THESIS / THÈSE

DOCTOR OF SCIENCES

Methods for Improving QoS-driven Management of Web Services and their Services and their Service Level Agreements

Herssens, Caroline

Award date:
2010

Awarding institution:
University of Namur

Methods for Improving QoS-driven Management of Web Services and their Service Level Agreements

LOUVAIN
School of Management

Caroline Herssens
Louvain School of Management
University of Namur

Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences de Gestion

March 2010

Examination committee
Advisor : Pr. Stéphane Faulkner
Examiner : Pr. Manuel Kolp
Examiner : Pr. Marco Saerens
Reader : Pr. Yves Pigneur
Reader : Pr. Esteban Zimanyi

Acknowledgements

I would like to acknowledge the Professor Stéphane Faulkner, my advisor, for the numerous discussions about this work, his patience and the trust he gave me. I especially notice the pragmatism and the efficiency of his supervision. I thank Stéphane for his continuous support to the fulfillment of this thesis.

I thank the Professor Marco Saerens for his precious collaboration and advices to the achievement of my thesis. Most of all, I thank Marco for his kindness and his enthusiasm.

I thank the Professors Manuel Kolp, Yves Pigneur and Esteban Zimanyi for accepting to participate to the jury of this dissertation.

I thank the Professor Philippe Chevalier for the supervision of the jury and for his punctual help.

I also would like to thank the Doctor Ivan J. Jureta for his involvement in this work and the time he spent reading and commenting it. I also thank him for his relevant suggestions, his effectiveness and his many ideas.

I thank past and present members of the Louvain School of Management, among others: Adrien, Anne-Cécile, Anne-Laure, Bertrand, Catherine, Didier, Donatien, Florence, François F., François M., Géraldine, Jean-Christophe, Jérôme, Kenneth, Kevin, Luh, Silvia, Valérie and Youssef. I thank them for the numerous talks about research and other topics.

I thank Valérie Klein for her time and her proofreading.

I finally thank my family and my friends for their support.

Abstract

The Internet is today subject to growing needs of information exchanges and modular applications. The Web now proposes information through different media (texts, pictures, videos, sounds) available on different websites (social networks, blogs, hosting sites) reached from a large set of devices (PCs, PDAs, smartphones). The data transfered is related to an infinite amount of topics as stock exchange information, picture editing or flight booking. To cope with such possibilities, one of the most fitted solution is offered by the use of Web Services. Web Services are self-describing, open components that support rapid, low-cost composition of distributed applications over the Web.

Web Services Management tackles issues as installation, configuration, collecting metric and tuning to ensure responsive service execution. Improving the management of Web Services is essential to increase the adoption of Web Services and to impose the adoption of standards. There are different possibilities to improve Web Services management but managing services through quality offers large opportunities. The emergence of Web Services involves an increasing number of available services on the Internet. A service requester is then faced to a large choice while it executes a service. Indeed, a particular functionality can be offered by numerous Web Services emanating from multiple providers. To discriminate among available services, the requester has to compare the quality levels offered by providers. Quality provides several opportunities to respond to most management issues (i.e.: service monitoring, performance measurement, capacity planning) and, more specifically, allows to differentiate functionally equivalent services.

This thesis presents a set of contributions related to the management of Web Services through quality information. The first significant contribution we made is a quality model that enables stakeholders of the service execution to specify their capabilities and expectations about quality. This model is essential to allow a quality driven management of Web services. The quality model is used by stakeholders to express involved quality criteria, measurement functions, precisions about favored qualities and quality information that are necessary to manage Web services. The second important contribution proposed in this work is a set of methods improving the management of Web services with help of quality information specified by stakeholders of the service execution. Three different methods are proposed: (1) A selection method which defines the best service according to non-functional requirements of the requester while multiple services are available to perform the same task at different quality levels. (2) A composition method which defines the best available composition of services to perform a complex task involving several services. (3) The last method proposed is a model enabling to determine the profile of a requester based on quality evaluations given to past services execution. The profiling method allows to automatically recommend to the requester services that correspond to his quality expectations. The last essential contribution of this thesis concerns the management of Service Level Agreements that allows to define quality contract between services requesters and providers. We propose two models enabling to improve the SLA control and monitoring. (1) The first model relies on normative autonomous agents managing services activities. In our proposal, normative agents are able to control the execution of a service while observing its quality conformance. (2) The second proposed model enables to monitor a service execution through context information. Quality elements are related to different context categories and context modifications are handled in order to ensure the quality conformance to the initial contract.

Contents

List of Figures

List of Tables

Glossary

$\mathtt{AC}$	Quality levels required by $\mathtt{C}$
$\mathtt{AC}_l$	Quality levels required by client l
$\mathtt{AC}_{lj}$	Quality level required by the client l for the quality property j
$\mathtt{AC}_{lj}^{prior_{rules}}$	Rules of priorities between quality levels required by client l for quality property j
$\mathtt{AC}_{lj}^{prior_{strength}}$	Strength of priorities between quality levels required by client l for quality property j
$\mathtt{AC}_l^{mod}$	Modalities of quality levels required by client l
$\mathtt{AC}_{lj}^{mod}$	Modalities of quality levels required by client l for property j
$\mathtt{AC}_l^{opt}$	Optimal values of quality levels required by client l
$\mathtt{AC}_{lj}^{opt}$	Optimal values of quality levels required by client l for quality property j
$\mathtt{AC}_l^{prior}$	Priorities between quality levels required by client l
$\mathtt{AC}_{lj}^{prior}$	Priorities between quality levels required by client l for quality property j
a_l^q	Sensitivity of the client l to the brand image of providers
a_l^x	Sensitivity of the client l to the deception induced by of the gap between the expected and the delivered quality level
$\mathtt{AP}$	Advertised quality levels of $\mathtt{P}$
$\mathtt{AP}_{ki}$	Advertised quality levels of provider k for transaction i
$\mathtt{AP}_{ki_\alpha j}$	Advertised quality levels of quality property j announced by provider k of the service α fulfilling the task i
$\mathtt{AP}_{kij}$	Advertised quality levels of quality property j of provider k for transaction i
b_l	Bias of the client l
$\mathtt{C}$	Client of a transaction
c_s	Cost of the service s
D	The dependency submodel
DAH	Directed Acyclic Hypergraph
$\mathtt{f}$	Feedback given by $\mathtt{C}$ to $\mathtt{P}$
$\bar{\mathtt{f}}$	True feedback given by $\mathtt{C}$ to $\mathtt{P}$
F_l	Feedback profile of the client l
i	transaction
k	provider
l	client
MAS	Multi-Agent System
$\mathtt{M}$	Mediator agent between $\mathtt{P}$s and $\mathtt{C}$
n_c	Total number of clients
n_{kl}	Total number of transactions occurring between provider k and client l
N	Total number of transactions
NoMAS	Norm oriented Multi-Agent System
n_p	Total number of providers
$\mathtt{OP}$	Observed quality levels of $\mathtt{P}$
$\mathtt{OP}_{ki}$	Observed quality levels of provider k for transaction i
$\mathtt{OP}_{ki_\alpha j}$	Actually observed quality levels of quality property j offered by provider k of the service α fulfilling the task i
$\mathtt{OP}_{kij}$	Observed quality levels of quality property j of provider k for transaction i
P	The priority submodel
$\mathtt{P}$	Provider of a transaction
$\bar{\mathbf{q}}$	Quality characteristic
Q	The quality submodel
q	Quality dimension
q_k	Brand image of the provider k
QoS	Quality of Service
QVDP	The Quality-Value-Dependency-Priority model
r_s	Rating of a service s
$\mathcal{R}$	Reputation score of $\mathtt{P}$
$\mathcal{R}_\mathtt{P}$	Reputation score of $\mathtt{P}$
RL	Reinforcement Learning
s	Service
s_{i_j}	Service j able to fulfill the task i
σ_l	Stability of the client l in providing constant ratings
SLA	Service Level Agreement
SLO	Service Level Objective
SOS	Service-oriented System
t_i	Task of the composition process to be fulfilled by a service
$t_i^{f^j}$	Functional constraint j of the task i
t^f	Set of functional requirements of a task
$t_i^{nf^j}$	Non-functional constraint j of the task i
t^{nf}	Set of non-functional requirements of a task
V	The value submodel
WS	Web service
x_{kli}	True feedback of transaction i occurring between provider k and client l
y_{kli}	Feedback of transaction i occurring between provider k and client l
z_{kli}	Truncated feedback of transaction i occurring between provider k and client l

Chapter 1

Introduction

The Internet is today subject to growing needs of information exchanges and modular applications. The Web now proposes information through different media (texts, pictures, videos, sounds) available on different websites (social networks, blogs, hosting sites) reached from a large set of devices (PCs, PDAs, smartphones). The data transfered is related to an infinite amount of topics as stock exchange information, picture editing or flight booking. To cope with such possibilities, one of the most fitted solution is offered by the use of **service-oriented computing**.

Papazoglou and Georgakopoulos [162] define services as self-describing, open components that support rapid, low-cost composition of distributed applications. Services are offered by service providers organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. The service implementation and deployment is achieved through the **Web Service** Standard. The Web Service standard is a recent paradigm of emerging Web components. It combines a set of technologies, protocols, and languages to allow automatic communication between Web applications through the Internet. A Web Service is any application that exposes its functionalities through an interface description and makes it available for use by other programs on the Web. More precisely, the World Wide Web Consortium (W3C) [218] defines a Web Service as a software system designed to support interoperable machine-to-machine interaction over a network. Web Services rely on multiple existing technologies. These are described with the Web Service Description Language (WDSL) which defines Web Services as endpoints operating on messages containing either document-oriented or procedure-oriented information [217]. Usually, Web Services use the HyperText Transfer Protocol (HTTP) [216] as a fundamental communication protocol, carrying communication messages between Web Services and their clients. To communicate data, Web Services use the extensible markup language (XML) [220]. SOAP is a lightweight protocol designed for exchanging structured information in a decentralized, distributed environment [219]. Finally, the Universal Description, Discovery and Integration (UDDI) standard defines a universal method for enterprises to dynamically discover and invoke Web Services [155].

In addition to technologies supporting Web Services, several specifications about Web Services allow support activities (e.g.: WS-Policy [12], WSFL [122], WS-Security [149], WS-Management [48], etc.). The aim of such specifications is to improve the monitoring, the security, the orchestration and other activities related to the execution of Web Services. Among such activities, the Web Service Management tackles issues as installation, configuration, collecting metric and tuning to ensure responsive service execution. It involves gathering information about the managed service platform, services and business processes and managed-resource status and performance via root-cause failure analysis, SLA monitoring and reporting, service deployment, life-cycle management and capacity planning [163]. Improving the management of Web Services is essential to increase the adoption of the service-oriented computing and to impose the adoption of standards.

There are different possibilities to improve Web Services management but managing services through **quality** offers large opportunities. The emergence of Web Services involves an increasing number of available services on the Internet. A service requester is then faced to a large choice while it executes

a service. Indeed, a particular functionality can be offered by numerous Web Services emanating from multiple providers. The requester's choice is usually driven by his utility function which means that he chooses the service that maximizes its utility. To discriminate among available services, the requester has to compare the quality levels offered by providers. The quality level of a service is usually denominated as its Quality of Service (QoS) which is a combination of several characteristics of a service, such as availability, security, response time or throughput [143]. E.g.: two services providing the same functionality can be differentiated by their response time with one service fulfilling the functionality in 200 milliseconds and the other in 450 milliseconds. QoS can then play a major role in the Web Service management. The first research question addressed in this thesis is: '*How to improve the management of services through the use of quality information?*'

Quality provides several opportunities to respond to most management issues (i.e.: service monitoring, performance measurement, capacity planning) and, more specifically, allows to differentiate functionally equivalent services. Quality information can be easily expressed by the requester in order to state its requirements about quality and by the service provider to advertise its capabilities about the quality level offered. Quality information is critical to improve the management of Web Services in accordance with stakeholders expectations. However, quality information must be properly defined, collected and used to improve the efficiency of management activities. In this thesis, we propose to improve the management of Web Services through a quality driven framework tackling Web Services management activities.

Beyond quality-driven management of Web Services, another issue is essential to monitor Web Services executions. In order to ensure the right execution of a Web Service, it is essential to define a contract between stakeholders and to control its execution. This contract guarantees that the provider capabilities meet the requester expectations about functionalities and quality to deliver. The agreement about the service functionality and the quality level to deliver is stated between the requester and the provider with a **Service Level Agreement** (SLA). An SLA is a contract between the said parties which specifies the Quality of Service (QoS) levels that should be met [109]. However, an SLA definition does not guarantee the accurate execution of the service, i.e.: some agreements can be unmet (e.g.: the execution time observed is longer than expected) or the whole service execution can fail. To prevent and avoid such unexpected failures, the management of SLAs is essential. Although an SLA usually defines the functionality to deliver, its mains specifications refer to the level of quality criteria involved in the contract. SLAs then need an appropriate definition of quality and a continual observation of quality levels provided. The second research question addressed in this thesis is: '*How to improve the management of Service Level Agreements through the use of quality information?*'

SLAs mainly define quality to be achieved during services executions. Hence, it is then essential to monitor quality levels and to track unexpected changes in quality delivered. The framework we propose in this thesis allows to improve the SLA management through the use of quality information. The SLA management checks the Web Service behavior and proposes corrective actions to be taken while the service misses its quality objectives.

Scope of the thesis

This thesis presents contributions related to the management of Web services and service level agreements. This work addresses actual issues of Web services management and proposes different models and methods to solve these issues. Three main areas are approached in this thesis. The first area is the definition of a quality model allowing to express requirements about expected quality levels of a service transaction. The second area is the quality-driven management of Web services. We propose different methods relying on the quality model to improve and automate the management of Web services through quality information. The third area is the management of service level agreements. The SLA management is complementary to the QoS management of Web services. It allows to formally define the quality agreements of service users and providers resulting from quality-driven management methods. However, this thesis does not address all issues related to the management of Web services. Some research area as policy management, security or workflow engineering are not argued. We focus our efforts on quality-driven management of Web services. The three research areas considered make a coherent whole responding to some of the main

current issues of quality-driven Web services management.

Epsitemological field

The epistemological perspective of this thesis is the structuralism. Service-oriented Systems are subject to numerous recognized standards, languages and models. These standards improve the utilization of services but restrain their management possibilities. Models and languages act as power structures and narrow the research scope that must conform to them. The contributions proposed in this thesis are the resulting effects of social interactions occurring between involved actors of services executions. Indeed, this work aims at improving the management of Web services from the user side through a quality approach. The quality model and the proposed management methods have been designed according to user requirements about services utilization. Our work is driven by social interactions between services users and services providers. This research is within an existing conceptual framework and proposes new contributions resulting from social interactions between involved actors (i.e.: developers, requesters and providers).

1.1 Contributions

This thesis proposes contributions related to the issues of QoS management of Web Services and their SLA contracts management. The management of Web services and their SLAs is essential to ensure the requester satisfaction in spite of the increasing quantity and complexity of services available on the Web. The first significant contribution we made is a quality model that enables stakeholders of the service execution to specify their capabilities and expectations about QoS. This model is essential to allow a quality driven management of Web services. The QoS model is used by stakeholders to express involved quality criteria, measurement functions, precisions about favored qualities and quality information that are necessary to manage Web services. Quality specifications made by services requesters and providers can step in several service monitoring models and methods. The second important contribution proposed in this work is a set of methods improving the management of Web services with help of quality information specified by stakeholders of the service execution. Three different methods are proposed: (1) A selection method which defines the best service according to non-functional requirements of the requester while multiple services are available to perform the same task at different quality levels. (2) A composition method which defines the best available composition of services to perform a complex task involving several services. This composition method relies on quality information to propose the composition that will best satisfy the requester's expectations. (3) The last method proposed is a model enabling to determine the profile of a requester based on quality evaluations given to past services execution. The profiling method allows to automatically recommend to the requester services that correspond to his quality expectations. The last essential contribution of this thesis concerns the management of Service Level Agreements that allows to define quality contract between services requesters and providers. We propose two models enabling to improve the SLA control and monitoring. (1) The first model relies on normative autonomous agents managing services activities. In our proposal, normative agents are able to control the execution of a service while observing its quality conformance. (2) The second proposed model enables to monitor a service execution through context information. Quality elements are related to different context categories and context modifications are handled in order to ensure the quality conformance to the initial contract.

This thesis is organized through three main parts. The different parts provide advances allowing a QoS-driven management of SLAs. The first part is dedicated to the conceptual foundations of service quality and management quality. This part also outlines the quality model. The second part proposes management methods to improve the Web Services management through a quality dimension. The third part presents advances in management of SLAs established between the stakeholders of the service execution. The specific contributions related to each part are detailed here.

1.1.1 Quality of Service Definition and Specification

This first part defines main concepts involved in the QoS management. Part I outlines service quality, it proposes a QoS definition and explains its importance in service management. Next, it presents a brief state of the art of Web Service management. Once the foundations of QoS have been laid down, Part I introduces our quality model. The quality model will provide the specification of a conceptualization. The quality model proposed, the Quality-Value-Dependency-Priority (QVDP) model, can be used by any part concerned by the specification of QoS: (1) The service requester who wishes to specify its expectations regarding the service quality level to achieve during the service execution. (2) The service provider in order to advertise the quality level that provided services can reach. (3) Any stakeholder involved in the service management in order to use information about expectations or capabilities or to monitor quality level values.

The QVDP model provides quality constructs enabling to specify advanced concepts such as the priorities between quality characteristics, the preferences over values of quality characteristics and the dependencies. In comparison to existing QoS models, the QVDP model has several advantages: (1) It does not predefine characteristics. Indeed, the QVDP model can be used to specify usual QoS characteristics (e.g.: the latency, the availability or the reliability) but also to define context-specific characteristic (e.g.: reactivity of stock exchange information or resolution of photo editing). (2) The measurement of quality characteristics is left to involved actors. The requester and the provider can specify their definition of metrics and their aggregation functions. (3) It provides advanced concepts such as priorities, preferences and dependencies. Such constructs allow to give a rich description of how services perform. The specifications of these concepts are particularly relevant to manage trade-offs at runtime. These constructs clarify the characteristics and their respective values which must be optimized (priorities and preferences). These also provide an explicit representation of how qualities interact (dependencies), i.e., how some variation of a quality affects the degree of satisfaction of others.

We compare the QVDP model to several existing models. We compare it especially to the UML QoS metamodel [157]. We also extend the UML QoS metamodel it to allow the specification of advanced concepts introduced in QVDP.

1.1.2 QoS-driven management of Web Services

The QoS model proposed in Part I allows large possibilities of service management. Once requester expectations and providers capabilities have been defined with the QoS model, the resulting specifications offer a sound basis to process monitoring and management operations. Such operations involve QoS measurement, specification matching, service selection or contract definition. Part II outlines three QoS-driven management operations: (1) a selection method; (2) a composition method, and; (3) a profiling method, enabling to define the behavior and the reactivity of a service user.

