions
  -->
  <xs:complexType name="functionType">
    <xs:choice>
      <xs:element name="operation" type="operationType"/>
      <xs:element name="qualityParam" type="qualityParamType"/>
      <xs:element name="Value" type="xs:double"/>
    </xs:choice>
  </xs:complexType>
  <!--
    define objective function for service selection
  -->
  <xs:complexType name="objfunctionType">
    <xs:complexContent>
      <xs:extension base="functionType">
        <xs:attribute name="type" type="typeObjFunctionAttType"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!--
    define operation used inside functions
  -->
  <xs:complexType name="operationType">
    <xs:choice maxOccurs="2">
      <xs:element name="operation" type="operationType"/>
      <xs:element name="qualityParam" type="qualityParamType"/>
      <xs:element name="Value" type="xs:double"/>
    </xs:choice>
    <xs:attribute name="type" type="operationAttType"/>
  </xs:complexType>
  <!--
    define type of operations used for specifying the objective function or a function
      ADD - Plus
      SUB - Minus
      DIV - Divide
      MUL - Multiply
  -->
  <xs:simpleType name="operationAttType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ADD"/>
```

```
800       <xs:enumeration value="SUB"/>
801       <xs:enumeration value="DIV"/>
802       <xs:enumeration value="MUL"/>
803     </xs:restriction>
804   </xs:simpleType>
805   <!--
806     define if the objective function has to be minimized or maximized
807   -->
808   <xs:simpleType name="typeObjFunctionAttType">
809     <xs:restriction base="xs:string">
810       <xs:enumeration value="MIN"/>
811       <xs:enumeration value="MAX"/>
812     </xs:restriction>
813   </xs:simpleType>
814   <!--
815     define setactive-ruleset
816     - sets the active flag of a ruleset, i.e. the ruleset is activated or deactivated
817   -->
818   <xs:complexType name="setactiverulesetType">
819     <xs:simpleContent>
820       <xs:extension base="xs:boolean">
821         <xs:attribute name="id" type="xs:string"/>
822       </xs:extension>
823     </xs:simpleContent>
824   </xs:complexType>
825   <!--
826     define report action
827     - makes a report about all the monitored artifacts: the configurations
828       of the BPEL process, the measured QoS values, including exceptions
829       and events of a process during a given time period.
830   -->
831   <xs:complexType name="reportType">
832     <xs:sequence>
833       <xs:element name="period" type="periodType"/>
834     </xs:sequence>
835     <xs:attribute name="processid" type="xs:string" use="optional"/>
836   </xs:complexType>
837   <!--
838     define report-rules
839     - makes a report with the rules that were triggered for a process during
840       a given time period.
841   -->
842   <xs:complexType name="reportrulesType">
843     <xs:sequence>
844       <xs:element name="period" type="periodType"/>
845       <xs:element name="rulesetid" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
846     </xs:sequence>
847     <xs:attribute name="processid" type="xs:string" use="optional"/>
848   </xs:complexType>
849   <!--
850     define report-error
851     - makes a report with the exceptions and errors that were encountered
852       during process execution for a given time period.
853   -->
854   <xs:complexType name="reporterrorType">
855     <xs:sequence>
856       <xs:element name="period" type="periodType"/>
857     </xs:sequence>
```

```
858       <xs:attribute name="processid" type="xs:string" use="optional"/>
859     </xs:complexType>
860     <!--
861       define report-usercat
862       - makes a report that contains a list of the context-aware processes
863         that were created for a process. It contains the number of requests
864         delegated to each of the context-aware processes and the average
865         feedbacks.
866     -->
867     <xs:complexType name="reportusercatType">
868       <xs:sequence>
869         <xs:element name="period" type="periodType"/>
870       </xs:sequence>
871       <xs:attribute name="processid" type="xs:string" use="optional"/>
872     </xs:complexType>
873     <!--
874       define report-feedbacks
875       - makes a report with the feedbacks that were collected for the entire BPEL
876         process and for all contained Web services during a given time period.