The first method proposed in Part II is a selection method. This method uses QoS specification provided by the service requester and the service providers in order to fix the most suited service among available ones. Each provider advertises its performance capabilities with the QoS model introduced in Part I. To select the most appropriate service, the method applies trade-offs upon basis of priorities and preferences information specified by the service user. The selection method proceeds along the following steps: (1) The selector rejects services that do not fulfill user expectations about values of QoS properties. (2) The selector organizes QoS properties into positive and negative hierarchies that make explicit the contribution to more generic quality properties (e.g., responsiveness). (3) The selector links weights to QoS properties that reflect their relative importance. (4) Pairwise comparisons of values of quality properties are made by the service selector. (5) The result of pairwise comparisons is combined to the weights of criteria to determine the score of positive and negative hierarchies. Cost/benefit analysis is then performed on each service to establish their ranking and determine the most suitable service to the user request.

The second method proposed in Part II is a composition method. A composition of service is used to fulfill complex tasks involving multiple elementary services (e.g.: a travel booking involves flight booking and accommodation booking). In composition problems, the pool of available services fulfilling

the different tasks may vary. As the selection issue, the composition issue can be driven by quality and many QoS criteria can be used to describe the services. The best composition is the one that best fits to the requester's quality expectations. The composition method presented in this thesis is QoS-driven and relies on QoS specifications about requester priorities and preferences and about provider advertised and observed quality levels. These specifications have been introduced in the QoS model presented in Part I. With these specifications, the composition method uses a reinforcement learning algorithm to define optimal service compositions. In order to solve the composition issue, we propose a two-step service composition method. (1) We apply a multi-criteria method to aggregate QoS characteristics to obtain a single rating for each service. (2) These aggregated ratings are input to a reinforcement learning algorithm. This algorithm finds the composition that maximizes the QoS delivered to the requester according to its expectations. The RL approach is capable of handling variations in the pool of available services by exploring compositions other than those that historical data shows appropriate, i.e., the algorithm handles the exploration-exploitation tradeoff. Interestingly enough, the introduced RL algorithm guarantees (asymptotically) optimal exploitation for a given exploration level. Our first experiments reported here illustrate that, for a given level of exploration, our algorithm is more efficient than comparable approaches. The contributions of this chapter are the QoS aggregation and the RL algorithm, so that no commitments are made on, e.g., how composition proceeds once a selection is identified, how interoperability is ensured, and so on. This ensures that our results are generic.

The third method proposed in Part II is a user profiling method. A profiling method allows to define the usual behavior of a consumer. The behavior of a service user is useful to anticipate its expectations and recommend services to execute according to its profile. As other issues presented in Part II, the user profiling method relies on information specified by the QoS model. To build a user profile, we refer to its specification about preferences and priorities and observe its reactivity regarding expected and observed performances of the service. The priorities and preferences specifications have been elicited in the QoS model presented in Part I. The method proposed computes the feedback profile of users after service transactions. After each transaction, the method compares the true (unbiased) feedback that should have been given with the feedback actually given by the service user. The unbiased feedback is a function of the user preferences, the user priorities, the quality level advertised by the provider and the quality level observed during the transaction. The service quality is then computed with a function of the magnitude and the direction of the gap between expected service and perceived service. The actual feedback is then a unbiased quantification of the gap between user's expectations (priorities and preferences, expected quality level) and observed values of quality properties after the transaction. The feedback profile is then the source of the difference observed between the actual feedback and the feedback given by the user after the transaction. On the basis of such differences, the method presented computes the feedback profiles of users. A feedback profile characterizes a user on three dimensions, called outlook, sensitivity to deception and, sensitivity to brand image. Outlook describes a user's degree of optimism or pessimism in feedbacks. Outlook reflects the fact that some users consistently give more favorable or defavorable feedback. We then have two kinds of sensitivity. The sensitivity to deception characterizes a user's degree of deception when delivered quality varies from expected quality. Deception occurs while the quality level delivered is lower than the quality level expected by the user. Users can be slightly or strongly sensitive to deception and can be averse if it exists an aversion effect to the deception. Deception sensitive users will react negatively to deception (i.e., they will decrease their feedbacks) while deception averse ones will positively react (i.e., they will increase their feedbacks) when the observed level of quality is lower than the quality level advertised by the provider. In other words, for the same deception related to the same gap between quality expected and delivered, users will react differently. The sensitivity to brand image characterizes the effect of provider's brand image on the user. Users will react differently to provider's image, their feedbacks can be sensitive to the image of the provider but also adverse if the user has an aversion to image of providers. Users that are brand image sensitive will improve their feedbacks for providers with a notorious brand image while users averse to brand image will decrease their feedbacks for such users.

1.1.3 SLA management

The third part of this thesis presents models improving the management of Service Level Agreements. SLA contracts usually concern the obligations of the parts involved in the transaction. The obligations mainly define the quality level to be achieved by the provider during the transaction. The SLA definition is then connected to the user expectations and the provider capabilities specification made with the QoS model introduced in Part I. Part III introduces two models enabling a QoS-driven management of SLA.

The first approach proposed in Part III present an architecture managing SLAs based on normative agent. This architecture monitors the SLA achievement through: (1) A language enabling the communication between stakeholders involved in the SLA (i.e.: the service provider and the service requester). (2) The definition of a SLA that meets the provider capabilities and the client requirements. (3) The service execution monitoring to check the conformance between quality level expected and actually observed. (4) The insurance of the SLA achievement with mutual obligations involving both stakeholders of the contract. (5) A third part controller which monitors and evaluates the SLA execution and penalizes the agent that does not fulfill its obligations.

The second approach proposed in Part III enables an autonomic adaptation of SLA responding to occurring context modifications. To this aim, context elements are classified into five distinct context categories: user, provider, resource environment and the Web Service itself. The proposed approach also relies on dependencies existing between elements of context enabling to propagate context modifications. These dependencies have been previously introduced in the QoS model presented in Part I. The SLA approach is composed of an architecture leaning on an SLA manager to drive the autonomic adaptation based on context elements. The adaptation process uses context elements and dependencies to enable an autonomic adjustment of existing SLAs to ensure the service conformance to user expectations. The adaptation process involves the following steps : (1) Context modifications are reported to the SLA manager that identifies changes and starts the adaptation process. (2) Observed context variations are propagated through context dependencies existing over different elements of context by the SLA manager. (3) Once context variations have been propagated to all context categories, the SLA manager checks the compatibility between user expectations and provider capabilities. (4) Upon base of the result of the compatibility checking, the SLA manager keeps the existing SLA, set up a new SLA between the user and the same provider or select another service better fitting user expectations.

1.2 Organization

Most chapters in this thesis have been published as peer-reviewed publications or have directly served in the elaboration of other publications. In this respect, this thesis provides neither a central treatment of related work nor a discussion of limitations. Each of these considerations is locally dealt with in the relevant chapters, and for the given topics.

Part I introduces the conceptual foundations used throughout this thesis. This part defines the quality of service and lays the foundations of quality driven management of Web Services. Chapter 2 introduces the context of the quality management of Web Services and concisely outlines existing Web Service quality management approaches. Chapter 3 proposes a quality model allowing several Web Service management processes. This quality model allows requesters to specify quality expectations, providers to advertise service qualities and management third parties to compare alternative Web Services through multiple quality criteria.

Part II of this thesis presents methods improving the QoS-driven management of Web Services. Chapter 4 proposes a service selection method based on quality properties of a service. The selection method relies on the quality model introduced in Chapter 3 to apply multi-criteria decision making methods (MCDM). MCDM methods are applied on quality criteria elicited with the quality model in order to define the available service which best suits user expectations. Chapter 5 proposes a service composition method based on the QoS users' expectations and providers' capabilities. The selection method relies on specifications provided with the quality model introduced in Chapter 3 and a reinforcement learning method to determine the best possible combination of available services. Chapter 6 proposes a user

profiling method relying on the formulation of QoS users' expectations. The profiling method improves the management of Web Services by enabling accurate recommendations of services according to their observed consumer behavior.

Part III presents models improving the management of Service Level Agreements. Chapter 7 proposes a normative management of service level agreements defined between Web Services. The normative management improves the achievement of contract defined between stakeholders of the service execution. Chapter 8 proposes to use context to manage SLA during service executions. The context management of SLA allows to prevent and react to unexpected failures occurring during Web Services executions.

Part IV concludes this thesis. Finally, a summary is given and principal directions for future work are identified and discussed.

Part I

Quality of Service Definition and Specification

Outline of Part I

The first part of this thesis focuses on the introduction of concepts addressed in this thesis and the definition of conceptual foundations required to manage web services from quality information. The main contribution of this part is the definition of a quality model used by requesters and providers to specify theirs capabilities and expectations about QoS. A such model is essential to allow the management of web service through non-functional properties.

Chapter 2 introduces the context of the quality management of web services. It first gives a comprehensive definition of the web service quality. It then introduces the state-of-the-art about quality monitoring in service-oriented computing.

Chapter 3 proposes a quality model to allow requesters specify quality expectations, providers advertise service qualities and management third parties compare alternative services. Upon basis of observed similarities between various existing quality model, we review these and integrate then into a single quality model called QVDP. We also highlight the need for integration of priority and dependency information within any quality model for services an propose precise submodels for doing so. Our intention is for the proposed model to serve as a reference point for further developments in quality models for service-oriented systems. To this aim, we extend the part of the UML metamodel specialized for Quality of Service with QDVP concepts unavailable in UML. Chapter 3 has been published in Software Quality Journal [103].

Chapter 2

Quality Management of Web Services : Context

This Chapter briefly introduces the context of this research. The Chapter 2 defines the Quality of Service concept and presents the foundations of QoS monitoring with its state-of-the-art.

2.1 Web Service Quality Definition

Before executing a web service, numerous factors are involved in the choice of a service provider. Factors can be: the user's expectations, the provider's capabilities, the inherent characteristics of the service and, the perceived value of the service [151]. Among such factors, quality is a critical issue. When competing providers deliver the same functionalities, usually, the service the most requested is the one that deliver the better quality attributes at the lowest cost. Garvin has analyzed in [60] how quality can be perceived in different domains as economics, philosophy or marketing. Among existing views (transcendent, product-based, user-based, manufacturing-based and value based) of quality, two are particularly relevant to the web services quality issue: the *product view* and the *user view*.

- The product view Garvin's hypothesis is that if a product is manufactured with good internal properties, it will have good external properties. The quality level of a product indicates the presence or absence of measurable product properties. The product view of quality can be assessed in an objective manner [151].
- The user view is about the extent to which a product meets user needs and expectations. It is then useful to identify what product attributes users consider to be important [151].

According to these two definitions of quality level, we choose to refer to the user-view of quality. The user-view definition of quality is well suited to web services because:

- The same functionality can be provided by several providers, at different quality levels. A web service can also be offered by the same provider at different quality levels. The pool of available web service for a same functionality is then significant;
- There are also numerous potential users of web services, with different expectations. An user can also modify its expectations and change its consumer profile;
- A product-view of quality will result to a limited pool of web service with few available web services for a given functionality.

The user-view is then the quality view the most suited to web services. It enables a large pool of available services in which users can find the one that best fits their expectations.

The aim of this thesis is to provide management tools enabling the service user to execute services according to its expectations. To this aim ,we first provide a framework that allows the user to express

its needs (Part 1). This framework supports the quality user-view of web service. We then use this framework to determine services that best fit user's needs in web services issues such as selection or composition (Part 2). These steps are also related to the quality user-view of a service. We finally provide Service Level Agreements management tools, enabling to control contracts established between providers and users (Part 3). Agreements are managed according to users needs and then also refer to a user-view of quality.

The quality characteristics of a service need to be quantified to compute its efficiency and to compare its performance with other similar services. Measurement of characteristics allows to have a quantitative perception of the quality. The measurement usually establishes baselines for qualities and allows to quantify the improvement of modifications [151]. From the users view, the measurement of quality is characterized by quality factors such as reliability, availability or usability. Such factors are aggregated to define the overall Quality of Service (QoS). QoS reflects the quality of a web service, both in terms of correctness of functional behaviour and level of supported QoS. Many definitions and existing methods provide an overview of quality criteria [62; 87; 88; 89; 113; 136]. However, some definitions better fits to the web service quality measurement. The main quality criteria used to define QoS of a web service have been defined by Mani and Nagarajan [131]:

- Availability: Availability is the quality aspect of whether the Web service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Also associated with availability is time-to-repair (TTR). TTR represents the time it takes to repair a service that has failed. Ideally smaller values of TTR are desirable.

- Accessibility: Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. High accessibility of Web services can be achieved by building highly scalable systems. Scalability refers to the ability to consistently serve the requests despite variations in the volume of requests.

- Integrity: Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction. A transaction refers to a sequence of activities to be treated as a single unit of work. All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back.

- Performance: Performance is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. Throughput represents the number of Web service requests served at a given time period. Latency is the round-trip time between sending a request and receiving the response.

- Reliability: Reliability is the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.

- Regulatory: Regulatory is the quality aspect of the Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement. Web services use a lot of standards such as SOAP, UDDI, and WSDL. Strict adherence to correct versions of standards (for example, SOAP version 1.2) by service providers is necessary for proper invocation of Web services by service requestors.

- Security: Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control. Security has added importance because Web service invocation occurs over the public Internet. The service provider can have different approaches and levels of providing security depending on the service requestor.

In addition to these criteria, some criteria may be added according to the user expectations or the execution context of the web service. E.g.: the responsiveness is a critical criteria for web service providing stock exchange information.

2.2 Quality of Service Monitoring

The section presented here is inspired by Benharref et al. [14]. Once quality characteristics of a service are defined, the quality must be appropriately managed. Quality criteria must be measured and monitored. Management issues of Web service are usually divided into two dimensions: (1) management of functional aspects, namely fault management, and (2) management of non functional aspects such as Quality of Service. Usually, management of web services is highly platform-dependent, implying some limitations: (1) management features are usually available to web services providers but often not to other partners (clients, third parties); (2) management solutions are usually restricted to only one management aspect, functional or non-functional; and (3) most of management solutions require considerable amount of computer and network resources to be deployed and used. Among existing approaches of web service management, the World Wide Web Consortium (W3C)[1] presents a set of requirements that Web service (WS) management architectures should satisfy to provide management features [21]. This includes the definition of standard metrics, management operations, and methodologies for accessing management capabilities. The complying architectures must provide a manageable, accountable, and organized environment for Web Services operations. It must support at least resource accounting, usage auditing and tracking, performance monitoring, availability, configuration, control, security auditing and administration, and service level agreements. Another approach in which the WS provides specific interfaces for management is presented by Farell and Kregel [51] in which the developer is supposed to supply commands and APIs for management operations that are invoked by the management system. Casati et al. [34] classify the management of WS into three levels: infrastructure-level, application-level, and business-level. The infrastructure-level deals with the Web service platform while the application-level focuses on the WS themselves. The business-level takes into consideration the conversations between a WS and its client.

Such management approaches assume that the WS provide operations that one can invoke. Developers have to develop and deploy these operations in addition to the core business operations the WS is offering.

Some management tools to be integrated into WS environment are already available. Hewlett Packard's WS management engine [76] is a collection of software components that enables some management features, including the definition and enforcement of SLA. Parasoft [164] provides a set of tools (SOAP test, .TEST, WebKing) to assist during the lifecycle of a WS. These tools have to be installed and configured, thus requiring extra resources and introducing new cost for WS providers.

Work on WS management can be divided into two main groups: works targeting functional aspects of WS and works tackling non-functional aspects. The first group is about the correctness of interactions between WS and their clients while the second group is concerned with QoS management of WS. The second group is concerned with the QoS management of WS.

2.2.1 Functional management

The majority of work on functional management is based on active testing where appropriate test cases have to be carefully generated, executed, and their result analyzed. This unavoidable phase of active testing has, however, practical limitations. First of all, exhaustive testing is impractical for quite large

[1] http://www.w3c.org/

WS. In fact, test cases can not cover all possible execution scenarios that a WS will have to handle while serving client's requests. The size of test cases is bounded by the cost a Web services provider is willing to spend on testing activities. Usually, active testing stops whenever developers are confident that the Web service is good enough to be put into the market. Many recent results were published lately describing test cases generation methods for Web services; they are mainly based on static analysis of WSDL documents. Xiaoying et al. [7] present a method for test data generation and test operation generation based on three types of dependencies: input, output, and input/output. ChangSup et al. [111] combined both EFSM models and WSDL documents to generate test cases.

2.2.2 Non-functional management

The management of QoS includes definition of QoS attributes, QoS publication, discovery, validation, and monitoring. Existing approaches for QoS management can be classified into two groups: one based on extending related technologies including WSDL and UDDI to support QoS and the other mandating independent entities to perform some or all of QoS management tasks. In the first category, W3C (2003) extends SOAP header to include QoS information. WSDL is also extended to describe QoS parameters, their associated values, computation units (e.g., millisecond, request/second), and so forth. UDDIe, a UDDI extension, consists of extending the current UDDI data structure with QoS information [192]. The aim of these extensions is to allow QoS-based publication and discovery of Web services. In the second group, solutions are presented for one or more of the following QoS management operations:

- QoS attributes: The first step in QoS management is the definition of evaluations criteria and attributes. A set of attributes have been defined, studied, and used in software engineering for a long time [53; 65; 182].
- QoS publication and discovery [105; 173; 189]: This operation allows providers to include QoS information in WSDL. This information is then used by requestors when selecting the appropriate Web service in terms of functional and QoS requirements.
- QoS verification [105; 189; 208]): This operation allows the provider to certify that the QoS claimed by the Web Service is accurate.
- QoS negotiation [189]: If the available published QoS requirements do not satisfy a clients needs, negotiation operations and strategies can be followed to reach an agreement on different QoS attributes.
- QoS monitoring [13; 94; 184; 189]: Performs monitoring of Web services during interactions with clients to assess if the QoS attributes agreed upon in previous points are delivered.

Chapter 3

Quality of Service Specification Model

This Chapter has been published in Software Quality Journal [103]. The Chapter 3 presents the first significant contribution of this thesis, our quality model. The quality model is essential to allow a quality driven management of web services. The quality model is used by requesters and providers to provide quality specifications considered in services management. The expression of requirements and capabilities enables to drive activities as the services selection and composition from stakeholders sides. The definition of an appropriate quality model is essential to enable the efficiency of a quality-driven management of web services. The quality model proposed here will be used to specify quality information required to apply the methods proposed in Part II.