877     -->
878     <xs:complexType name="reportfeedbacksType">
879       <xs:sequence>
880         <xs:element name="period" type="periodType"/>
881       </xs:sequence>
882       <xs:attribute name="processid" type="xs:string" use="optional"/>
883     </xs:complexType>
884   </xs:schema>
```

Listing A.1: BPRules XSD Schema

B Publications as (co-)author

[BCG13] Harun Baraki, Diana Comes, and Kurt Geihs
Context-aware Prediction of QoS and QoE Properties for Web Services
Conference on Networked Systems (NetSys), IEEE Xplore, pages 102-109, Suttgart, 2013.

[CBR12] Diana Comes, Harun Baraki, Roland Reichle, and Kurt Geihs
BPRules and the BPR-Framework: Comprehensive Support for Managing QoS in Web Service Compositions.
In *Proceedings of the 12th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS '12)*, Lecture Notes in Computer Science, Volume 7272, pages 222-235, Springer, Stockholm, 2012.

[CEG12] Diana Comes, Christoph Evers, Kurt Geihs, Axel Hoffmann, Romy Kniewel, Jan-Marco Leimeister, Stefan Niemczyk, Alexander Rossnagel, Ludger Schmidt, Thomas Schulz, Matthias Söllner, and Andreas Witsch
Designing Socio-Technical Applications for Ubiquitous Computing - Results from a multidisciplinary Case Study.
In *Proceedings of the 12th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS '12)*, Lecture Notes in Computer Science, Volume 7272, pages 194-201, Springer, Stockholm, 2012.

[CEG11] Diana Comes, Christoph Evers, Kurt Geihs, Daniel Saur, Andreas Witsch, and Michael Zapf
Adaptive Applications are Smart Applications.
International Workshop on Smart Mobile Applications, San Francisco, 2011.

[SSJ11] Thomas Schulz, Hendrik Skistims, Julia Zirfas, Diana Comes, Christoph Evers
Vorschläge zur rechtskonformen Gestaltung selbst-adaptiver Anwendungen.
In *Proceedings des Workshops Sozio-technisches Systemdesign im Zeitalter des Ubiquitous Computing (SUBICO)*, INFORMATIK, Berlin, 2011.

[GBC10] Kurt Geihs, Steffen Bleul, and Diana Comes
Automatische Dienstvermittlung in dienstorientierten Architekturen in Aktion.
Technical Report, University of Kassel, 2010.

[CBR10] Diana Comes, Harun Baraki, Roland Reichle, Michael Zapf, and Kurt Geihs
Heuristic Approaches for QoS-based Service Selection.
In *Proceedings of the 8th International Conference on Service Oriented Computing (ICSOC)*, Lecture Notes in Computer Science, Volume 6470, pages 441-455, Springer, San Francisco, 2010.

[CZG10] Diana Comes, Michael Zapf, and Kurt Geihs
QoS-based Self-Management for Business Processes.
In *Proceedings of the SAKS 2010 Workshop*, Electronic Communications of the EASST, Volume 27, ISSN 1863-2122, 2010.

[CBW09] Diana Comes, Steffen Bleul, Thomas Weise, and Kurt Geihs
A Flexible Approach for Business Processes Monitoring.
In *9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, Lecture Notes in Computer Science, Volume 5523, pages 116-128, Lissabon, 2009.

[BGC09] Steffen Bleul, Kurt Geihs, Diana Comes, and Marc Kirchhoff
Automated Integration of Web Services in BPEL4WS Processes.
16. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS) , Informatik aktuell, pages 105-116, Springer, Kassel, 2009.

[CBZ09] Diana Comes, Steffen Bleul, and Michael Zapf
Management of Business Processes with the BPRules Language in Service Oriented Computing.
In *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS)*, VDE, Kassel, 2009.

[BGC08] Steffen Bleul, Kurt Geihs, Diana Comes, and Marc Kirchhoff
Automated Management of Dynamic Web Service Integration.
In *15th Annual Workshop of HP Software University Association (HP-SUA)*, Infonomics-Consulting, Marrakech, Maroc, 2008 .

[WBC08] Thomas Weise, Steffen Bleul, Diana Comes, and Kurt Geihs
Different Approaches to Semantic Web Service Composition.
In *Third International Conference on Internet and Web Applications and Services (ICIW)*, pages 90-96, IEEE, Athens, 2008.

[BCG08] Steffen Bleul, Diana Comes, Marc Kirchhoff, and Michael Zapf
Self-Integration of Web Services in BPEL Processes.
In *Workshop on Self-Organising, Adaptive, Context-Sensitive Distributed Systems (SAKS)*, Wiesbaden, 2008.

Bibliography

[1] OASIS UDDI Specifications.
https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3 (accessed 2012-11-12).

[2] Apache ODE.
http://ode.apache.org/ (accessed 2014-08-07).

[3] Event-driven Process Chain (EPC).
http://www.ariscommunity.com/event-driven-process-chain (accessed 2013-06-21).

[4] Apache Axis2.
http://axis.apache.org/axis2/java/core/ (accessed 2014-08-07).

[5] BizTalk Server.
http://www.microsoft.com/en-us/biztalk/default.aspx (accessed 2013-06-19).

[6] Business Process Model and Notation.
http://www.bpmn.org/ (accessed 2013-06-29).

[7] IBM Business Process Manager.
http://www-03.ibm.com/software/products/us/en/business-process-manager-family/ (accessed 2013-06-19).

[8] Oracle BPEL Process Manager.
http://www.oracle.com/technetwork/middleware/bpel/overview/index.html (accessed 2013-06-04).

[9] SOAP Version 1.2: Messaging Framework (Second Edition).
http://www.w3.org/TR/soap12-part1/ (accessed 2012-11-12).

[10] Web Services Description Language (WSDL).
http://www.w3.org/TR/wsdl20/#component-ElementDeclaration (accessed 2012-05-29).

[11] Web Service Choreography Interface (WSCI) 1.0.
http://www.w3.org/TR/ws-cdl-10/ (accessed 2014-07-10), 2002.

[12] QoS for Web Services: Requirements and Possible Approaches.
http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/ (accessed 2012-06-26), 2003.

[13] Service-oriented Modeling and Architecture.
http://www.ibm.com/developerworks/library/ws-soa-design1/ (accessed 2012-08-24), 2004.

[14] Web Services Choreography Description Language Version 1.0.
http://www.w3.org/TR/ws-cdl-10/ (accessed 2014-07-10), 2005.

[15] OASIS: Reference Model for Service Oriented Architecture.
http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
(accessed 2012-08-11), 2006.

[16] Web Services Business Process Execution Language Version 2.0.
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
(accessed 2013-06-19), 2007.

[17] Web Service QoS Dataset of Zibin Zheng.
http://www.wsdream.net (accessed 2013-02-08), 2012.

[18] Automatic Service Brokering in Service Oriented Architectures.
http://www.vs.uni-kassel.de/ADDO/index.html (accessed 2012-11-20),
2013.

[19] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles.
Towards a Better Understanding of Context and Context-Awareness.
In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany*, pages 304–307. Springer-Verlag, 1999.

[20] 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).
http://www.ogf.org/documents/GFD.107.pdf
(accessed 2012-06-29), 2007.

[21] Daniel Austin, Abbie Barbir, Ed Peters, and Steve Ross-Talbo.
Web Services Choreography Requirements.
http://www.w3.org/TR/ws-chor-reqs/ (accessed 2013-05-16).

[22] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux.
A Declarative Approach for QoS-Aware Web Service Compositions.
In *Proceedings of the 5th International Conference on Service-Oriented Computing, Vienna, Austria*, ICSOC '07, pages 422–428. Springer, 2007.

[23] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux.
QoS Policies for Business Processes in Service Oriented Architectures.
In *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia*, pages 483–497, 2008.

[24] Harun Baraki, Diana Comes, and Kurt Geihs.
Context-Aware Prediction of QoS and QoE Properties for Web Services.
In *Conference on Networked Systems (NetSys), 2013*, pages 102–109. IEEE Xplore, Stuttgart, Germany, 2013.

[25] Luciano Baresi and Sam Guinea.
Towards Dynamic Monitoring of WS-BPEL Processes.
In *ICSOC 2005, Third International Conference of Service-Oriented Computing, Amsterdam, Netherlands*, pages 269–282. Springer, 2005.

[26] Luciano Baresi, Carlo Ghezzi, and Sam Guinea.
Smart Monitors for Composed Services.