3.1 Introduction

Quality has been variously defined as value, conformance to specifications, conformance to requirements, fitness for use, loss avoidance, or achieving and/or exceeding customer expectations (see, e.g., [175] for a business-oriented overview). Reasoned, structured, and systematic action taken to achieve desired quality—i.e., *quality management*—has been an active area of enquiry in various fields, most notably business (e.g., [46; 52; 61; 90; 97]). In relation to software engineering, the International Organization for Standardization [85] and the IEEE [82] define software quality as the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs. Ensuring the quality of software has become a major issue in research and practice since the 1970s [18]. As increasingly complex software plays a critical role in business and other areas of life, it is clear that comprehensive and precise methods and tools are needed to create and run software that is, among other, safe, dependable, and efficient [158]. Service-oriented systems (e.g., [138; 162]) intended for enabling the Semantic Web [16; 190] are no exception.

Service-oriented systems are one relevant current response to the increasing complexity of computing systems and the variability of their operating environments (e.g., [54; 110; 202]). A *service* is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network [138; 162]. Services are offered by *service providers*, i.e., organizations that ensure service implementations, supply service descriptions, and provide related technical and business support. A *service-oriented system* (SOS) incorporates *service composers*. A service composer receives *service requests* from human users or other systems, then discovers, selects, and coordinates the execution of services so as to fulfill a given service request. SOS fit well the Semantic Web, i.e., a next-generation of the World Wide Web on which services' properties, capabilities, interfaces, effects, and qualities, and data exchanged between services are described in an unambiguous, machine-understandable form. Within efforts towards the realization of the Semantic Web, ontologies prove a particularly relevant means for enabling the sharing, understanding, and automated processing of data

about, and exchanged between, heterogeneous services available on the Web. It is now well established that the aims of automated service discovery, access, composition, and management cannot be achieved without expressive ontologies [11; 79; 140; 198].

An ontology is a specification of a conceptualization [68]. Among the various ontologies relevant for a SoS—including those for, e.g., interfaces, capabilities, and behaviors of services—an ontology to describe quality attributes (such as, e.g., security, safety, performance) and reason thereon is necessary. A quality (or Quality of Service, i.e., QoS) ontology is used (i) in service requests to specify expectations on quality levels to achieve when fulfilling the request; (ii) by service providers to advertise quality levels that their services can achieve; and (iii) by service composers to select among alternative services, those that are to take part in fulfilling the service request. Limited expressiveness of a quality ontology (a) unnecessarily restricts the requester when defining expectations on service delivery; (b) does not allow a service provider to give a rich description of how its services perform; and (c) limit the set of criteria over which a service composer compares alternative services when performing service composition.

Various SOS quality ontologies proposed in the literature integrate concepts and constructs intended for the representation of similar aspects of a system's quality. For instance, all admit the need for metrics, so that all include constructs for the definition of a metric's identifier, unit, observed and desired values, and so on. Such apparent similarities (reviewed in §3.4) led us to conclude that an effort is needed for their comparison and integration within a comprehensive framework. Knowing that work on SOS quality ontologies will further evolve, such a framework is intended to (i) integrate the previous results within a comprehensive and consistent quality model from which the various prior ontologies can be derived; (ii) assist in future work on SOS quality conceptualization by facilitating the positioning of new modeling primitives w.r.t. available ones; and (iii) to make it clear how various proposed ontologies overlap, and thus, what constructs are agreed upon as necessary when modeling quality. The framework is therefore not an ontology *per se*, but a model which integrates various relevant concepts and constructs for conceptualizing quality in SOS, and which is instantiated when defining ontologies.

The proposed model has two salient characteristics. First, it does not itself define particular qualities (such as, e.g., availability, reliability, safety, security, adaptability, maintainability)—instead, it lets the modeler decide what sets of measures that can be collected over the given SOS are aggregated (and how they are aggregated) to define such qualities. In this respect, the model is not an attempt at settling the debates on what a universal definition of, e.g., usability would be. Second, it integrates two submodels that are novel w.r.t. available quality ontologies—namely, a submodel for specifying priority between qualities, and a submodel for specifying the dependencies between qualities (i.e., how one quality varies when another one varies). Both submodels are particularly important for managing tradeoffs at runtime: without the former, it is unclear what quality to optimize when the optimization of some quality harms the degree of satisfaction of another; without the latter, we do not have an explicit representation of how qualities interact (i.e., we do not know how some variation of a quality affects the degree of satisfaction of others).

The proposed quality model has been validated in two realistic application settings. We have used part of the quality model elsewhere [100; 101] to specify the quality information in service requests, to characterize the quality of individual services, and to define quality criteria used when comparing alternative services within a reinforcement learning algorithm for automated service composition. Herein we present a case study performed in cooperation with the European Space Agency (ESA). The case study illustrates how the proposed model has been used to represent quality information about a service that ESA makes available to researchers. We describe the case study in detail below (§3.2). The quality model is then presented and exemplified through the case study (§3.3). Source quality models used to define the proposed model are then reviewed (§3.4). Among the related efforts we discuss, we place particular attention on the part of the Unified Modeling Language specialized for QoS [157]. We compare it in detail with our quality model, extend the UML QoS metamodel with novel primitives, and show how the extended model is used within the case study (§3.5). The chapter closes with a discussion of the proposed model (§3.6), along with conclusions (§3.7).

3.2 Motivation and Case Study

3.2.1 Motivation

We have highlighted earlier that expressive quality models are necessary to engineer high-quality systems. Our principal motive for the present work stems from our observation (discussed in detail later on §3.4) that currently available quality models cannot be used to describe considerations clearly relevant when representing and reasoning about quality. Namely, they are of limited use when preferences and priorities are to be written down in a quality model. In other words, available models have difficulties expressing evidently relevant quality information. For example, we need to express in a quality model that it is preferred for the value of a quality metric to be in one given range than in another, whereby satisfying this preference is less important than maintaining some values for overall response time. In the said requirement, both preferences and priorities intervene. Moreover, we may consider this requirement conditionally on the satisfaction of another requirement—we therefore require not only a quality model which allows preference and priority orders to be expressed, but also conditional preferences and priorities. Such considerations are beyond the expressivity of most available quality models, as we show later on (§3.4). We cannot hope to engineer high quality systems if we cannot accommodate quality-related requirements that come naturally to engineers and future users.

3.2.2 Case study

ESA program on Earth observation allows researchers to access and use infrastructure operated and data collected by the agency. The infrastructure comprises a high performance computing cluster, significant storage capacity, and services accessible through the Web.[1] The platform, called "ESA Grid Processing on-Demand" can be used to access data collected by the Envisat ESA satellite, deployed to measure the atmosphere, ocean, land, and ice over a five year period. It is also possible for researchers to execute their own algorithms on the data.[2]

The case study focuses on the information provided by the MERIS instrument on the Envisat ESA satellite. MERIS is a programmable, medium-spectral resolution, imaging spectrometer operating in the solar reflective spectral range. MERIS is used in observing ocean color and biology, vegetation and atmosphere and in particular clouds and precipitation. In relation to MERIS, web services are made available by the ESA for access to the data the instrument sends and access and use of the associated computing resources.

We are interested in the remainder in the service called "MERIS MVGI Regional", which provides vegetation indexes for a given region of the globe. A vegetation index measures the amount of vegetation on the Earth's surface. MERIS is a particularly relevant for the acquisition of such data for it increases precision over comparable instruments. The service which accesses the MERIS data generates regional maps for specific time periods. A visual example of what the service provides to the user is given in Figure 3.2. Below, we briefly review the functional requirements that the service satisfies. We then consider quality requirements.

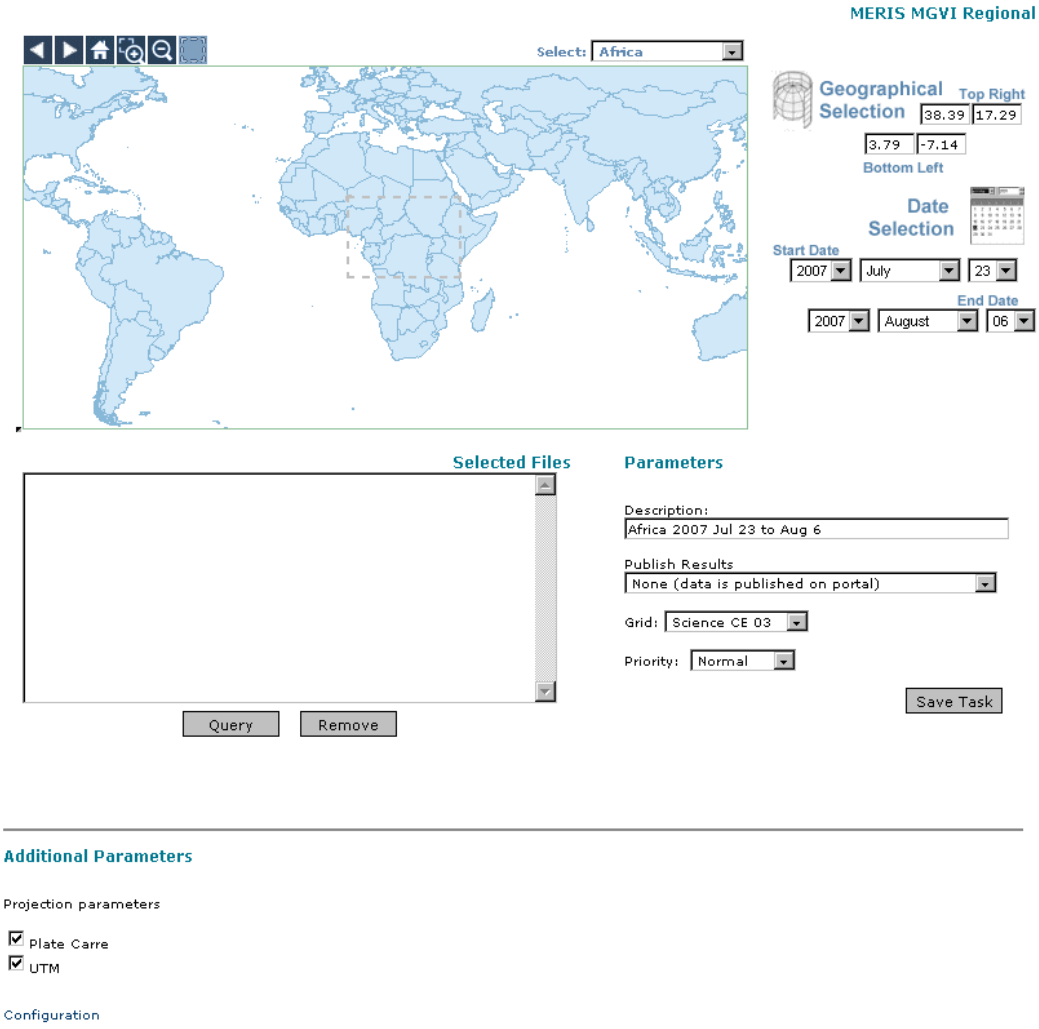
Functional requirements. The service processes MERIS data and extracts the vegetation index. The processing can be selected for any time range (with the start of the satellite mission as the earliest time point); an option is available to delimit the region of the world of interest. The graphical user interface used to access the service is shown in Figure 3.1. Figure 3.2 illustrates the visualization of the output obtained for the Senegal region. The following are the required inputs of the service: *Time range*, *Bouding box* (to select a region of the globe), *Dataset*, *Publish site*, and *Projection type*.

Quality requirements. Due to the calculations executed by the service and its parallel use, expected delays and availability are relevant quality considerations from the user's perspective. To make this information available to the users, quality considerations need to be expressed and measured during the execution of the service. We focus on three such considerations, namely availability, reliability, and

[1] http://gpod.eo.esa.int

[2] http://eopi.esa.int/esa/esa?type=file&ts=1186489893497&table=aotarget&cmd=image&id=1260

Figure 3.1: Graphical user interface of the ENVISAT/MERIS MGVI web service



latency. For now we define them as follows. We then return to each throughout the chapter and illustrate how they are defined using the model we propose.

- Availability indicates the duration when a component is available for queries. Its value in percent is obtained as follows [173]:

$$Availability = \frac{upTime}{upTime + downTime}$$

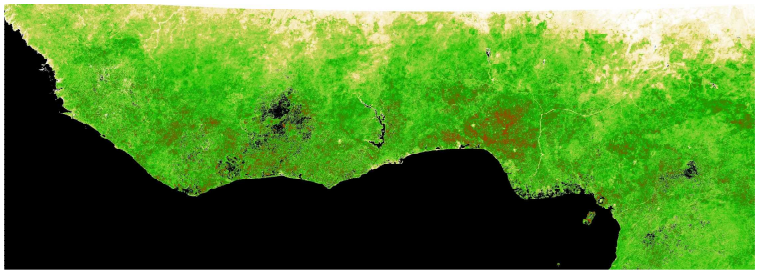
  Historical data on the service's operation indicate that the values are higher than 94%.

- Reliability is a measure of confidence that the service is free from errors. Its value is given in percent and calculated as follows:

$$Reliability = \frac{succeededAttempts}{succeededAttempts + failedAttempts}$$

  Relliability historically remained above 82%.

Figure 3.2: An illustration of the result provided by the ENVISAT/MERIS MGVI Web Service



- Latency measures the mean time taken by the platform to return the expected result. The value is given in minutes.

$$Latency = \frac{\sum_1^n networkTime + selection/compositionTime + executionTime}{n}$$

where n is the total number of past executions. Latency should not exceed 6 hours by day of the selected period but must be superior to 4 hours by day of the selected period. (This range is certified for a service requestor having network bandwidth of a least 15 mbits/s.)

This basic specification is incomplete. Further explanations are needed. For example, how different values are obtained is not described in the above informal specification. A quality model will provide a checklist of relevant information and in this respect assist the requestor and the provider in evaluating and managing the quality of the service.

3.3 Quality Model for Service-Oriented Systems

We introduce the quality model which integrates all components of the ontologies compared and reviewed in the subsequent part of the chapter (§3.4). We call it the *Quality-Value-Dependency-Priority (QVDP)* model. The instantiation of its modeling primitives is illustrated using simple examples mainly for clarity of presentation.

Definition 1. The *Quality-Value-Dependency-Priority (QVDP)* model $\mathbf{QVDP} \equiv \langle \mathbf{Q}, \mathbf{V}, \mathbf{D}, \mathbf{P} \rangle$ consists of: the Quality Characteristics submodel $\mathbf{Q}$, the Quality Value submodel $\mathbf{V}$, the Quality Dependency submodel $\mathbf{D}$, and the Quality Priority submodel $\mathbf{P}$.

Overall, the QVDP intergates submodels for different purposes related by shared modeling primitives. $\mathbf{Q}$ integrates the concepts of *quality dimension* $\mathbf{q}$ which is instantiated to represent measurable properties of a given SOS and *quality characteristic* $\bar{\mathbf{q}}$. A quality dimension can also be an aggregate of other quality dimensions obtained by applying some *aggregation function*. By grouping measurable $\mathbf{q}$ into $\bar{\mathbf{q}}$, the modeler can define how more abstract quality aspects of a system (e.g., safety, security, usability, maintainability, etc.) are conceptualized in a given SOS. A $\mathbf{q}$ is characterized using a set of attributes which contains predefined, commonly used attributes and optional attributes to be defined as needed by the modeler for a particular SOS or class of SOSs. $\mathbf{V}$ is instantiated when specifying desired values (in service requests or when advertising services) for various quality dimensions and/or quality dimension aggregates.

While it is established that a quality model should represent the information on quality characteristics and their measurable quality dimensions, as in $\mathbf{Q}$ and $\mathbf{V}$, this information alone is of limited use in dealing with tradeoffs at runtime. Since it is unlikely that various quality dimensions can all be satisfied to the desired extent simultaneously in a given SOS, indications are required on: (i) how various quality dimensions are interdependent, and (ii) what quality dimensions to optimize within a set of interdependent and conflicting quality dimensions (this is of relevance when defining service requests). A quality model is an apparent candidate for including modeling primitives for providing these indications. To respond to (i),

we complement **Q** and **V** with the quality dependency submodel **D**, which is instantiated to represent that some value of a quality dimension is accompanied by some value of another **q**. To address (ii), we include the quality priority submodel **P** which is instantiated to define a (partial or complete) priority ordering over the various quality dimensions. By extending the basic quality modeling information considered in **Q** and **V** with that of **D** and **P**, we enrich the quality information with information relevant for managing tradeoffs at runtime in a SOS. This is a novelty of the QVDP.

Below, each submodel of QVDP is defined for a particular SOS. We thus speak of, e.g., a set of quality dimensions in a submodel instance since the instantiation of the submodel for a given SOS will necessarily involve the definition of a non-empty set of quality dimensions. We believe that this facilitates the understanding of what is obtained when using the QVDP model.

3.3.1 Quality characteristics submodel (Q)

Definition 2. The **Quality Characteristics** submodel for a given SOS

$$\mathbf{Q} \equiv \left\langle \langle \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}, \{\alpha_1, \ldots, \alpha_m\}, f^{\mathbf{q}} \rangle, \left\langle \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}, f^{\bar{\mathbf{q}}} \right\rangle, f^{(\mathbf{q},\bar{\mathbf{q}})} \right\rangle$$

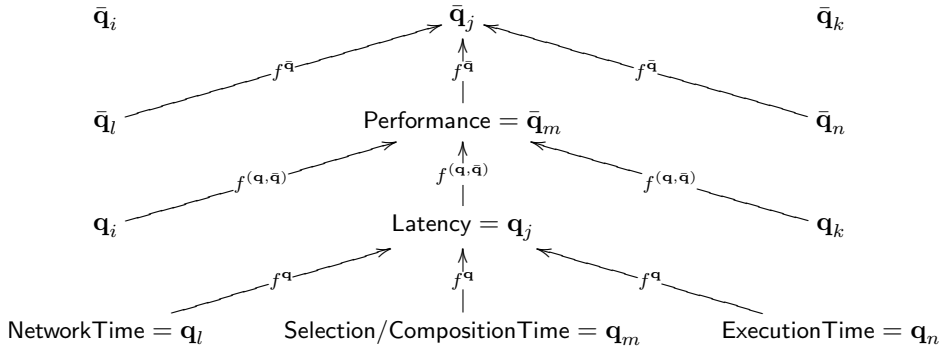
consists of:

1. $\langle \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}, \{\alpha_1, \ldots, \alpha_m\}, f^{\mathbf{q}} \rangle$ which contains a set $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ of quality dimensions defined for the given SOS, a set $\{\alpha_1, \ldots, \alpha_m\}$ of aggregation functions, and a function $f^{\mathbf{q}} : \mathbb{P}(\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}) \times \{\alpha_1, \ldots, \alpha_m\} \longmapsto \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ which maps a set of quality dimensions onto a quality dimension obtained by applying the aggregation function α on the set of quality dimensions. An aggregation procedure α specifies how to aggregate a set of quality dimensions. ($\mathbb{P}$ denotes powerset.)

2. $\langle \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}, f^{\bar{\mathbf{q}}} \rangle$ which contains a set $\{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ of quality characteristics for the given SOS, and a function $f^{\bar{\mathbf{q}}} : \mathbb{P}(\{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}) \longmapsto \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ which maps a sets of quality characteristics onto a quality characteristic which is defined as a group of the former.