In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 193–202, New York, NY, USA, 2004. ACM Press.

[27] Luciano Baresi, Sam Guinea, and Pierluigi Plebani.
WS-Policy for Service Monitoring.
In *6th VLDB International Workshop on Technologies for E-Services, Trondheim, Norway*, pages 72–83. Springer, 2005.

[28] Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea, and Paola Spoletini.
A Timed Extension of WSCoL.
In *IEEE International Conference on Web Services, ICWS 2007, Salt Lake City, UT, USA*, pages 663–670. IEEE Computer Society, 2007.

[29] Anjay Basal, M. Brian Blake, Srividya Kona, Steffen Bleul, Thomas Weise, and Michael C. Jaeger, editors.
WSC-08: Continuing the Web Services Challenge, IEEE Joint Conference (CEC/EEE 2008) on E-Commerce Technology (Tenth CEC'07) and Enterprise Computing, E-Commerce and E-Services (Fifth EEE'08), Washington D.C., USA, 2008. IEEE.

[30] Rainer Berbner, Tobias Grollius, Nicolas Repp, Oliver Heckmann, Erich Ortner, and Ralf Steinmetz.
An approach for the Management of Service-oriented Architecture (SOA) based Application Systems.
In *Workshop Enterprise Modelling and Information Systems Architectures (EMISA 2005), Klagenfurt, Austria*, pages 208–221, 2005.

[31] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz.
Heuristics for QoS-aware Web Service Composition.
In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 72–82, Washington, DC, USA, 2006. IEEE Computer Society.
ISBN 0-7695-2669-1.

[32] John Bird.
Higher Engineering Mathematics.
Sixth Edition, Elsevier, Newnes, 2010.

[33] Steffen Bleul, Michael Zapf, and Kurt Geihs.
Flexible Automatic Service Brokering for SOAs.
In *10 th IFIP / IEEE Symposium on Integrated Management (IM 2007)*, pages 410–419, Munich, Germany, 2007.

[34] Steffen Bleul, Diana Comes, Marc Kirchhoff, and Michael Zapf.
Self-Integration of Web Services in BPEL Processes.
Workshop on Self-Organising, Adaptive, Context-Sensitive Distributed Systems (SAKS), Wiesbaden, 2008.

[35] Steffen Bleul, Kurt Geihs, Diana Comes, and Marc Kirchhoff.
Automated Management of Dynamic Web Service Integration.

In *15th Annual Workshop of HP Software University Association (HP-SUA)*, Marrakech, Maroc, 2008. Infonomics-Consulting, Stuttgart, Germany.

[36] Steffen Bleul, Kurt Geihs, Diana Comes, and Marc Kirchhoff.
Automated Integration of Web Services in BPEL4WS Processes.
In *16. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, pages 105–116, Kassel, Germany, 2009. Springer.

[37] John S. Breese, David Heckerman, and Carl Kadie.
Empirical Analysis of Predictive Algorithm for Collaborative Filtering.
In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, WI, USA*, pages 43–52, 1998.

[38] Kjell Brunnström, Sergio Beker, Katrien De Moor, Ann Dooms, Sebastian Egger, Marie-Neige Garcia, and et al.
Qualinet White Paper on Definitions of Quality of Experience.
In *Dagstuhl seminar 12181*, 2012.

[39] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani.
An Approach for QoS-aware Service Composition based on Genetic Algorithms.
In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1069–1075, New York, NY, USA, 2005. ACM.

[40] Antonio Jorge Silva Cardoso.
Quality of Service and Semantic Composition of Workflows.
PHD thesis, University of Georgia, USA, 2002.

[41] Anis Charfi.
Aspect-Oriented Workflow Languages: AO4BPEL and Applications.
PHD thesis, Technical University of Darmstadt, Germany, 2007.

[42] Anis Charfi and Mira Mezini.
AO4BPEL: An Aspect-oriented Extension to BPEL.
World Wide Web, 10(3):309–344, September 2007.

[43] Xi Chen, Xudong Liu, Zicheng Huang, and Hailong Sun.
RegionKNN: A Scalable Hybrid Collaborative Filtering Algorithm for Personalized Web Service Recommendation.