3. A function $f^{(\mathbf{q},\bar{\mathbf{q}})} : \mathbb{P}(\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}) \longmapsto \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ which maps a set of quality dimensions onto a quality characteristic. The quality characteristic is defined as the given set of quality dimensions.

Example 1. Among the quality characteristics of the MERIS MGVI Regional service, we look into latency. Let Latency be an aggregate quality dimension obtained by summing the time needed to communicate the service request over the network and to receive the desired output (measured using NetworkTime), the time required for select and compose the services needed to fulfil the request (Selection/CompositionTime), and the time needed to execute the composition (ExecutionTime). Latency is part of the Performance quality characteristic. The following diagram illustrates how these measures fit into the terminology of the above Definition 2, and thus in the instance of **Q** for a given SOS.



Definition 3. A **Quality Dimension q** is a collection of the following attributes and attribute sets:

1. Name gives **q** a unique identifier. This identifier is the name of the quality dimension;

2. Description is a human-readable description of the quality dimension;

3. Purpose indicates why a given quality dimension is defined;

4. Type identifies the type of the quality dimension according to a taxonomy of variable types;
5. Unit defines the unit of measurement for $\mathbf{q}$;
6. Aggregate is given for a $\mathbf{q}$ that is an aggregate of other quality dimensions. This attribute indicates the aggregation procedure and the quality dimensions aggregated to obtain $\mathbf{q}$.
7. MeasurementSource indicates what is measured and how on/in the SOS in order to associate a value to the given quality dimension;
8. MeasurementTransformation defines the algorithm or formula used to transform the data obtained by measurement into the value reported for $\mathbf{q}$ (e.g., average, moving average, etc.);
9. $\{\mathsf{AdditionalAttribute}_1, \ldots, \mathsf{AdditionalAttribute}_n\}$ is the set of n additional attributes defined by the modeler.

A quality dimension can be seen as a metric used to quantify a quality characteristic using a certain (transformed) measurement on a property or behavior of a given SOS. Attributes are used to describe various characteristics of the metric.

Example 2. Latency for the MERIS service is is a collection of the following quality dimensions: NetworkTime, Selection/CompositionTime and ExecutionTime. Let the dimension NetworkTime be an aggregate of SendTime and ReceiveTime, the former being the time for the service request to arrive from the requester to the composer, while the latter is the time for the composition execution output to be sent from the composer to the requestor. The following can describe NetworkTime:

Name	NetworkTime
Description	Average time (over 100 same requests) it takes to send and receive information between a service requestor and a service composer w.r.t. a given service request.
Purpose	Used as an indicator when deciding what quantity of network bandwidth to demand from the bandwidth provider.
Type	Continuous/Ratio
Unit	Millisecond (ms)
Aggregate	NetworkTime = $f^{\mathbf{q}}$({ SendTime, ReceiveTime }, AggregateSum)
Measurement Source	Measurement sources for SendTime: `requestTime` attribute value for a `ServiceRequest` class instance and `requestReceptionTime` attribute value for a `Composer` class instance. Measurement sources for ReceiveTime: `requestCompletionTime` attribute value for a `Composer` class instance and `requestEndTime` attribute value for a `ServiceRequest` class instance.
Measurement Transformation	NetworkTime = $\frac{1}{100}\sum_{i=1}^{100}(\mathit{SendTime}_i + \mathit{ReceiveTime}_i)$

Above, MeasurementSource assumes that a service request is described, among other, with a class called `ServiceRequest` which carries the attribute `requestTime`, and that its value can be collected by, e.g., a monitoring service (or some other SOS component). This value is then used, along with other values (as indicated in the MeasurementSource attribute) to calculate SendTime and ReceiveTime. In other words, MeasurementSource identifies where to find data to calculate quality dimension values.

We place no particular constraints on the structure of a quality characteristic. It is merely a means to organize quality dimensions that are considered as somehow related by the modeler.

Definition 4. A **Quality Characteristic** $\bar{\mathbf{q}}$ is a set of distinct (aggregate) quality dimensions.

3.3.2 Quality value submodel (V)

V is instantiated to define how quality is to be measured on a given SOS. At runtime, it is necessary to provide means for expressing desired values of the various quality dimensions. This information is subsequently used by composers to discriminate among alternative services when performing service composition—services that cannot achieve desired values for a set of quality dimensions will not be selected to participate in a composition. Simplistic models for expressing desired values over quality dimensions involve the definition of a single desired value. This is inappropriate because interdependencies between different behaviors of the givens SOS are likely—it is thus necessary to relate the achievement

of a particular quality dimension value with conditions on the system and/or its operating environment, and conditions that the achievement of some quality dimension value entails in the system and/or its operating environment. Hence the concepts of value precondition and value postcondition in the Quality Value submodel. Uncertainty in system operation leads to the probabilistic characterization of value and value pre/postcondition pairs.

Definition 5. The **Quality Value** submodel $\mathbf{V}$ for a given SOS is a set of p tuples, where each tuple $i = 1, \ldots, p$ is of the form

$$\left\langle \mathbf{q}_i.\mathsf{Name}, \left\{ \left\langle \mathbf{v}_{i,1}, \mathbf{v}_{\mathsf{Pre},i,1}, \mathbf{v}^P_{\mathsf{Pre},i,1}, \mathbf{v}_{\mathsf{Post},i,1}, \mathbf{v}^P_{\mathsf{Post},i,1} \right\rangle, \ldots, \left\langle \mathbf{v}_{i,n}, \mathbf{v}_{\mathsf{Pre},i,n}, \mathbf{v}^P_{\mathsf{Pre},i,n}, \mathbf{v}_{\mathsf{Post},i,n}, \mathbf{v}^P_{\mathsf{Post},i,n} \right\rangle \right\}, \left\{ \mathbf{v}^U_{i,1}, \ldots, \mathbf{v}^U_{i,m} \right\} \right\rangle$$

where:

1. $\mathbf{q}_i.\mathsf{Name}$ is the value of the attribute Name of the quality dimension $\mathbf{q}_i$.

2. $\left\{ \left\langle \mathbf{v}_{i,1}, \mathbf{v}_{\mathsf{Pre},i,1}, \mathbf{v}^P_{\mathsf{Pre},i,1}, \mathbf{v}_{\mathsf{Post},i,1}, \mathbf{v}^P_{\mathsf{Post},i,1} \right\rangle, \ldots, \left\langle \mathbf{v}_{i,n}, \mathbf{v}_{\mathsf{Pre},i,n}, \mathbf{v}^P_{\mathsf{Pre},i,n}, \mathbf{v}_{\mathsf{Post},i,n}, \mathbf{v}^P_{\mathsf{Post},i,n} \right\rangle \right\}$ is the set of values for a given quality dimension $\mathbf{q}_i$, where each tuple associates a particular value $\mathbf{v}$ with a value precondition $\mathbf{v}_{\mathsf{Pre}}$ and the probability $\mathbf{v}^P_{\mathsf{Pre}}$ that the value $\mathbf{v}$ will be achieved if the precondition holds, and a value postcondition $\mathbf{v}_{\mathsf{Post}}$ and the probability $\mathbf{v}^P_{\mathsf{Post}}$ for the postcondition to hold if the given value of $\mathbf{q}$ is achieved.

3. $\mathbf{v}$ declares values for a given $\mathbf{q}$. The values are declared using syntax and semantics which differ depending on whether Type of $\mathbf{q}$ is continuous or discrete. If continuous, the semantics is defined in terms of an interpretation $(\mathbb{C}, \cdot^{c\mathcal{I}})$, which uses the continuous domain $\mathbb{C}$ defined by Type of $\mathbf{q}$, and an interpretation function $\cdot^{c\mathcal{I}}$ which associates with each v a $v^{c\mathcal{I}}$ in $\mathbb{C}$. If discrete, the domain of the interpretation is a set $\mathbb{D}$ which includes all allowed discrete values as defined by the value of Type of $\mathbf{q}$, and the interpretation function $\cdot^{d\mathcal{I}}$ associates with each e a $e^{d\mathcal{I}}$ in $\mathbb{D}$. Below, the syntax, semantics, and syntax rules are given for the continuous and discrete case.

Type of $\mathbf{q}$ *is continuous.*		Type of $\mathbf{q}$ *is discrete.*	
Syntax	Semantics	Syntax	Semantics
v	$v^{c\mathcal{I}} \in \mathbb{C}$	e	$e^{d\mathcal{I}} \in \mathbb{D}$
$\neg v$	$\mathbb{C} \setminus v^{c\mathcal{I}}$	$\neg e$	$\mathbb{D} \setminus e^{d\mathcal{I}}$
$(\geq v)$	$\{v'^{c\mathcal{I}} \mid v'^{c\mathcal{I}} \geq v^{c\mathcal{I}}\}$	E	$E^{d\mathcal{I}} \subseteq \mathbb{D}, E^{d\mathcal{I}} = \{e^{d\mathcal{I}} \mid \forall e \in E\}$
$\neg (\geq v)$	$\mathbb{C} \setminus (\geq v)^{c\mathcal{I}}$	$\neg E$	$\mathbb{C} \setminus E^{d\mathcal{I}}$
$(\leq v)$	$\{v'^{c\mathcal{I}} \mid v'^{c\mathcal{I}} \leq v^{c\mathcal{I}}\}$	$E_i \vee E_j$	$E_i^{d\mathcal{I}} \cup E_j^{d\mathcal{I}}$
$\neg (\leq v)$	$\mathbb{C} \setminus (\leq v)^{c\mathcal{I}}$	$E_i \wedge E_j$	$E_i^{d\mathcal{I}} \cap E_j^{d\mathcal{I}}$
$(\leq v) \vee (\geq v)$	$(\leq v)^{c\mathcal{I}} \cup (\geq v)^{c\mathcal{I}}$		
$(\leq v) \wedge (\geq v)$	$(\leq v)^{c\mathcal{I}} \cap (\geq v)^{c\mathcal{I}}$		
Syntax formation rules		Syntax formation rules	
$\lambda^c ::= v \mid (\geq v) \mid (\leq v)$		$\lambda^d ::= e \mid E$	
$\Lambda^c ::= \lambda^c \mid \neg\Lambda^c \mid \Lambda^c_i \vee \Lambda^c_j \mid \Lambda^c_i \wedge \Lambda^c_j$		$\Lambda^d ::= \lambda^d \mid \neg\Lambda^d \mid \Lambda^d_i \vee \Lambda^d_j \mid \Lambda^d_i \wedge \Lambda^d_j$	
$\mathbf{v} ::= \Lambda^c$		$\mathbf{v} ::= \Lambda^d$	

4. $\mathbf{v}_{\mathsf{Pre}}$ gives an assertion which describes the precondition for the value(s) defined in $\mathbf{v}$ to be achieved.

5. $\mathbf{v}^P_{\mathsf{Pre}}$ indicates the probability that a value in $\mathbf{v}$ will be achieved if the precondition $\mathbf{v}_{\mathsf{Pre}}$ holds.

6. $\mathbf{v}_{\mathsf{Post}}$ gives an assertion which describes the postcondition that holds after achieving a value in $\mathbf{v}$.

7. $\mathbf{v}^P_{\mathsf{Post}}$ indicates the probability that the postcondition $\mathbf{v}_{\mathsf{Post}}$ holds if a value in $\mathbf{v}$ is achieved.

8. $\left\{ \mathbf{v}^U_{i,1}, \ldots, \mathbf{v}^U_{i,m} \right\}$ is the set of partial preference orderings $\mathbf{v}^U \equiv (\cdot) \overset{\cup}{\succeq} (\cdot)$ on values for a given $\mathbf{q}_1$. $\overset{\cup}{\succeq}$ defines a partial order (i.e., $\overset{\cup}{\succeq}$ is a reflexive, transitive, and anti-symmetric relation). Both $(\cdot)$ follow the syntax and semantics of either Λ^c or Λ^d (depending on whether Type is continuous or discrete). Values specified on the left hand side of $\overset{\cup}{\succeq}$ are preferred at least as much as the values given on the right hand side of the preference ordering relation. Strict ordering is defined in a strainghtforward manner (i.e., $x \overset{\cup}{\succ} y \equiv x \overset{\cup}{\succeq} y \wedge \neg(y \overset{\cup}{\succeq} x)$).

Example 3. The following provides the instantiation of the quality value submodel for one value of NetworkTime, one of the dimension aggregated to define the Latency characteristic. Preconditions and postconditions are normally written in a formal language (e.g., the Semantic Web Rule Language (SWRL) [80]). Below, natural language and simple structured expressions are used to avoid introducing more formalism in this chapter. The probabilities are usually estimated and continually updated at runtime.

$\mathbf{q}$.Name	NetworkTime
$\mathbf{v}$	$(\leq 3500ms) \wedge (\geq 1500ms)$
$\mathbf{v}_{\mathsf{Pre}}$	ConnectionFailureProbability ≤ 0.05
$\mathbf{v}^{P}_{\mathsf{Pre}}$	70%
$\mathbf{v}_{\mathsf{Post}}$	Change of network bandwidth unnecessary.
$\mathbf{v}^{P}_{\mathsf{Post}}$	90%
$\mathbf{v}^{U}$	$((\leq 350Oms) \wedge (\geq 1500ms)) \overset{\mathsf{U}}{\succ} (> 3500ms)$

3.3.3 Quality dependency submodel (D)

The quality dependency submodel is instantiated to express interdependencies between values of distinct quality dimensions. As noted earlier, the dependency submodel is of particular relevance for managing tradeoffs: it is the quality dependency model that makes explicit the possible tradeoffs, and summarizes how values of quality dimensions are related so that the outcome of tradeoffs can be anticipated.

Definition 6. The **Quality Dependency** submodel $\mathbf{D} \equiv \{\mathbf{d}\}$ for a given SOS is a set of dependency relations for pairs of distinct quality dimensions. Each dependency relation is written using the following formation rule:

$$\mathbf{d} ::= \Big(\mathbf{q}_i.\mathsf{Name} \xrightarrow{\Lambda_k|\Lambda_l} \mathbf{q}_j.\mathsf{Name}\Big)\ @(\phi, P) \mid \Big(\mathbf{q}_i.\mathsf{Name} \xleftrightarrow{\Lambda_k|\Lambda_l} \mathbf{q}_j.\mathsf{Name}\Big)\ @(\phi, P)$$
$$\mid \Big(\mathbf{q}_i.\mathsf{Name} \xleftrightarrow{f} \mathbf{q}_j.\mathsf{Name}\Big)\ @(\phi, P)$$

where:

- $\mathbf{q}_i$.Name is the value of the Name attribute of $\mathbf{q}_i$ and $\mathbf{q}_j$.Name is the value of the Name attribute of some other quality dimension $\mathbf{q}_i$.
- Syntax and semantics of Λ_k follow that of Λ^c if the quality dimension $\mathbf{q}_i$ is of continuous type; if not, then the syntax and semantics of Λ^d are followed. Same applies for Λ_l and $\mathbf{q}_j$.
- The dependency is directed if Λ_k and Λ_l are related with "$\longrightarrow$", so that it if the value Λ_k of $\mathbf{q}_i$ is achieved, the value specified in Λ_l will be achieved for $\mathbf{q}_j$ at the probability P. "$\longleftrightarrow$" indicates that the dependency is directed both ways: if either of the quality dimensions reaches its given value Λ, the other will also have its value in its given Λ, at the probability P.
- When the relationship between the values of two quality dimensions can be described with a function, f gives that functional relationship.
- As the dependency may only be relevant when particular conditions hold, a condition ϕ can be added to indicate when the interaction applies (otherwise, ϕ remains empty). The condition is written as an assertion in the language used to write value precondition and value postcondition.
- P is a value that designates the probability for the dependency to be actually observed. It is usually estimated at runtime.

Example 4. The motivating example makes appear that the availability of the service is calculated with the help of the dimension DownTime. Similarly, the reliability depends on FailedAttempts. It seems clear that the number of failed attempts will be influenced by the down time of the system. The probability to observe this particular dependency of values is 0.8:

$$\Big(\mathsf{DownTime} \xrightarrow{(increases)|(increases)} \mathsf{FailedAttempts}\Big)\ @(0.8)$$

3.3.4 Quality priority submodel (P)

In addition to **D**, **P** is another novelty of the proposed QVDP model. When a pair of quality dimensions is such that the values of both cannot be simultaneously optimized when, e.g., seeking appropriate service compositions, priority must be known over the pair in order to know which of the two to optimize at the expense of the other. The quality priority submodel is instantiated in order to make explicit the priority order between pairs of quality dimensions. Clearly, and as highlighted earlier, the priority submodel combines with the dependency submodel when managing tradeoffs: the latter indicates what pairs of quality dimensions are involved in tradeoffs, while the former indicates what to decide when tradeoff is to be performed between quality dimensions.

Definition 7. The **Quality Priority** submodel

$$\mathbf{P} \equiv \left\langle \{\langle \mathbf{p}_1^{\mathbf{q}}, \phi_1 \rangle, \ldots, \langle \mathbf{p}_n^{\mathbf{q}}, \phi_n \rangle\}, \{\langle \mathbf{p}_1^{\bar{\mathbf{q}}}, \phi_1 \rangle, \ldots, \langle \mathbf{p}_m^{\bar{\mathbf{q}}}, \phi_m \rangle\} \right\rangle$$

for a given SOS contains a set of (conditioned) priority orders for pairs of distinct quality dimensions, and a set of (conditioned) priority orders for pairs of quality characteristics. For quality dimensions, each priority order is $\mathbf{p}^{\mathbf{q}} \equiv \mathbf{q}_i.\mathsf{Name} \overset{\mathsf{P}}{\succeq} \mathbf{q}_j.\mathsf{Name}$, where $\mathbf{q}_i.\mathsf{Name}$ is different from $\mathbf{q}_j.\mathsf{Name}$. The given priority order indicates that optimizing $\mathbf{q}_i$ is important at least as much as optimizing $\mathbf{q}_j$. For each quality characteristic, the priority is given as $\mathbf{p}^{\bar{\mathbf{q}}} \equiv \bar{\mathbf{q}}_i \overset{\mathsf{P}}{\succeq} \bar{\mathbf{q}}_j$. $\overset{\mathsf{P}}{\succeq}$ is a partial order; strict priority is defined as usual (i.e., $x \overset{\mathsf{P}}{\succ} y \equiv x \overset{\mathsf{P}}{\succeq} y \wedge \neg(y \overset{\mathsf{P}}{\succeq} x)$). A priority order is conditioned if the optional ϕ is specified. If ϕ holds, the given priority order applied. ϕ is written as an assertion in the language used to write value precondition and value postcondition.