In *The IEEE International Conference on Web Services - ICWS 2010*, pages 9–16. IEEE Computer Society, 2010.
ISBN 978-0-7695-4128-0.

[44] Tomasz Ciszkowski, Wojciech Mazurczyk, Zbigniew Kotulski, Tobias Hossfeld, Markus Fiedler, and Denis Collange.
Towards Quality of Experience-based Reputation Models for Future Web Service Provisioning.
In *Future Internet Architectures: New Trends in Service Architectures (2nd Euro-NF Workshop)*, Santander, Spain, 2009.

[45] Tomasz Ciszkowski, Wojciech Mazurczyk, Zbigniew Kotulski, Tobias Hossfeld, Markus Fiedler, and Denis Collange.
Towards Quality of Experience-based Reputation Models for Future Web Service Provisioning.

Telecommunication Systems, 51(4):283–295, 2012.
ISSN 1018-4864.

[46] Workflow Management Coalition.
Terminology and Glossary.
Document Number WFMC-TC-1011, 1999.

[47] Diana Comes, Steffen Bleul, Thomas Weise, and Kurt Geihs.
A Flexible Approach for Business Processes Monitoring.
In *9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Lecture Notes in Computer Science, Volume 5523*, pages 116–128, Lissabon, Portugal, 2009.

[48] Diana Comes, Steffen Bleul, and Michael Zapf.
Management of Business Processes with the BPRules Language in Service Oriented Computing.
In *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS) 2009, Kassel*, Kassel, Germany, 2009. University of Kassel, VDE.

[49] Diana Comes, Harun Baraki, Roland Reichle, Michael Zapf, and Kurt Geihs.
Heuristic Approaches for QoS-based Service Selection.
In *8th International Conference on Service Oriented Computing (ICSOC), Lecture Notes in Computer Science, Volume 6470*, pages 441–455, San Francisco, USA, 2010. Springer.

[50] Diana Elena Comes, Harun Baraki, Roland Reichle, and Kurt Geihs.
BPRules and the BPR-Framework: Comprehensive Support for Managing QoS in Web Service Compositions.
In *Distributed Applications and Interoperable Systems (DAIS), Stockholm, Sweden, Lecture Notes in Computer Science, Volume 7272*, pages 222–235. Springer, Heidelberg, 2012.
ISBN 978-3-642-30822-2.

[51] Marlon Dumas and Arthur H. M. ter Hofstede.
UML Activity Diagrams as a Workflow Specification Language.
In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 76–90, London, UK, 2001. Springer-Verlag.
ISBN 3-540-42667-1.

[52] Schahram Dustdar and Wolfgang Schreiner.
A Survey on Web Services Composition.
International Journal of Web and Grid Services, 1(1):1–30, August 2005.
ISSN 1741-1106.

[53] Abdelkarim Erradi, Vladimir Tosic, and Piyush Maheshwari.
MASC - .NET-Based Middleware for Adaptive Composite Web Services.
In *IEEE International Conference on Web Services, ICWS 2007, Salt Lake City, UT, USA*, pages 727–734. IEEE Computer Society, 2007.

[54] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia.

A Generic Quantitative Relationship between Quality of Experience and Quality of Service.
Network, IEEE, 24(2):36–41, 2010.

[55] Andy P. Field.
Analysis of Variance (ANOVA).
In Neil J. Salkind, editor, *Encyclopedia of Measurement and Statistics*, pages 33–36. SAGE Publications, Inc., 2007.

[56] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan.
Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments.
In BettyH.C. Cheng, Rogerio Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 146–163. Springer, Heidelberg, 2009.
ISBN 978-3-642-02160-2.

[57] Kurt Geihs, Steffen Bleul, and Diana Comes.
Automatische Dienstvermittlung in dienstorientierten Architekturen in Aktion (ADDOaction).
Technical report, University of Kassel, Distributed Systems Group, 2010.

[58] Kurt Geihs, Christoph Evers, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan.
Development Support for QoS-Aware Service-Adaptation in Ubiquitous Computing Applications.
In *SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computing, Taichung, Taiwan*, pages 197–202, 2011.

[59] Michael Hammer and James Champy.
A Manifesto for business revolution.
Harper Business, 1997.

[60] Mark Hansen.
SOA Using Java Web Services.
Prentince Hall, United States, 2007.

[61] Michael Jaeger, Gregor Rojec-Goldmann, and Gero Mühl.
QoS Aggregation for Web Service Composition using Workflow Patterns.
In *Eighth IEEE International Enterprise Distributed Object Computing Conference - EDOC 2004, Monterey, California, USA*, pages 149–159, sept. 2004.

[62] Nicolai Josuttis.
SOA in der Praxis.
dpunkt.verlag, Heidelberg, 2008.

[63] Matjaz B. Juric.
A Hands-on Introduction to BPEL, Part 2: Advanced BPEL.
`http://www.oracle.com/technetwork/articles/matjaz-bpel2-082861.html` (accessed 2013-06-17).

[64] Kenneth Karta.

An Investigation on Personalized Collaborative Filtering for Web Service Selection.
Honours Programme thesis, University of Western Australia, Brisbane, 2005.

[65] Markus Keidl and Alfons Kemper.
A Framework for Context-Aware Adaptable Web Services.
In *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 826–829. Springer, 2004.

[66] Alexander Keller and Heiko Ludwig.
The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services.
Journal of Network and Systems Management, 11(1):57–81, 2003.

[67] Ulrich Küster, Birgitta König-Ries, Michael Klein, and Mirco Stern.
DIANE - A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding, and Invocation on the Web.
International Journal of Electronic Commerce (IJEC), 12 - Special Issue: Semantic Matchmaking and Resource Retrieval on the Web(2):41–68, 2007.

[68] Ulrich Küster, Birgitta König-Ries, and Matthias Klusch.
Evaluating Semantic Web Service Technologies: Criteria, Approaches and Challenges.
In *Progressive Concepts for Semantic Web Evolution: Application and Developments*, pages 1–24. IGI Global, 2010.

[69] 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 *Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10*, pages 365–380, 2010.

[70] Qinghua Lu and Vladimir Tosic.
Support for Concurrent Adaptation of Multiple Web Service Compositions to Maximize Business Metrics.
In Nazim Agoulmine, Claudio Bartolini, Tom Pfeifer, and Declan O'Sullivan, editors, *Integrated Network Management*, pages 241–248. IEEE, 2011.

[71] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck.
Web Service Level Agreement (WSLA) Language Specification.
`http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf` (accessed 2012-06-26), 2003.

[72] Anbazhagan Mani and Arun Nagarajan.
Understanding Quality of Service for Web Services.
`http://www.ibm.com/developerworks/webservices/library/ws-quality/index.html` (accessed 2013-11-12).

[73] Michael Maximilien and Munindar P. Singh.
A Framework and Ontology for Dynamic Web Services Selection.
IEEE Internet Computing, 8:84–93, 2004.

[74] Ingo Melzer.
Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis.
Spektrum, Heidelberg, 2010.

[75] Daniel A. Menasce.
Composing Web Services: A QoS View.
IEEE Internet Computing, 8(6):88–90, November 2004.

[76] Jaeger Michael, Gero Mühl, and Sebastian Golze.
In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*.

[77] Douglas C. Montgomery and George C. Runger.
Applied Statistics and Probability for Engineers .
Fifth Edition, John Wiley, 2011.

[78] Elisabetta Nitto, Massimiliano 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, Vienna, Austria*, pages 295–306, Berlin, Heidelberg, 2007. Springer-Verlag.
ISBN 978-3-540-74973-8.

[79] Michael Papazoglou.
Web Services: Principles and Technology.
Pearson Prentince Hall, England, 2008.

[80] Michael Papazoglou.
Web Services and SOA, Principles and Technology.
Pearson, England, 2012.

[81] Jose A. Parejo, Pablo Fernandez, and Antonio R. Cortes.
QoS-Aware Services composition using Tabu Search and Hybrid Genetic Algorithms.
In *Actas de los Talleres de las Jornadas de Ingenieria del Software y Bases de Datos, Vol. 2, No. 1*, pages 55–66, 2008.

[82] Chris Peltz.
Web Services Orchestration and Choreography.
Computer, 36(10):46–52, October 2003.
ISSN 0018-9162.