Example 5. A requester of a service providing information such that those provided by the MERIS MGVI Regional web service can instantiate the quality priority submodel to indicate in a service request that it is strictly more important to him to optimize reliability than latency, i.e., $\mathbf{p}_i^{\bar{\mathbf{q}}} = \mathsf{Reliability} \overset{\mathsf{P}}{\succ} \mathsf{Latency}$. This priority reflecting the fact that due to the long execution time expected for extracting the result, the requestor prefers to insure himself that the service will properly achieve and wait longer than the opposite. At the level of quality dimensions, the same request can indicate, e.g., that it is strictly more important to optimize the network time than to optimize the selection and composition time (i.e., the requester is interested in having the request transmitted rapidly, even if this entails longer time to find the appropriate services and their composition): $\mathbf{p}_k^{\mathbf{q}} = \mathsf{NetworkTime} \overset{\mathsf{P}}{\succ} \mathsf{Selection/CompositionTime}$.

3.4 Comparison with Prior Quality Models

Table 3.1 summarizes the comparison. It indicates concepts and constructs of prior quality ontologies and models that correspond to those of **QVDP**. The table highlights the novelty of including submodels and constructs for representing preference, dependency, and priority among quality dimensions and characteristics.

Only some of the many approaches capable to describe service quality are considered here. Other approaches include: QuA [199], QML [58], WSOL [205], UniFrame [24], CORBA Trading Object Service [156], QuO [126], SLAng [195]. They are not overviewed here for two reasons: first, previous comparisons (e.g., [195]) indicate that the fragments of their models specialized for service QoS description are subsets of the set of concepts and constructs present in the approaches considered herein; second, the aim here is only to identify key concepts and constructs manipulated when specifying a service's QoS, so that we do not discuss how each of the concepts adapts to a particular implementation framework – therefore, we do not consider in this section the results focused on adopting and adapting the reviewed approaches to particular implementation technologies and frameworks.

In Table 3.1, "$\surd$" indicates that the given concept or construct can be expressed in the relevant quality model, but the name of concept or construct for doing so cannot be identified in the cited work. "$\times$" indicates that the given QVDP concept or construct has no corresponding concept or construct in the relevant quality model.

Table 3.1: Comparison of **QVDP** with a selection of prior quality models.

QVDP	Q-WSDL	WSLA	DAML-QoS	Maximilien and Singh	Zeng and colleagues
$\mathbf{Q}$	QoS profile	Service Level Agreement	QoS Profile	√	Quality vector
$\mathbf{q}$	QoS dimension	Metric	Metric	Quality	Quality criterion
Name	name	name	metricName	√	√
Description	definition	√	√	√	√
Purpose	×	×	×	×	√
Type	type	type	√	√	√
Unit	unit	unit	√	√	√
Aggregate	×	√	√	√	√
Measurement Source	source	Function or Measurement Directive	√	√	√
Measurement Transformation	×	√	Function	AggregateQuality	√
Additional Attribute	×	×	×	QAttribute	×
α	×	Function or Measurement Directive	√	√	√
$f^{\mathbf{q}}$	×	√	√	√	√
$\bar{\mathbf{q}}$	QoS characteristic and QoS category	QoS Property	×	×	×
$f^{\bar{\mathbf{q}}}$	×	×	×	×	×
$f^{(\mathbf{q},\bar{\mathbf{q}})}$	√	√	×	×	×
$\mathbf{V}$	×	Service Level Objectives or Action Guarantees	QoS Precondition, QoS Effect, QoS constraints	QoS Policy	×
$\mathbf{v}$	value	√	√	QValue typical	√
$\mathbf{v}_{\mathsf{Pre}}$	×	√	√	×	×
$\mathbf{v}^{P}_{\mathsf{Pre}}$	×	×	×	×	×
$\mathbf{v}^{\mathsf{Post}}$	×	√	√	×	×
$\mathbf{v}^{P}_{\mathsf{Post}}$	×	×	×	×	×
$\mathbf{v}^{U}$	direction	√	×	×	√
$\mathbf{D}$	×	×	×	QRelationship	√
$\mathbf{d}$	×	×	×	ValueImpact, ValueDirection	√
$\mathbf{P}$	×	×	×	×	√
$\mathbf{p}^{\mathbf{q}}$	×	×	×	×	√
ϕ (for $\mathbf{p}^{\mathbf{q}}$)	×	×	×	×	×
$\mathbf{p}^{\bar{\mathbf{q}}}$	×	×	×	×	√
ϕ (for $\mathbf{p}^{\bar{\mathbf{q}}}$)	×	×	×	×	×

3.4.1 Q-WSDL

D'Ambrogio proposes Q-WSDL [41], a Quality of Service[1] (QoS) extension to WSDL [39] which includes a set of QoS characteristics accepted in a UML extension for specifying QoS [157]. In Q-WSDL, a service's QoS is described through QoS characteristics (e.g., Availability), each quantified through QoS dimensions (e.g., UpTime). A QoS dimension is described with a value, unit of the value, source of measurement (e.g., measured, assumed, predicted, etc.), and optionally, a type of statistical value (e.g., mean, variance, etc.) and an order relation to compare values (i.e., to answer which values are preferred). QoS categories bring together QoS characteristics related to a common subject—e.g.: reliability and availability are grouped in the dependability category.

Q-WSDL highlights the need to specify preferences over quality values, yet remains limited in doing so compared to QVDP. Namely, Q-WSDL allows specifying the desired direction for the value of the quality dimension, but cannot deal with value preconditions and value postconditions, or uncertainty thereof (i.e., through probabilities). Q-WSDL is not extensible, in that it has no apparent mechanism for introducing additional attributes to finely describe a quality dimension. In contrast to all other models noted in Table 3.1, Q-WSDL omits aggregation of quality characteristics. Note that lack of means to represent dependency and priority information in Q-WSDL entails that any directives on tradeoffs need

[1] According to the relevant ISO standard [86], QoS refers to characteristics that contribute to the overall quality of a service as perceived by the consumer of the service. A QoS characteristic is a quantifiable aspect of QoS which is defined independently of the means by which it is represented, managed, or controlled.

to be specified outside the quality model. Additional representation formalism (unspecified in case of Q-WSDL) is thus necessary, and further effort is consequently required for ensuring consistency between the information in the quality model and the tradeoff formalism. Such problems are avoided in QVDP.

3.4.2 WSLA

While focusing on the contracting between the service provider and requester, the Web Service Level Agreement language [109] integrates similar kind of information as the Q-WSDL when describing quality characteristics. The Web Service Level Agreement (WSLA) language [109] focuses on the contracting between the service provider and requester. It is used to specify quality of service within a Service Level Agreement (SLA). A quality dimension is expressed by a SLA parameter and described with a name, type, unit, definition and purpose. The value of a parameter is quantified by a metric, whereby a metric can be an aggregate of other metrics. If aggregate, it is given by a function or a measurement directive; a function uses other metrics as operands while a measurement directive specifies how an individual metric is retrieved from the source. For example, a function can be such that the value of the metric used to calculate the Availability characteristic can be obtained by dividing the value of UpTime by the value of the sum of UpTime and DownTime. Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message, or the command for invoking scripts of compiled programs. The notion of value specification is similar to the concept of obligation which define guarantees and constraints on SLA parameters. These obligations cover the service level objectives that represent promises with respect to the state of SLA parameters and the action guarantees that are promises of a signatory party to perform an action. Value preconditions and postconditions in QVDP enable can be used to express such constraints-related information. A service level objective expresses a commitment to maintain a particular state of the service in a given period, e.g., the SLA parameter TimeNeededToTransferData must be lower than 100 ms if the SLA parameter NetworkTime is less than 30 ms. An action guarantee expresses a commitment to perform a particular activity if a given precondition is met; e.g., sending an event to one or more signatory party and supporting parties, opening a problem report, performing the payment of a penalty, or of a premium, and so on.

While close to QVDP in terms of specifying quality dimensions and quality characteristics, and describing values thereof, WSLA is similar to Q-WSDL in not providing indications on how to deal with tradeoffs. This is an important limitation, since a contract between a provider and a requester can involve situations in which tradeoffs need to be managed. For instance, adding dependency information would allow the provider to check whether some requested levels of quality dimensions can be realized: knowing their interdependency, the provider might highlight that the requested levels are unrealistic, or that more desirable levels can be achieved. Adding priority indications can lead to more extensive contracts, in which contingencies can be managed in a finer way: e.g., if the provider indicates that in some situation particular levels of some metrics cannot be achieved together, the requester might indicate which of the metrics is to be optimized in place of others when such a situation occurs. Both dependency and priority information thus appear relevant when contracting between the provider and the requester, but are absent from the WSLA.

3.4.3 DAML-QoS

Zhou and colleagues [233] extend DAML-S with a means for describing QoS in a generic manner. Apart from the usual definition of concepts for metrics, metric types, and metric aggregation, their DAML-QoS ontology introduces the possibility for defining QoS characteristics whose value at the service's input is different from its value at output (e.g., in a converter, input and output bit rates differ). Since QVDP is not directly associated to a particular model of the service-oriented system, relating service inputs and outputs to different values can be accomplished by instantiating the quality value submodel, and defining a function to map specific values to inputs or outputs of a given service. Also, the service may only be capable of achieving some specific quality level if some external quality level is satisfied (e.g., for two interacting services, a minimal throughput rate at the first service might be needed if the second service

is to ensure some desired throughput rate), so that a condition on QoS characteristics can be defined (as a QoSPrecondition). Similarly, the level of quality achieved over the service's various QoS characteristics may change the effects of the service—the effect of QoS can be defined as QoSEffect in DAML-QoS. Both of these are supported with value preconditions and postconditions in QVDP.

While DAML-Qos is innovative in terms of value preconditions and value postconditons when compared to other prior quality models, DAML-QoS cannot deal with preferences over values, dependencies, and priorities. It thus suffers from the same limitations as Q-WSDL and WSLA when contrasted to QVDP: we see no explicit means in DAML-QoS to address tradeoffs at runtime.

3.4.4 Maximilien and Singh

Maximilien and Singh [134] describe service QoS through an ontology which abstracts from particular QoS characteristics: each QoS characteristic is called a quality, is associated to a typed variable, indications on how it is measured, and its relationships to other qualities (in terms of strength and direction). Namely, QMeasurement quantifies a quality while AggregateQuality combines several Qmeasurements into aggregate metrics. QRelationships indicate related qualities in a manner that shows through their values. A quality related to another has a valueimpact (weak, mild and strong) and a valuedirection (opposite or parallel). A QRelationship is similar to a dependency specification, though the latter allows more precise information to be given on relationahips between metrics. The QoSPolicy specification indicates a level of commitment of the provider to the advertised policy (bestEffort, guaranteed, notSpecified or noGuarantee). As QoSPolicy aims at add constraints on the value, so that it is equivalent to the value specification.

An important characteristic of Maximilien and Singh's approach is the integration of information on relationships between qualities using the notions of value impact and value direction. Their approach qualifies value impact as either weak, mild, or strong, while value direction is described as either opposite or parallel. QVDP thus covers their dependency representation, adds the possibility to indicate conditions for interdependencies and uncertainty thereon, and allows more detail in describing interactions. Again, as Q-WSDL, WSLA, and the DAML-QoS, Maximilien and Singh's model has no means to deal with priority.

3.4.5 Zeng and colleagues

Zeng and colleagues [228] use an ad-hoc informal quality model in which they represent price, execution duration, reputation, reliability and availability to guide service composition at runtime. A quality of service is expressed by means of a quality vector. The vector is composed of quality criteria that we define with quality dimensions. Each of the criteria is described similarly to our quality dimension, that is, by a name, a unit, a type, a short definition and an (optional) expression defining how to perform the measurement. The desired values are given with modalities which depend on the quality's purpose; e.g., execution price is minimized whereas reliability is maximized. The idea is the same in the preference fragment of our quality value submodel where an order relation is given between possible values. Zeng and colleagues represent priority by associating weights to each of the criteria used in the aggregation function which is optimized when performing service composition.

Zeng and colleagues' approach is closest to QVDP in terms of dependency and priority representation, although it is comparatively limited when dealing with value descriptions. Both interdependency and priority representations in Zeng and colleagues' proposal remain implicit from the specific set of quality criteria they use during composition. In this respect, theirs is not a quality model per se, but a particular case of an implicit quality model. In this sense, they do not discuss a quality model, but only specific quality dimensions and their application in service composition. In addition, QVDP differs in the detail it allows—e.g., we can specify conditions for priority orderings to apply while this is not considered by Zeng and colleagues.

3.5 QVDP and QoS in UML

Among the related efforts, the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [157] stands out in its coverage of various concepts and constructs for conceptualizing quality and its status of standard. Therein, a metamodel is proposed as an extension to the UML metamodel to support the definition of QoS properties for systems of which other aspects are modeled with UML. In this section, we review that metamodel, compare it to QVDP, and extend the UML QoS metamodel with concepts and constructs available in QVDP and unavailable there. Finally, we propose a case study using the UML QoS Framework and our extensions derived form the service presented in section 3.2.

3.5.1 Elements of the metamodel

The UML QoS Framework metamodel includes different submetamodels describing the QoS extension for UML. The QoSCharacteristics package contains the elements required to define QoSCharacteristics and QoSDimensions. The QoSConstraints package comprises the modeling elements used to describe QoSContracts and QoSConstraints. The last package, QoSLevels covers components specifying QoSModes and QoSTransitions. We review these packages below:

- **QoSCharacteristics package**
  - **QoSCharacteristic**
    A QoSCharacteristic is a description for some quality consideration, such as, e.g., latency, availability, reliability, capability. A characteristic is quantified by means of specific parameters and methods. These concepts are provided by the metaclass QoSParameter. Extensions and specializations of such elements are available with the sub-parent self-relation. A characteristic has the ability to be derived into various other characteristics as suggested by the templates-derivations self-relation. The attribute isInvariant indicates if the value of the characteristic can be dynamically updated.
  - **QoSDimension**
    A QoSDimension specifies a measure that quantifies a QoSCharacteristic. The attribute direction defines the direction (increasing, decreasing) in which it is desired that the value of the QoSDimension moves. Unit and statistiqualQualifier attributes specify, respectively, the unit for the value dimension and the type of the statistical qualifier; e.g.: maximum value, minimum value, range, mean, frequency, distribution, etc.
  - **QoSCategory**
    QoSCategories are used to group QoSCharacteristics related to the same abstract quality consideration or topic, such as, e.g., performance or security. While performance may group, e.g., latency and trouhgput, security might bring together, e.g., reliabiality and availability. QoSCategories are therefore not quantifiable themselves, but rely on the quantification of their components.
  - **QoSValue**
    QoSValues are instantiations of QoSDimensions that define specific values for dimensions depending on the value definitions given in QoSDimensionSlots.
  - **QoSDimensionSlot**
    A QoSDimensionSlot represents the value of QoSValue. It can be either a primitive QoSDmension or a referenced value of another QoSValue.
  - **QoSContext**
    While constraints usually combine functional and non-functional considerations about the system, QoSContext is used to describe the context in which quality expression are involved. A context includes several QoSCharacteristics and model elements. A single QoSCharacteristic defines the QoSContext for expressions whose references are only to this QoSCharacteristic.

The attribute isQoSObservation defines that a QoSContext represents an environment of quality observation. The quality observation records the values of characteristics included in the relation BasedOn. This way, constraints including more than one quality characteristic can be represented. The main aim of constraints is to limit the set of allowed values of characteristics.

- **QoSConstraints Package**
  - **QoSConstraint**
    The aim of QoSConstraints is to restrict values of QoSCharacteristics. Constraints describe limitations on characteristics of modeling elements identified by application requirements and architectural decisions. Constraints rely on contexts which establishes the QoS characteristics and functional element that can be involved in the constraints. To limit allowed values, constraints put maximum and minimum values to characteristics as well as dependencies between characteristics. These quality constraints can be seen from provider's and client's point of view leading to approaches named "constraints provided" and "constraints offered". The attribute qualification refers to the nature of the constraint, with the following possible values: guaranteed, best-effort, treshold-best-effort, compulsory-best-effort, and none. Each constraint is associated to at least one QoSContext which references values related to the constraint.
  - **QoSRequired**
    Required QoSConstraints can be defined either by the provider either by the client. When the requirements are defined by the client, the provider must support the required quality that fulfill the client's required constraints. This constraint limits the set of values the client accepts for the given characteristic. The required constraints can also be defined by the provider, in this case, the client must achieve some required level of quality to obtain the quality that the provider offers.
  - **QoSOffered**
    The set of QoSOffered by a client or a provider defines its interface—that is, it advertises the qualities for which the offered component is designed. Evidently then, quality is not guaranteed for characteristics that do not appear in QoSOffered.
  - **QoSContract**
    QoSContract is assembles client and provider constraints. In general, client required QoS need to be subsets of provider offered QoS and similarly, provider required QoS need to be subsets of client provided QoS. If no matching is possible between offered and provided constraints, the contract needs to be negotiated between parties involved.
  - **QoSCompoundConstraint**
    A QoSCompoundConstraint is a set of constraints that together represent a constraint for one model element. Another purpose of compound constraints is to allow the representation of a global constraint composed of a set of subconstraints. This way, we can define a precedence relation between subconstraints, to represent, e.g., how to decompose a latency constraint in a set of subconstraints.
- **QoSLevelsPackage**
  - **QoSLevel**
    Depending of available infrastructure and particular algorithms, a service can be executed to several working modes; each working mode provides different qualities for the same services. A QoSLevel is intended to represent a mode of QoS that a service can support, so that a QoSLevel is associated to each of these working modes.
  - **QoSTransition**
    A QoSTransition specifies an allowed transition between QoSLevels.
  - **QoSCompoundLevel**
    A QoSCompoundLevel includes all QoSLevels involved in the quality behavior of a service.

Table 3.2: Comparison of the QVDP and the UML QoS Framework Metamodel.