[83] Shuping Ran.
A Model for Web Services Discovery with QoS.
SIGecom Exch., 4(1):1–10, March 2003.
ISSN 1551-9031.

[84] Nicolas Repp.
Überwachung und Steuerung dienstbasierter Architekturen - Verteilungsstrategien und deren Umsetzung.
PHD thesis, Technical University of Darmstadt, Germany, 2002.

[85] Nicolas Repp, Julian Eckert, Stefan Schulte, Michael Niemann, Rainer Berbner, and Ralf Steinmetz.
Towards Automated Monitoring and Alignment of Service-based Workflows.
In *IEEE International Conference on Digital Ecosystems and Technologies 2008 (IEEE DEST 2008), Phitsanulok, Thailand*, pages 235 – 240, 2008.

[86] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl.
GroupLens: An Open Architecture for Collaborative Filtering of Netnews.
In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, Chapel Hill*, pages 175–186, 1994.

[87] Michael Rosen, Boris Lublinsky, Kevin T. Smith, and Marc J. Balcer.
Applied SOA- Service Oriented Architecture and Design Strategies.
Wiley, England, 2008.

[88] Sheldon M. Ross.
Statistik für Ingenieure und Naturwissenschaftler.
3. Auflage, Spektrum, Muenchen, 2006.

[89] Rune Saetre, Mohammad Ullah Khan, and Peter Herrmann.
End-user Composition of Web-based Services: The Plus Alpha Approach.
In *India-Norway Workshop on Web Concepts and Technologies, Trondheim, Norway*, pages 51 –61, 2011.

[90] Rich Seeley.
Business analyst role key in SOA software development projects.
`http://searchsoa.techtarget.com/news/1325388/Business-analyst-role-key-in-SOA-software-development-projects`
(accessed 2013-06-17), 2008.

[91] Junaid Shaikh, Markus Fiedler, and Denis Collange.
Quality of Experience from user and network perspectives.
Annales des Télécommunications, 65(1-2):47–57, 2010.

[92] Lingshuang Shao, Jing Zhang, Yong Wei, Junfeng Zhao, Bing Xie, and Hong Mei.
Personalized QoS Prediction for Web Services via Collaborative Filtering.
In *IEEE International Conference on Web Services - ICWS 2007, Salt Lake City, UT, USA*, pages 439–446. IEEE, 2007.

[93] Mazen M. Shiaa, Jens E. Vaskinn, and Richard T. Sanders.
Service Composition for End-users: A Tool for Telephony Services.
In *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference, Berlin, Germany*, pages 1 –2, 2010.

[94] Rafal Stankiewicz, Piotr Cholda, and Andrzej Jajszczyk.
QoX: What is it really?
IEEE Communications Magazine, 49(4):148–158, 2011.

[95] Mingdong Tang, Yechun Jiang, Jianxun Liu, and Xiaoqing (Frank) Liu.
Location-Aware Collaborative Filtering for QoS-Based Service Recommendation.

In Carole A. Goble, Peter P. Chen, and Jia Zhang, editors, *19th International Conference on Web Services, Honolulu, Hawaii, USA*, pages 202–209. IEEE, 2012.
ISBN 978-1-4673-2131-0.

[96] Vladimir Tosic.
Autonomic Business-Driven Dynamic Adaptation of Service-Oriented Systems and the WSPolicy4MASC Support for Such Adaptation.
International Journal of Systems and Service-Oriented Engineering, 1(1):79–95, 2010.

[97] Vladimir Tosic, Kruti Patel, and Bernard Pagurek.
WSOL - Web Service Offerings Language.
In *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, CAiSE '02/ WES '02, pages 57–67, London, UK, 2002. Springer-Verlag.
ISBN 3-540-00198-0.

[98] Vladimir Tosic, Abdelkarim Erradi, and Piyush Maheshwari.
WS-Policy4MASC - A WS-Policy Extension Used in the MASC Middleware.
In *IEEE International Conference on Service Computing (SCC), Salt Lake City, UT, USA*, pages 458–465. IEEE Computer Society, 2007.

[99] Vladimir Tosic, Basem Suleiman, and Hanan Lutfiyya.
UML Profiles for WS-Policy4MASC as Support for Business Value Driven Engineering and Management of Web Services and their Compositions.
In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007) - Annapolis, Maryland, USA*, pages 157–168. IEEE Computer Society, 2007.

[100] Hong-Linh Truong and Schahram Dustdar.
A Survey on Context-aware Web Service Systems.
`http://www.infosys.tuwien.ac.at/staff/sd/papers/surveycontextws-submittedversion.pdf` (accessed 2014-07-20), 2009.

[101] Tobias Unger, Frank Leymann, Stephanie Mauchart, and Thorsten Scheibler.
Aggregation of Service Level Agreements in the Context of Business Processes.
In *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, Munich, Germany*, pages 43–52. IEEE Computer Society, 2008.

[102] Wil M.P. van der Aalst and Arthur H. M. Ter Hofstede.
YAWL: Yet Another Workflow Language.
Information Systems, 30:245–275, 2003.

[103] Lo Wei, Yin Jianwei, Deng ShuiGuang, Li Ying, and Wu Zhaohui.
Collaborative Web Service QoS Prediction with Location-Based Regularization.
In *19th International Conference on Web Services, Honolulu, Hawaii, USA*, pages 464–471. IEEE, 2012.

[104] Thomas Weise, Steffen Bleul, Diana Comes, and Kurt Geihs.
Different Approaches to Semantic Web Service Composition.

Third International Conference on Internet and Web Applications and Services (ICIW), IEEE, Athens, Greece, pages 90–96, 2008.

[105] Thomas Weise, Steffen Bleul, Marc Kirchhoff, and Kurt Geihs, editors.
Semantic Web Service Composition for Service-Oriented Architectures, 2008 IEEE Joint Conference on E-Commerce Technology (CEC'08) and Enterprise Computing, E-Commerce and E-Services (EEE'08), Washington D.C., USA, July 2008. IEEE.

[106] Mathias Weske.
Business Process Management: Concepts, Languages, Architectures.
Springer, Germany, 2007.

[107] *Web Ontology Language (OWL)*, 2009. World Wide Web Consortium (W3C).
URL `http://www.w3.org/TR/owl2-overview/`.

[108] Qi Xie, Kaigui Wu, Jie Xu, Pan He, and Min Chen.
Personalized Context-Aware QoS Prediction for Web Services Based on Collaborative Filtering.
In Longbing Cao, Jiang Zhong, and Yong Feng, editors, *Advanced Data Mining and Applications (2)*, volume 6441 of *Lecture Notes in Computer Science*, pages 368–375. Springer, 2010.
ISBN 978-3-642-17312-7.

[109] Tao Yu and Kwei-Jay Lin.
A Broker-Based Framework for QoS-Aware Web Service Composition.
In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE '05, Hong Kong, China*, pages 22–29. IEEE Computer Society, 2005.
ISBN 0-7695-2073-1.

[110] 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, May 2004.

[111] Yilei Zhang, Zibin Zheng, and Michael R. Lyu.
Exploring Latent Features for Memory-Based QoS Prediction in Cloud Computing.
In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, October 4-7, 2011*, pages 1–10. IEEE, 2011.

[112] Zibin Zheng, Yilei Zhang, and Michael R. Lyu.
Distributed QoS Evaluation for Real-World Web Services.
In *IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA, July 5-10, 2010*, pages 83–90. IEEE Computer Society, 2010.

[113] Zibin Zheng, Hao Ma, Michael R. Lyu, and Irwin King.
QoS-Aware Web Service Recommendation by Collaborative Filtering.
IEEE Transactions on Services Computing, 4:140–152, April 2011.

[114] Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser.
Perspectives on Web Services- Applying SOAP, WSDL and UDDI to Real-World Projects.

Springer, Heidelberg, 2003.

[115] Farhana Zulkernine, Patrick Martin, Chris Craddock, and Kirk Wilson.
A Policy-based Middleware for Web Services SLA Negotiation.
In *7th IEEE International Conference on Web Services - ICWS 2009, Los Angeles, CA, USA*, pages 1043 –1050, 2009.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlich-materiellen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich hierfür nicht die Hilfe eines Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Kassel, im September 2014

M.Sc. Diana- Elena Reichle