QVDP	UML QoS
Q	QoSContext
q	QoSDimension
Name	$\surd$
Description	$\times$
Purpose	$\times$
Type	$\surd$
Unit	unit
Aggregate	$\times$
Measurement Source	QoSDimensionSlot
Measurement Transformation	statisticalQualifier
α	$\times$
$f^{\mathbf{q}}$	$\times$
$\bar{\mathbf{q}}$	QoSCharacteristic and QoSCategory
$f^{\mathbf{q}}$	sub-parent relationship
$f^{(\mathbf{q},\bar{\mathbf{q}})}$	QoSParameter
V	$\times$
v	QoSConstraint
$\mathbf{v}_{\mathsf{Pre}}$	Provider QoSRequired
$\mathbf{v}^{P}_{\mathsf{Pre}}$	$\surd$ (QoSLevel)
$\mathbf{v}^{\mathsf{Post}}$	QoSOffered
$\mathbf{v}^{P}_{\mathsf{Post}}$	$\surd$ (QoSLevel)
$\mathbf{v}^{U}$	direction
D	$\surd$
d	QoSConstraint
P	$\times$
$\mathbf{p}^{\mathbf{q}}$	$\times$
ϕ (for $\mathbf{p}^{\mathbf{q}}$)	$\times$
$\mathbf{p}^{\mathbf{q}}$	$\times$
ϕ (for $\mathbf{p}^{\mathbf{q}}$)	$\times$

3.5.2 Comparison of the QVDP and the UML QoS Framework Metamodel

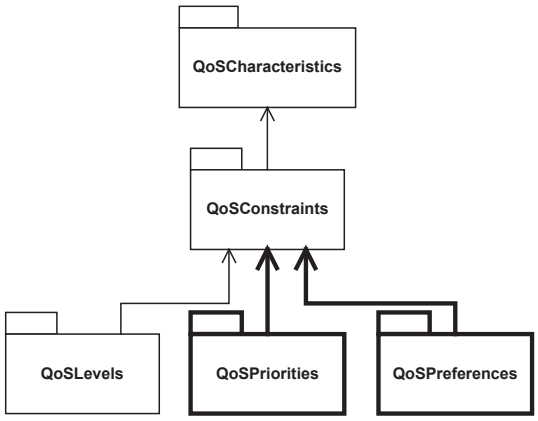
Table 3.2 sumarizes the comparison. In both models, QoS characteristics are quantified by dimensions which own unit, type and name. In the UML QoS Framework metamodel, description and purpose do not appear in the class describing dimensions. UML provides a way to define a parent characteristic from sub-characteristics but does not allow the composition of a dimension from other dimensions. This way, no aggregate function is provided by the QoSDimension metaclass. The QoSParameter metaclass defines how a characteristic is composed from dimensions. The measurement transformation concept appears in UML but possibilities are limited; the statistical qualifier only compute values with a modality (max-value, minvalue, mean, and similar). The source of measurement is represented by the QoSDimensionSlot metaclass in the UML QoS Framework metamodel. Valid values are limited by means of QoSConstraints elements; provider's required constraints allow to highlight preconditions while provider's offered constraints can define postconditions. The probability associated to these conditions can be given using QoSLevel. Even if levels do not define directly probabilities, they take into account available resources to determine if constraints can be respected. The attribute direction indicates preference relations over values for a same dimension. We can write dependencies as constraints over values of different dimensions. UML QoS does not provide any elements enabling to point up priorities among dimensions or characteristics.

We see that the UML QoS Framework metamodel can be usefully extended with concepts and constructs from the QVDP. The extension is outlined below.

3.5.3 Extending the UML QoS Framework Metamodel

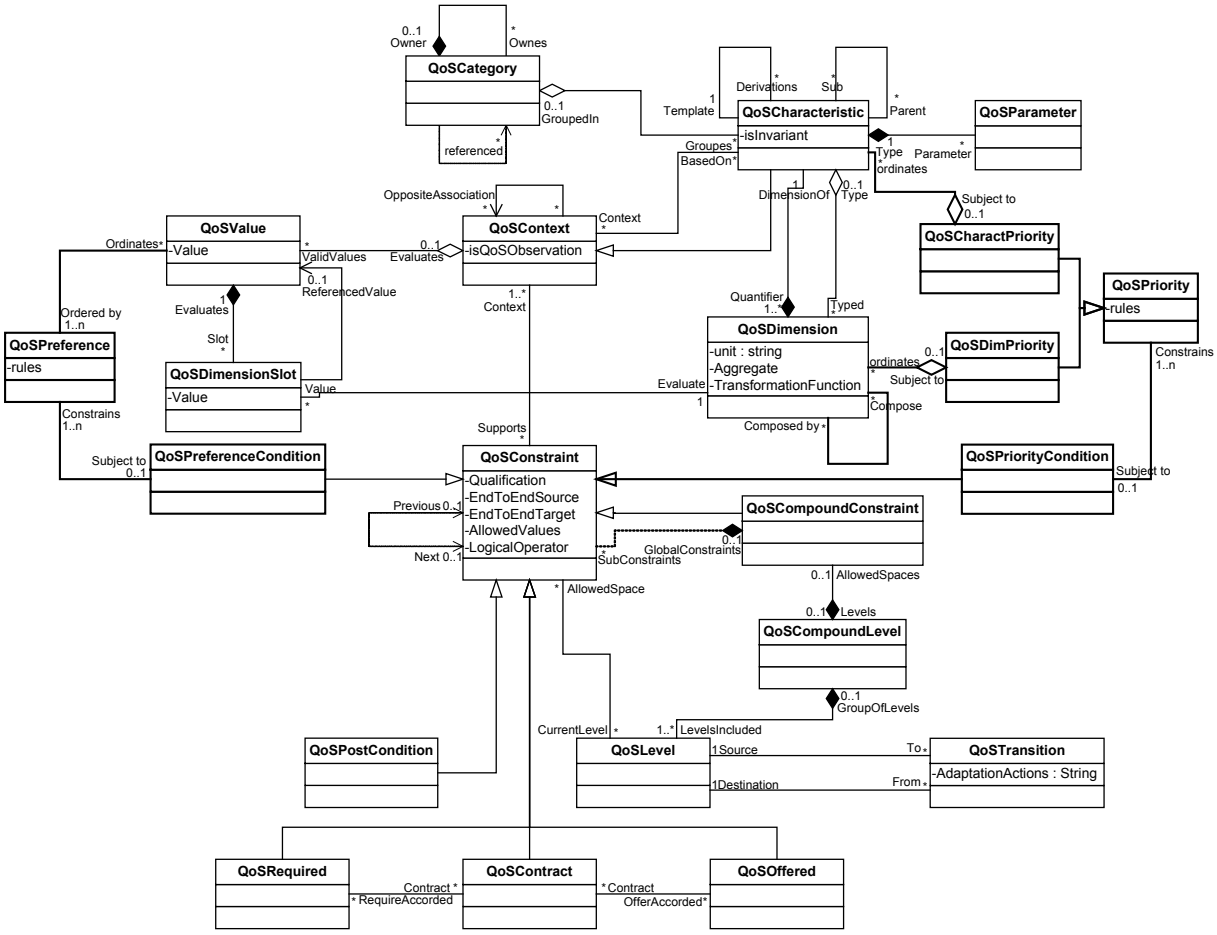
Figure 3.3 shows the submodels of the extented UML QoS Framework metamodel. Constraints are defined over characteristics and levels and priorities are specifications of constraints. QoSCharacteristics, QoSConstraints and QoSLevels were defined in the original metamodel while QoSPriorities and QoSPreferences are added submodels allowing to introduce, respectively, the concepts of priorities and preferences. Besides these submodels, various extensions have been added to the model in order to express all elements available in QVDP. Figures 3.3 and 3.4 summarize the extended UML QoS Framework Metamodel. Changes and extensions are given below.

Figure 3.3: Submodels of extended UML QoS Framework metamodel. Extensions are in bold.



- **Priorities submodel**
  As the UML QoS Framework metamodel does not account for priorities over quality elements, the submodel presented here is an extension introducing extra classes to specify priorities over characteristics and over dimensions.
  - **QoSPriority**
    The main class of this submodel is used to express rules that define priorities over characteristics or dimensions. These rules determine the order at which characteristics or dimensions are considered for optimization when services are being selected. A rule defines an order relation between elements. Different methods can be used in order to rank characteristics or dimensions; simple precedence relation can be established or weighted functions over elements can be used to account for some criteria.
  - **QoSDimPriority** and **QoSCharactPriority**
    These classes are specializations of QoSPriority defining specific elements for priorities over, respectively, characteristics and dimensions.
  - **QoSPriorityCondition**
    Conditions on priorities are constraints specifying when priorities hold. We use this element to specify, e.g., the priority that holds only if some value over a quality dimension is achieved.
- **QoSPreferenceSubmodel**
  We use the QoSPreferenceSubmodel to write preferences over values. In the original UML QoS Framework metamodel, preferences are defined by means of an attribute that indicates the preferred value direction (i.e., increase or decrease). The introduction of the QoSPreference submodel makes the framework considerably more expressive; e.g.: set of values can be prefered over others under some specific conditions.
  - **QoSPreference**
    The purpose of QoSPreference class is to sort values of dimensions. The sorting is established by rules determining a precedence order between values. Rules can delimit precedence over disjoint sets of value and not only following a modality as previously proposed with the direction attribute.
  - **QoSPreferenceCondition**
    The QoSPreferenceCondition class is a specialization of the QoSConstraint class. It indicates conditions for the preference on values to hold.
- **Aggregate** and **TransformationFunction** In the original UML QoS Framework metamodel, the value calculation is specified in the attribute statisticalQualifier of the QoSDimension class. That approach allows us only to define the modality under which the value of a dimension can be

Figure 3.4: UML QoS Metamodel with proposed extensions



calculated on a set value. In QVDP, it is possible to define a dimension from other dimensions, so that the value calculation has to be more expressive. We augment expressivity by replacing statisticalQualifier with two attributes: Aggregate which defines from which other dimensions the value is calculated and TransformationFunction which provides the formula to compute the value of the dimension.

- **Compose-composed relationship**
  In QVDP it is possible to compose a dimension from other dimensions as for characteristics. This possibility is expressed in the metamodel thanks to the compose-composedby self-relationship of QoSDimension class.

- **QoSPostCondition** As no element in the original UML QoS Framework metamodel allows us to specify the postcondition expressed in QVDP, this possibility is added under the form of a class which is a specialization of the QoSConstraint class.

3.5.4 Case study

As suggested by the OMG, the UML metamodel can be used by service requestors and providers to define their respective requests and capabilities about QoS. We propose in this case study some examples of utilization of the UML QoS Metamodel to express such information. We refer to the service ENVISAT/MERIS MGVI Regional introduced in the section 3.2 to illustrate how to use the metamodel and its submodels to organize QoS advertising specification.

Figure 3.5: UML QoS Characteristics submodel

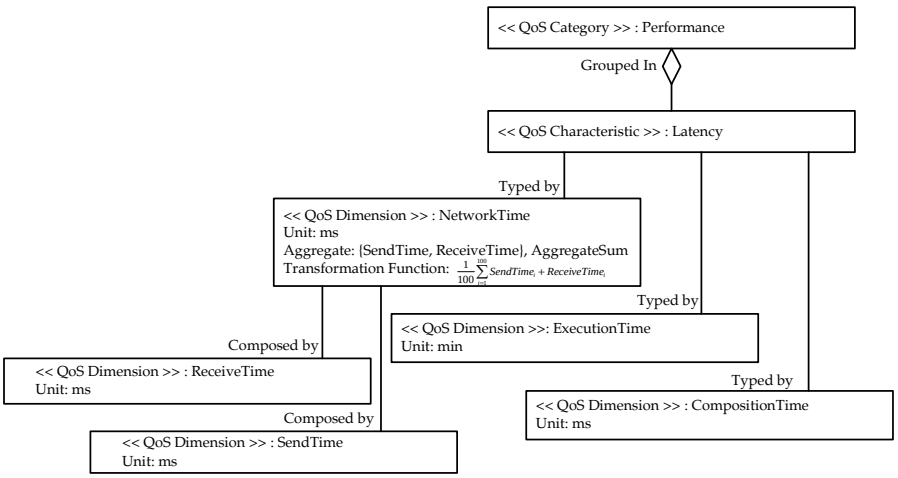
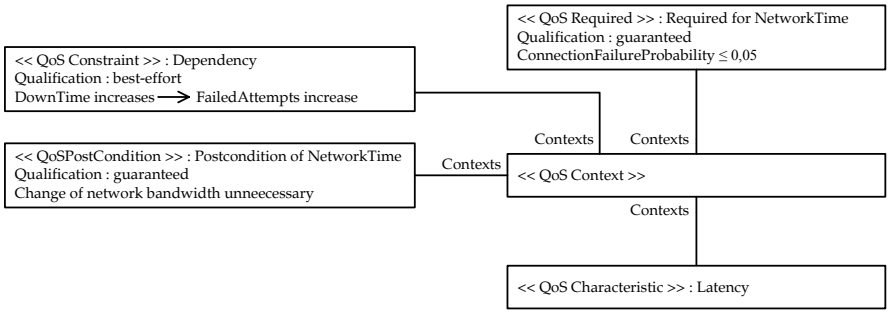


Figure 3.6: UML QoS Constraints submodel



The first submodel, the QoS Characteristic submodel, is illustrated in figure 3.5. It illustrates the characteristic latency and the dimensions used to compute its value. The calculation of one of these dimensions, Network Time is more developed with association of dimensions used to compose its own value and its transformation function. The Performance Category groups all characteristics related to the service performance. The modeling constructs initially introduced in the UML QoS Framework Metamodel do not allow the quantification of a QoS Dimension by aother QoS Dimensions. However, in the context of the ENVISAT/MERIS MGVI Regional service, the latency is a critical point. Specifying how its different parts are computed permits to the user or the provider to specify its preferences at different levels and so give an upper limit to the Receive Time, which is used to compose the Network Time.

The second submodel, the QoS Constraint submodel, is presented in figure 3.6. It exposes the context and its related constraints. Among those, one constraint illustrates the dependency existing between the down time and the total number of failed attempts. One is used to specify the precondition necessary to an acceptable value of network time and another the induced postcondition. The specification of such constraints, made by the service provider, in the context of our MERIS service, are used to inform the service user of particular requirements and observations about service behavior. The specification of the QoS Dependency related to the Down Time and the number of Failed Attempts is useful for the user of the service because it enables the anticipation of quality degradations.

The constraint submodel is also used to indicate the service provider capabilities of QoS. These capabilities are described with the help of QoS Offered, a specialization of QoS Constraint. The figure 3.7 highlights the capabilities of the MERIS MGVI Regional Service for the availability characteristic. To advertise its capabilities about quality properties, the service provider has to specify them. In the context or our case study, these specifications are simply the observations made on the system that are expressed

Figure 3.7: UML QoS Offered Constraints

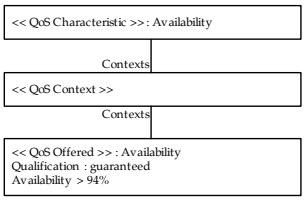
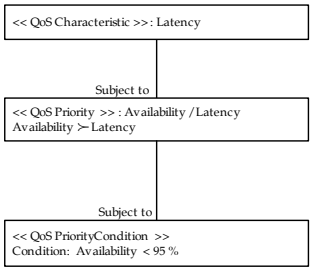


Figure 3.8: UML QoS Priorities submodel



with the help of the QoS Offered metaclass.

The QoS Priorities submodel is expressed in figure 3.8 and concerns a priority fixed between the latency and the availability. Services as the MERIS MGVI Regional have an important latency, mainly due to the huge execution time of their requests. This way, as the latency is important, it is not a relevant characteristic to choose among alternatives and other characteristics as availability are more relevant to discriminate among available services. Such a specification is written by the user of the ENVISAT/MERIS MGVI Regional service that wishes express which quality properties to favor to others.

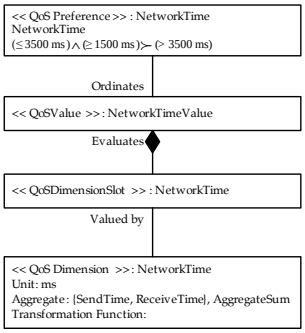
An example of expression abilities of the QoS Preferences submodel is proposed in figure 3.9. It illustrates how to specify favored values for the network time dimension. This specification of desired values for Network Time is written by the service user to make appear its expectations. The service user makes a similar specification for every QoS property.

The UML QoS Framework permits to service providers and users to express simply their advertisings and their requests. With our added extensions, it covers a large range of construct modeling and possibilities. Their complete description enables their powerful utilization in selection or composition approaches. Indeed, quality characteristics account for selection criteria in selection problems while they appear as local or global constraints in composition issues. UML specifications of quality requirements (or offers for providers) need to be translated in a language such that Q-WSDL in order to be sent and used by services selectors or composers. The model of QoS needs or offers should be established with the help of an appropriate tool allowing to construct the UML model and to translate this into an exchangeable language between interested parts. Any tool that uses our QoS model will allow users to define their expectations, requirements and advertisements. A tool that incorporates all of our proposed modeling constructs can easily be used to state quality properties of the service. Moreover, a such tool can also involve coherence rules to check the consistency of a model, with transitivity of priorities or propagation of dependencies at different levels of specification.

3.6 Discussion

We have observed at the outset of the chapter that quality has been variously defined. While quality is undoubtedly a polysemous concept, using it in software engineering requires agreement, even if local to a system or application domain, as to what is understood by quality management and quality assurance during the engineering, operation, and maintenance of software, and therefore service-oriented systems. Quality modeling is in this respect one of key activities. A quality model in this perspective has several

Figure 3.9: UML QoS Preferences submodel



purposes. Prominent among these are (i) to highlight the information to account for when representing and reasoning about quality, (ii) to indicate how quality is measured so as to assist in the assessment of a given software system, (iii) and to structure the information about users' and software engineers' quality requirements. The quality model proposed in this chapter has been designed with the said three purposes in mind. It focuses on service-oriented systems whose salient characteristic is dynamicity during operation, whereby dynamicity herein amounts primarily to the uncertainty about the levels of quality that the system achieves during operation. The uncertainty arises from the fact that the pool of available services varies over time. This may be desirable w.r.t. quality when new, more appropriate services appear in the pool of available services, but can also be undesirable when some relevant services become unavailable resulting in inconsistent quality levels.

3.6.1 Experience

Service orientation is intended to enable large scale systems. Many competing services are therefore available to perform the same tasks. In such a setting, the service composer aims to *select* the set of services that optimally satisfies the quality considerations laid out in the request, and this relative to alternative sets of services that can perform the same tasks. Our quality model has been applied primarily to the problem of web service selection. In an SOS, web service requesters specify tasks that need to be executed and the quality levels to meet, whereas service providers advertise their services' capabilities and the quality levels they can reach. Service selectors then match to the relevant tasks, the candidate services that can perform these tasks to the most desirable quality levels. One of the key problems in QoS-aware service selection lies in managing tradeoffs among QoS expectations at runtime, that is, situations in which service requesters specify quality levels that cannot be simultaneously met. We have used the quality model presented in this chapter within a wider service selection framework [74], in order to be able to deal with QoS tradeoffs. That framework consists of: (i) rich QoS models obtained by instantiating the model in the present chapter, used by service requesters when expressing QoS expectations and service providers when describing services' QoS; and (ii) a multi-criteria decision making technique that uses the models for service selection. The additional expressivity of the quality model presented in the present chapter proved a significant advantage in dealing with QoS tradeoffs. Indeed, the multi-criteria method that was used requires the identification of preferences over QoS values and priorities over QoS criteria, whereby a criterion equates with a quality dimension. The conceptual bases laid out in the present chapter proved immediately useful in constructing a web service selection framework that can deal with QoS tradeoffs.

The service selection framework [74] mentioned above uses a specific multi-criteria decision making method to rank alternative services. In our other work [75], we have also used our quality model to develop a generic addon to various service selection approaches so as to make these approaches aware

of service requesters' priorites over quality dimensions and/or characteristics. Instead of developing a particular service selection procedure which accommodates priorities between QoS considerations, we provided an extension compatible with (i.e., that can be used with) available selection approaches. Our approach is based on the premise that SOS operate in a setting in which QoS levels vary and are observed at runtime. The approach is based on a specific class of multi-criteria decision making techniques, called the outranking methods. The approach enables the definition of a global priority constraint to be used as an ordinary constraint in a service selection algorithm. We do not ask for a specific class of service selection algorithms; any algorithm which proceeds to select the optimal services and accounts for QoS considerations can be used in conjunction with the present proposal. This is for instance the case with the reinforcement learning algorithm we suggested elsewhere [100; 101]. The global priority constraint is relevant because: (i) it allows priorities to be accounted for during service selection in algorithms that originally cannot accommodate priorities; (ii) it can be integrated with various available service selection algorithms, and regardless of their specific optimization functions; (iii) it enables automatic optimization of user preferences by the service composer. It is our quality model that provides the conceptual foundations for the definition of the global priority constraint, and thereby the extension of service selection algorithms to accommodate richer specification of quality-related requirements.

This quality model has primarily been applied to service-oriented systems, and we presented it in that context in the present chapter. In this respect, we cannot advance claims on its applicability to other kinds of systems. We can however, identify arguments in support for its wider relevance. First, preferences and priorities expressed by the stakeholders (e.g., a system's users, owners, and so on) are not a kind of information specific to service-orientation. We have argued elsewhere [99] that preferences and priorities are expressed by the system's stakeholders and should be accounted over the course of the requirements engineering phase in the software development process. Requirements engineering deals with the elicitation, analysis, and specification of stakeholders' functional and quality requirements. Decision on the computing paradigm to adopt (e.g., service-orientation or agent-orientation, or a combination, or otherwise) arises in part from the requirements that the stakeholders express. It follows that our quality model may be used, at least as a starting point in the definition of quality models for kinds of systems other than service-oriented ones. Second, we have used our model to extend the UML QoS framework, which is not specific to service-oriented systems. Below, we consider the lessons learned from the use of our quality model in the context of service-oriented systems.

3.6.2 User Evaluation

The quality model and the service selection approaches that arose from and use the model have been used in cooperation with the European Space Agency, and the MERIS project. While this does not provide enough empirical data on to reach definite conclusions on the ease of use of the present quality model, some preliminary observations are available.

It was to be expected that a more expressive quality model requires more effort in use. This is the case with our quality model. The additional effort equated to: (i) additional training of the modeling participants that is necessary to understand the new modeling primitives and their use; (ii) effort involved in acquiring and analyzing the information to carry over to the instances of the modeling primitives; (iii) effort involved in using the instances of our quality model in decision-making during the engineering of a service-oriented system. The effort involved in (i) and (ii) are topics for research in requirements engineering, while (iii) is a concern for research in service selection. In practice, we have encountered difficulties in relation to both issues (i) and (ii). In a separate discussion [99], we show that information about preferences and priorities is not properly accounted for in ontologies for requirements engineering. The direct consequence of this is that there is limited research on the elicitation and analysis of preferences and priorities in requirements engineering, and thereby little systematic guidance on these tasks. In addressing issue (iii), we have used multi-criteria decision making techniques developed mainly in management science. Our experiences with the use of these techniques are reported elsewhere [74; 75]. Overall, multi-criteria decision making techniques allowed us to automate some of the tasks involved in using the instances of our quality model to rank services, while accounting for preferences and priorities

of the service requesters. In addition, these techniques incorporate features that allowed us to address issues (i) and (ii) to some extent. For example, and as explained elsewhere [74; 75], the methods we chose can cope with partial specifications of preferences and priorities, thus limiting the amount of information to elicit before instantiating our quality model. This is critical, for it reduces the efforts of both kinds (i) and (ii) mentioned above.

A separate and difficult issue concerns dealing with change of quality dimensions or characteristics, and/or preferences and priorities at runtime. It still remains unclear how precisely such changes affect even the engineering of functional requirements, which appear easier to pin down than quality requirements. Some changes can be managed. We have performed and presented elsewhere [102] our investigations on the impact changes in functional and quality requirements on the requirements engineering process for SOS. We showed therein that keeping track of changes is an issue that can only be properly dealt via appropriate tool support, although we did conclude that no easy to use solution is available at present.

3.6.3 Strengths

Our model's principal strength is its additional expressivity in comparison to related quality models for SOS. This additional expressivity raised relevant questions in relation to the problems of web service selection. Namely, we have shown elsewhere [74; 75] that the use of our quality model enables us to capture more detailed information from the service requesters – in particular, their preferences and priorities. This additional information proves invaluable when dealing with QoS tradeoffs during service selection. Namely, when QoS levels requested by the users cannot be simultaneously reached, preferences and priorities obtained by instantiating our quality model allow us to perform tradeoffs in accordance with the requirements of the service requesters.

3.6.4 Weaknesses

We have observed in practice, within the context of the MERIS project, that additional effort was required for the acquisition, analysis, and specification of preferences and priorities. We have presented elsewhere [74; 75] our first investigations on how this additional effort can be reduced within a broader framework for service selection in SOS. Our quality model does not feature primitives that are tailored to managing change in quality dimensions and characteristics, and/or preferences and priorities.

3.6.5 Future work

Work with more expressive models requires additional resources. Striking a balance between expressivity and efficiency is a difficult question, with answers undoubtedly confined to domain-specific experience. Experience in the use of the model points out that additional effort involved in specifying preferences and priorities is relevant for it provides guidance for service selection and composition in the face of uncertainty about which services are available at any moment. Automating the elicitation of preferences and priorities is a particularly relevant direction of future effort, especially as research and practice moves towards automated service selection and composition in which preferences and priorities can be revised and new elicited at runtime, then fed to automated service composers. Combining Brafman and colleagues' results [23] with the present quality model is of current interest.

Definition of quality dimensions and characteristics also requires guidance. Requirements engineering frameworks, such as, e.g., Tropos [36], can be used to obtain initial functional and quality goals of the service requesters. Metric definition needed for the writing of quality dimensions and quality characteristics has been treated for instance by Basili and Rombach [10] whose Goal-Question-Metric approach can be deployed to obtain quality dimensions and characteristics in a systematic manner. The use of our quality model within these frameworks is a topic of ongoing work.

We do not address in this chapter the issue of model compatibility, that is, the situation in which users and provider define the same quality property with different names or measurement transformation. One appropriate approach towards ensuring the compatibility of models amounts to annotate then with semantic information, then proceed to semantic matching – the reader may be interested in the approach

suggested by Zeng and colleagues [230], where service requests are translated by accounting for semantic information that is available. Such a process could be applied to quality specifications produced in our approach in order to avoid compatibility issues.

Another relevant consideration for future effort is the provision of more precise semantics for the various submodels and the subsequent more rigorous definition of the relationships between the proposed submodels. Achieving this aim requires formal semantics which can be obtained in two ways. One is to locate the model within a specific application domain, and therefore have domain-specific semantics. Another way is to attempt to provide a general formal semantics for the notion of quality dimension and build from there. In both cases the effort is currently only preliminary. Careful choices among alternative semantics and their potential extensions is needed, and must therefore be relegated to a separate future treatment.

3.7 Conclusions

Expressive quality models are needed to let requesters specify quality expectations, providers advertise service qualities, and composers finely compare alternative services. Having observed many similarities between various quality models proposed in the literature, we reviewed these and integrated them into a single quality model for service-oriented systems. We advance current research by integrating precise submodels for dependency and priority information closely with concepts and constructs already established as relevant when performing quality modeling.

The proposed quality model, the QVDP, integrates concepts and constructs for extensive representations of quality information. A simple, yet realistic example illustrated the relevance of instantiating the various proposed concepts and constructs when performing quality modeling. We highlighted the need for integration of dependency and priority information within any quality model for services. We pointed out how lack of dependency and priority information limits the value of various prior quality models. As a solution, we proposed precise submodels for representing dependencies and priorities between quality dimensions and between quality characteristics. The comparison of QVDP with prior quality models indicated that extending a particular quality model with dependency and priority submodels is possible. It also highlighted that an expressive and practical quality model ought to closely integrate concepts and constructs for all of the quality information already established as relevant in related research, with the concepts and constructs of the dependency and priority submodels. To achieve such close integration between the various relevant concepts and constructs, we proposed QVDP as an integrative model instead of extending an existing quality model with the dependency and priority submodels. Our intention is for the proposed QVDP model to serve as a reference point for further developments in quality models for service-oriented systems.

While we have used simple examples to illustrate the use of QVDP, parts of the model have been employed and tested elsewhere [100; 101]. There, we QVDP to specify quality expectations in service requests and defining criteria to guide the learning of optimal service compositions. Our proposed extension of the UML metamodel for QoS [157] further facilitates the use of novel concepts and constructs present within QVDP in actual applications.

Acknowledgments

We are grateful to Emmanuel Mathot of the European Space Agency, who provided precise information about the GPOD project and assisted our efforts in describing quality information of services related to the GPOD project. The first author acknowledges funding from the Belgian ICM/CIM Doctoral Fellowship Program.

Part II

QoS-driven Management of Services

Outline of Part II

The second part of this thesis presents contributions related to the management of web services driven by quality information. Services management is essential to ensure the efficiency of executions and to prevent failures while quality plays an important role in service monitoring. Among others quality enables to measure the performance of functionally equivalent web services. Moreover, service requesters do not have the same behavior in relation to quality, they have different non-functional requirements. To offer an user-driven approach of service management, it is essential to consider the quality expectations of users. The different models proposed in this Part relies on QoS to improve the services management. These models use the output provided by the service quality model presented in Chapter 3 to execute management activities. Indeed, once QoS expectations have been formulated by the service user and QoS capabilities expressed by the service provider, resulting specifications offer large possibilities as QoS-driven selection, QoS-driven composition or QoS-driven classification. Each of the three Chapters of this Part illustrates one management application. We detail these Chapters here:

Chapter 4 proposes a service selection method based on quality properties of a service. In service-oriented systems, service requests specify users' requirements at runtime, i.e.: tasks to execute and Quality of Service criteria to meet and to optimize. Automated service selectors receive service requests, then select the service capable of executing the required tasks at the expected QoS levels. Problems arise when a selector cannot simultaneously optimize QoS criteria while enabling also the definition of preferences over the values of QoS criteria, and priorities between QoS criteria. By drawing on multi-criteria decision making techniques, we suggest a service selection framework that uses the information specified with the QoS model in order to select the most appropriate service at runtime. This chapter has been published in the Service Computing Conference'08 proceedings [72]. Other similar selection approaches have been published in the Conference on Advanced Information Systems'08 proceedings [74] and in the International Conference on Autonomic Computing'08 proceedings [75].

Chapter 5 proposes a service composition method based on the QoS users' expectations and providers' capabilities. Satisfying a user's requirements often requires the composition of a set of distinct services. A key challenge in this respect is to compare competing services, then select those that will deliver the most desirable feasible levels of Quality of Service. This chapter proposes an approach to the computation of individual services' aggregate QoS ratings when multiple QoS criteria are given, and a reinforcement learning algorithm which uses these ratings in order to find the selection of services that maximizes the overall QoS level delivered to the users. The RL approach is capable of handling variations in the pool of available services by exploring selections other than those than historical data shows appropriate, and guarantees optimal exploitation for a given exploration level.

Chapter 6 proposes a user profiling method relying on the formulation of QoS users' expectations. A service provider's reputation is a function of the feedback, given after every past transaction by the users of the service. A reliable reputation score reflects the differences between the quality levels advertised by the provider, and those delivered in past transactions. Obtaining a reliable reputation score is difficult. Since users may have interest in strategically manipulating feedback in order to influence a provider's reputation, efforts have been invested in designing incentive mechanisms to motivate honest feedback. We argue in this Chapter that dishonesty is not the only cause of bias in feedbacks and ensuing reputation scores. Consequently, even if the feedback is honest, the resulting reputation score need not be reliable. To obtain a reliable reputation score from honest feedback, it is necessary to account for the bias arising

from each user's outlook (i.e., tendency to provide overly optimistic or pessimistic feedbacks), sensitivity to deception (tendency to react positively or negatively with weak or strong importance to differences between the quality level expected and delivered), and sensitivity to brand image (tendency to react positively or negatively with weak or strong importance to the image of a provider). These three form together the feedback profile specific to a user. We propose a probabilistic model able to compute the feedback profile of each user. This chapter has been submitted to Decision Support Systems.

Chapter 4

QoS based Service Selection

This Chapter has been published in Service Computing Conference proceedings [72]. This Chapter outlines one of the quality-driven management methods proposed in Part II. The Chapter 4 presents an answer to one fundamental service management issue: *which service to select when a requester is faced to multiple functionally equivalent web services?* We propose a quality-driven selection of web services as quality information enables to discriminate such services through their non-functional properties. The selection method proposed here relies on quality specifications made by requesters and providers with the quality model (QVDP) proposed in Chapter 3. This method also relies on advanced concepts such as priorities and preferences highlighted in the QVDP model.

4.1 Introduction

Engineering and managing the operation of increasingly complex information systems is a key challenge in computing. It is now widely acknowledged that degrees of automation needed in response cannot be achieved without coordinated, open, distributed, interoperable, and modular systems capable of dynamic adaptation to changing operating conditions. Among the various approaches to building such cooperative information systems, service-orientation stands out in terms of its reliance on the World Wide Web infrastructure, availability of standards for describing and enabling interaction between services, attention to interoperability, and uptake in industry.

Services are self-describing components that support rapid, low-cost composition of distributed applications [162]. Service providers advertise their services by way of service descriptions (which indicate a service's interfaces, capabilities, behaviors, and quality), and provide technical and business support. Different providers can advertise competing services. Competing services provide the same functionality, but at different levels of quality. Services are used by service requesters.

Quality of Service (QoS) is a combination of several characteristics of a service, such as availability, security, response time or throughput [143; 159]. Many providers may compete to offer the same services. Consequently, it is necessary to discriminate between competing services. To do so, services can be compared over QoS criteria, to which their respective providers can commit. This implies that quality requirements of services need to be accurately defined by service requesters if they are to be accounted for in the service selection process. Quality requirements are defined using QoS models; A QoS model includes a set of concepts and relationships found to be useful in the definition of quality requirements. It is by instantiating the QoS model that quality requirements are made explicit [41]. The QoS model is used in an service-oriented system (SoS) to make explicit the various QoS dimensions and characteristics that can be used to specify QoS considerations in service requests and measure them at runtime on each service. Any such model is therefore used (i) by service requesters to specify the expected quality levels of service delivery; (ii) by service providers to advertise quality levels that theirs services achieve; and (iii) by service selectors when selecting among alternative services those that are the most efficient. Given a QoS model, a service requester may specify that several QoS dimensions that need to be optimized.

The service selector then manages the selection process, and is in charge of assigning available services

to service requests based on the matching between the requested and offered QoS. To take into consideration multiple quality properties, Multi-Criteria Decision Making (MCDM) is usually used, whereby each QoS property is seen as a criterion [69; 134; 229]. If the said quality properties involve tradeoffs, they cannot be optimized and additional information is needed in a service request. Namely, we expect the stakeholders to indicate the priority over the said QoS dimensions so that an order of importance is established, and subsequently used to guide optimization. Preferences over values of quality properties are also useful to discriminate services as they reflect user expectations. Although there is a clear need for priorities and preferences, limited effort has been invested in dealing with these considerations during service selection.

Contributions. We propose a selection framework, which includes a QoS model and a selection mechanism to enable the assignment of the 'best' available service to each service request. The users' quality requirements are specified by instantiating our QoS model, in which preferences and priorities are accounted for. Once quality requirements are given, the selection mechanism proceeds along the following steps: (1) The selector rejects services that do not fulfill user expectations about values of QoS properties. (2) The selector organizes QoS properties into positive and negative hierarchies that make explicit the contribution to more generic quality properties (e.g., responsiveness). (3) The selector links weights to QoS properties that reflect their relative importance. (4) Pairwise comparisons of values of quality properties are made by the service selector. (5) The result of pairwise comparisons is combined to the weights of criteria to determine the score of positive and negative hierarchies. Cost/benefit analysis is then performed on each service to establish their ranking and determine the most suitable service to the user request.

Organization. Section 4.2 gives an overview of the method used by the service selector and introduces the case study used throughout this chapter. Section 4.3 discusses the details of our QoS model and illustrates its use through an example drawn from the case study. Section 4.4 details the steps of the selection algorithm and illustrates them via the case study. Section 4.5 justifies the different steps of our selection mechanisms. Section 4.6 discusses the related work and Section 4.7 concludes this chapter.

4.2 Preliminaries

4.2.1 Overview of the service selection approach

Our selection approach consists of two main steps: the first is the specification of quality requirements, along with the preferences and priorities thereon. The second step amounts to use these quality requirements in the service selection algorithm. User specifications and their transformation into concepts used by the service selector will be outlined in Section 4.2. The service selection algorithm of the service selector is presented here, in Algorithm 1 and will be detailed in Section 4.4.

Lines 2-6 of the Algorithm 1 refer to the limitation of accepted services to those that respect constraints on QoS values, this limitation is detailed in Subsection 4.4.1. Lines 7-19 of the Algorithm 1 concern the separation of quality requirements into two sets: those to maximize and those to minimize. Then, each set of QoS properties that undeline quality requirements is organized in a hierarchical tree, to highlight links between QoS considered. The benefits of such hierarchies is explained in Subsection 4.4.2. Lines 20-27 indicate how priorities are assigned to QoS characteristics and/or dimensions. In Subsection 4.4.3, we explain the Analytics Hierarchy Process (AHP), used to assign weights to QoS characteristics and/or dimentions. These weights reflect priorities. Lines 28-35 formalize the Promethee method applied to compare services on the basis of preferences. They also define how scores are given to all services on both hierarchy trees, more details are given in Subsection 4.4.4. Finally, lines 36-38 present the attribution of a the final score with the benefits/costs analysis. More details are given in Subsection 4.4.5.

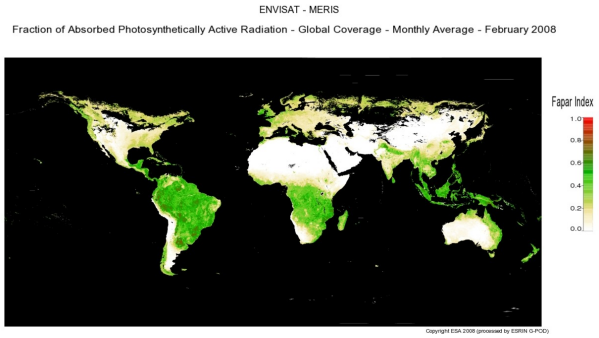
Algorithm 1 Selection Algorithm

i: the requested service transaction, l: the client of the transaction i, $s, s', s_1, s_2, ..., s_n$: available services satisfying functional requirements of the requested service, $k \in K$: a provider with K the set of providers offering the service s
$\mathtt{AC}_l$: Set of QoS Characteristic or QoS Dimension requested by l
for all s **do**
  for all $\mathtt{AC}_{lj}$ **do**
    if $\mathtt{AP}_{kij}\neg$ satisfies $\mathtt{AC}_{lj}$ **then**
      reject s
for all $\mathtt{AC}_{lj}$ **do**
  if $\mathtt{AC}_{lj}^{mod}$ = maximize **then**
    put $\mathtt{AC}_{lj}$ in positive criteria set set^+
  else if $\mathtt{AC}_{lj}^{mod}$ = minimize **then**
    put $\mathtt{AC}_{lj}$ in negative criteria set set^-
for both sets set^+ and set^- **do**
  for all $\mathtt{AC}_{lj}$ in set^+ or in set^- **do**
    $\mathtt{AC}_{lj}$ is a node
    for all $\mathtt{AC}_{lj_{sub-parent}}$ relationship **do**
      add an edge from $\mathtt{AC}_{lj}$ to its parent node
  add a super-node n to the top of the tree t
  for all top level node **do**
    add an edge to the node n
for both trees t; positive tree: $tree^+$ and negative tree: $tree^-$ **do**
  starting from the top node to the down levels
  for all level of t **do**
    for all set of sibling nodes **do**
      for all node n **do**
        retrieve rules $\mathtt{AC}_{lj}^{prior_{rules}}$ and strength $\mathtt{AC}_{lj}^{prior_{strength}}$ priorities specifications of $\mathtt{AC}_{lj}$
      make AHP pairwise comparison with $\mathtt{AC}_{lj}^{prior_{rules}}$ and $\mathtt{AC}_{lj}^{prior_{strength}}$ specifications to fix the relative weights of priorities $\mathtt{AC'}_{lj}^{prior}$ of sibling nodes
    final weights $\mathtt{AC}_{lj}^{prior} \leftarrow \mathtt{AC'}_{lj}^{prior} \times \mathtt{AC'}_{lj}^{prior}$ with $\mathtt{AC}_{lp}^{prior}$ the weight of the parent node p
for both $tree^+$ and $tree^-$ **do**
  for all couple s_1, s_2 **do**
    for all $\mathtt{AC}_{lj}$ **do**
      $F_q[d_q(s_1, s_2)] \leftarrow$ Promethee pairwise comparisons between s_1 and s_2 on $\mathtt{AC}_{lj}$ with the type F and parameters specified in $\mathtt{AC}_l$
    fix aggregated preference indexes: $\pi(s_1, s_2) \leftarrow \sum_{j=1}^{n} F_j[d_j(s_1, s_2)]\mathtt{AC}_{lp}^{prior}$ and $\pi(s_2, s_1) \leftarrow \sum_{j=1}^{n} F_j[d_j(s_2, s_1)]\mathtt{AC}_{lp}^{prior}$
  for all s **do**
    determine outranking flows of s: $\phi^+(s) \leftarrow \frac{1}{n-1}\sum_{s'} \pi(s, s')$and $\phi^-(s) \leftarrow \frac{1}{n-1}\sum_{s'} \pi(s', s)$ with $s' \neq s$
    determine the complete outranking flow of s: $\phi(s) \leftarrow \phi^+(s) - \phi^-(s)$
for all s **do**
  calculate the final score of s: $r_s \leftarrow \phi(s)_{pos}/\phi(s)_{neg}$
sort $s_1, s_2, ..., s_n$ with their final scores and determine the best one

4.2.2 Case study

In this section, we propose a case study subsequently used throughout the chapter. This case study is described in Subsection 3.2.2.

Figure 4.1: Vegetation indexes



We are interested in the remainder in services that provide the Fraction of Absorbed Photosynthetically Active Radiation (FAPAR) index (i.e.: the vegetation index) for the world map at a given period of time provided by the MERIS/MGVI service introduced in Subsection 3.2.2. A vegetation index measures the amount of vegetation on the Earth's surface. Figure 4.1 illustrates an example of the visualization of the vegetation indexes obtained for February 2008. The only required input of this service is the time range on which the mean vegetation indexes will be calculated. With the ESA program on Earth Observation and the access to data and computing resources given to researchers, multiple such services emanating from different providers or with different QoS levels are available.

4.3 Conceptual Foundations

We introduce in this section concepts used by the service user to specify their its requirements. We describe modeling constructs enabling the service user to specify its preferences and priorities about QoS. We suggest an illustrative example derived from our case study.

We suggest a QoS model that enables the user to express accurately its needs about quality properties of its required service. To account for various aspects of user expectations, this model must include advanced concepts such as priorities over quality characteristics or preferences on offered values.

Our model is based on the UML QoS Profile proposed by the OMG [157] and is shown in Figure 4.2. The original UML QoS Framework metamodel, introduced by the Object Management Group [157],

Figure 4.2: UML metaclasses to user modeling

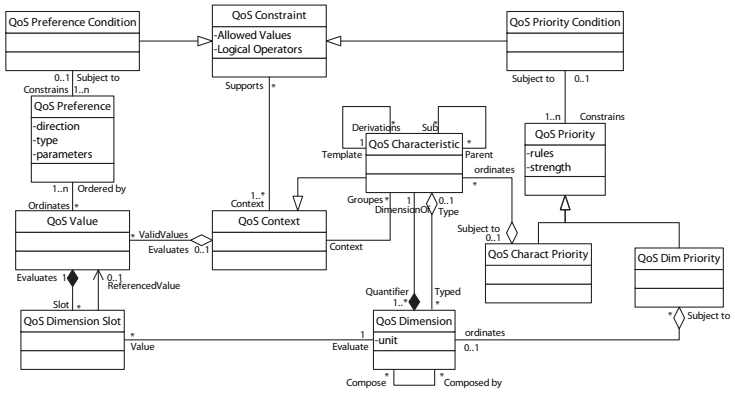
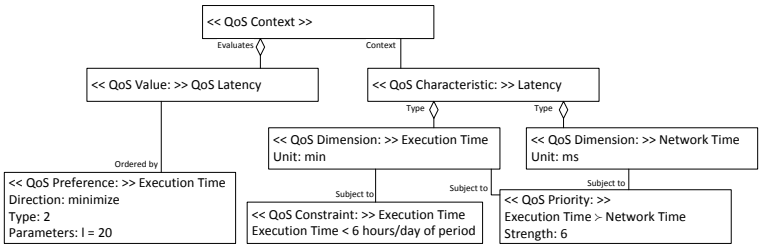


Figure 4.3: User specifications



includes modeling constructs for the description of QoS considerations. In that metamodel, a QoS Characteristic describes some abstract quality consideration, whereas a QoS Dimension describes measures that quantify QoS Characteristics. QoS Constraints restrict values of QoS Characteristics to those that are desired, and this across modeling elements identified during requirements engineering and architectural design. In comparison with the original OMG metamodel, we make some additional assumptions:

- In the OMG standard, *QoS Characteristics* are quantified by means of one or several *QoS Dimensions*. We assume that the value of a QoS Dimension can similarly be calculated with quantitative measures of other QoS Dimensions. This assumption is expressed in the metamodel in Figure 4.2 through the *Compose-Composed by* relationship of the *QoS Dimension* metaclass.

- We allow the user to express its priorities over QoS Characteristics and over QoS Dimensions by means of, respectively, *QoS Charact Priority* and *QoS Dim Priority* metaclasses whose are specializations of the *QoS Priority* metaclass. Its attribute *rules* concerns QoS Characteristics or QoS Dimensions involved in the priority and the direction of the priority while the attribute *strength* indicates the relative importance of the priority. *QoS Priority Condition* indicates conditions that need to hold in order for the priority to become applicable.

- To enable the user to express its preferences over values of QoS Characteristics and QoS Dimensions, we add a specific metaclass: *QoS Preference*. Preferences over values are defined with some attributes: *direction* states if the value has to be minimized or maximized; *type* indicates the type that the user favors for defining the preference and; *parameters* is used to define parameters needed for the type used.

Suppose a service requester that wishes retrieve the vegetation indexes for a given period. Multiple services are available to process such a vegetation map with different QoS properties. This user want to optimize the following QoS Characteristics: *availability, cost, feedback, latency, reliability, reputation* and *security*. Some of these quality considerations are not directly quantifiable, and are measured with help of QoS Dimensions. All these informations are specified by the service requester with the help of our proposed QoS model. The set of QoS characteristics and QoS dimensions expected by the useer l is defined in$\mathtt{AC}_l$. Parts of the complete specification of the user are illustrated in Figure 4.3. The user specification limits accepted services to those that have an execution time inferior to 6 hour by day of the selected period. This delay may appear important but the quantity of data to process is huge and require long time of duration. The execution time is favored to the network time in the priority specification indeed, the execution time is considered as the bottleneck of the execution process.

4.4 Selection Framework

The service selection process uses all specified information in combination with MCDM techniques in order to establish a complete ranking of available services. This process involves the following steps: (1) apply hard constraints on services, to restrict the set of services upon whose MCDM calculation will be made. This step is explained in Subsection 4.4.1. (2) establish the hierarchy of QoS properties with information related to QoS Characteristics and QoS Dimensions decomposition, each property being considered as a

criterion of the MCDM model. Moreover, two distinct hierarchies are built. The first focuses on criteria to maximize, called benefits. The second concerns criteria to minimize, called costs. Subsection 4.4.2 presents this step in details. (3) fix the priorities of QoS properties by applying the Analytic Hierarchy Process (AHP) on both hierarchies, as explained in Subsection 4.4.3. (4) make pairwise comparisons on each criterion of available alternatives. This step is done with the Promethee process, which gives us the opportunity to make pairwise comparisons with few information given on criteria. The utilization of the Promethee method is described in Subsection 4.4.4. (5) combine intra criterion information with weights fixed on criteria on both hierarchies. Then for each alternative, the ratio benefits/costs is computed by service selector and a complete ranking is made on available alternatives. The application of the benefits/costs ratio is detailed in Subsection 4.4.5. We give illustrative examples derived from our case study for all steps of the selection framework.

4.4.1 Fixing hard constraints

Hard constraints on quality properties (i.e., QoS Characteristics or QoS Dimensions) are defined by the user to restrict the set of accepted services. $\texttt{AC}_{lj}$ defines the quality level required by a client l for a quality property j. These are specified with the *QoS Constraint* metaclass and fix thresholds to values of a QoS Dimension. While the service selector choose the best available service to fulfill the user request, services that do not fulfill thresholds values for the different QoS Dimensions taken into account are considered irrelevant. The process is described in lines 2-6 in the Algorithm 1. $\texttt{AP}_{kij}$ specifies the quality level advertised for the quality property j by the provider k for the transaction i. Services for whose $\texttt{AP}_{kij}$ does not satisfy $\texttt{AC}_{lj}$ are rejected. Constraints allow us to decrease the number of alternative services to consider when applying MCDM - all services that do not satisfy the constraints are not considered for comparison. The complete specification made by requester about the service providing the vegetation indexes is transmitted to the service selector that will process all steps of the selection. The selector starts by rejecting services that do not fulfill hard constraints. For example, in specification given in Figure 4.3, the selector restrains available services to those that have an execution time inferior to 6 hours by day of the selected period.

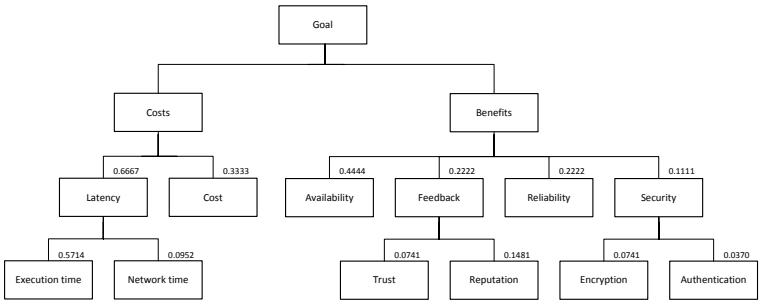
4.4.2 Hierarchies of QoS Characteristics and QoS Dimensions

Decomposition of QoS Characteristics into QoS Dimensions and QoS Dimensions into others QoS Dimensions may be used by the service selector to build a complete hierarchy of QoS properties. This information is expressed with help of the relations *Type - Typed* between the QoS Characteristic and the QoS Dimension metaclasses and *Compose - Composed by* defined over the QoS Dimension metaclass. The hierarchy established by the service selector allows us to link weights to QoS properties at different levels. This way, their relative importance is aggregated in accordance with the QoS properties that these quantify. To account for measurement of QoS Characteristics by QoS Dimensions and quantification of QoS Dimensions, we classify them into two separate hierarchies. The first is dedicated to benefits, that is, all quality properties that have to be maximized: availability, reliability, reputation, etc. The second is designed for costs, involving quality properties to minimize: execution time, failures, cost, etc. Modality (maximize or minimize) of QoS properties are defined with the attribute direction of the QoS Value class. The modality of the quality property j is then expressed in $\texttt{AC}_{lj}^{mod}$These two hierarchies are linked to the same global optimization goal. This top-down organization clearly indicates the contributions of lower levels of quality properties to upper ones. The final hierarchy takes the form of a tree. The process used to get this tree is detailed in lines 7-19 of the Algorithm 1. The second step of the selector is to establish benefits and costs hierarchies with the information provided by the service requester. The hierarchy corresponding to user requirements about the vegetation indexes service is illustrated in Figure 4.4.

4.4.3 Priorities as between criteria weights

Information about priorities is used to link weights to QoS Characteristics and QoS Dimensions, in order to express their respective relative importance. These weights are defined using QoS Priorities

Figure 4.4: Benefits and costs hierarchies



specifications given by the service user and are linked to the corresponding QoS properties. Once the hierarchy is established, the relative importance of each QoS property has to be fixed with a weight reflecting its contribution to the main optimization goal. These weights must be fixed independently for benefits criteria and for costs criteria in order to consider separately positive and negative QoS properties. To fix weights on such hierarchies, we use the Analytic Hierarchy Process (AHP) [177]. The Analytic Hierarchy Process fixes weights to criteria with the help of comparison matrices provided for each level of criteria. For a same level, each criterion is compared with other criteria of its level on a scale fixed between 1/9 and 9. Each matrix is build with QoS Priority specifications: rules express direction of pairwise comparisons of criteria and strength fixes the value chosen by the user on the scale for the comparison. Rules of priorities between quality properties of the client l are expressed in $\mathtt{AC}_{lj}^{prior_{rules}}$ and strength of priorities between quality properties of the client l are expressed in $\mathtt{AC}_{lj}^{prior_{strength}}$. Next, weights of QoS properties are obtained with the computation of the right eigenvector of the matrix. The eigenvector is computed by raising the pairwise matrix to powers that are successively squared each time. The rows sums are then calculated and normalized. The computation is stopped when the difference between these sums in two consecutive calculations is smaller than a prescribed value. The service selector adopts a top-down approach, the weights of each level being multiplied by the weight of the quality property of its upper level to determine its relative importance on the whole hierarchy. This process is performed on both sides of the tree, for positive and negative quality properties. Priorities processing relates to lines 20-27 of the Algorithm 1. The third step of service selection is to fix weights for each level of criteria with the AHP method. With the information provided by QoS Priority instance in Figure 4.3, the service selector is able to build a comparison matrix for dimensions quantifying the latency (i.e.: execution time and network time). This matrix is $\begin{pmatrix} 1 & 6 \\ 1/6 & 1 \end{pmatrix}$. This step refers to the line 26 of the Algorithm 1. The selector computes its eigenvector to obtain weights for these sibling nodes: 0.14286 for *Network Time* and 0.85714 for *Execution Time*. These weights are multiplied by weights of upper levels to determine weights of the whole hierarchy as explained by line 27 of the Algorithm 1. The final weights for *Network Time* and *Execution Time* are respectively 0.0952 and 0.5714. Final weights of the qualityproperty j for the client l is defined in $\mathtt{AC}_{lj}^{prior}$. Final weights for the whole hierarchy are illustrated in Figure 4.4.

4.4.4 Preferences as intra criterion comparison

Preferences information specified by the user on QoS Values is used by the service selector to compute the intra criterion comparison structure. We use this information to determine what values are preferred for a given QoS Characteristic or QoS Dimension. The priorities of quality properties have been fixed with weights reflecting their relative importance in Subsection 4.4.3. Preferences on values allow us to discriminate services on a given criterion. To quantify these preferences, we rely on a specific class of MCDM methods: outranking methods [55] and more specifically the Promethee method [25]. The Promethee method performs pairwise comparisons of alternatives by considering the deviation between the evaluations on multiple criteria. The more significant the deviation, the higher the preference. We interpret the higher preference as higher priority herein.

Table 4.1: Available services that meet QoS constraints

Service	Exec. time	Net. time	Cost	Av.	Trust	Reput.	Rel.	Encrypt.	Authent.
1	279	24	16	99	61	86	88	9	8
2	344	11	19	77	90	90	94	6	9
3	331	27	10	88	78	97	98	9	9
4	303	19	18	85	97	85	92	8	9
5	331	13	3	97	79	87	75	9	8
6	306	23	20	93	77	88	82	8	9
7	255	18	5	84	94	88	98	9	9
8	246	15	5	71	81	94	98	8	10
9	255	23	18	95	68	89	82	7	10
10	285	21	15	83	87	88	97	7	99

The result of the pairwise comparison for a criterion to maximize is given by:

$$P_j(a,b) = F_j[d_j(a,b)] \forall a, b \in A \tag{4.1}$$

where

$$d_j(a,b) = g_j(a) - g_j(b) \tag{4.2}$$

and for which

$$0 \leq P_j(a,b) \leq 1 \tag{4.3}$$

Where:

- $P_j(a,b)$ is the preference of the observed quality of some service (choice a) over the observed quality of another service (choice b) over the QoS property j;
- $g_j(a)$ is the score of the service a over the dimension j;
- $d_j(a,b)$ is the deviation between the choice a and choice b over the QoS property j;
- F_j is the function giving the intra criterion information associated to the QoS property j.

Depending on the inherent characteristics of a given QoS Characteristic or QoS Dimension, the selector must apply one of six types of functions for intra criterion deviation specified with the QoS Preference class with the attribute type. These types are outlined in details in [25; 55], each type necessitates some particular parameters also specified by the user with the QoS Preference class: type 1 is referred to as immediate preference; type 2 introduces an indifference threshold; type 3 increases continuously up to this indifference threshold; type 4 comprises an indifference and a preference thresholds; type 5 increases continuously between indifference and preference thresholds and; type 6 follows a Gaussian law with a fixed standard deviation.

We give in Table 4.1 the performance of available services able to perform the vegetation indexes based on MERIS data. With these data, we can illustrate the line 31 of the Algorithm 1.

$$d_{exec.\ time}(2,3) = 13$$

$$d_{exec.\ time}(2,4) = 41$$

$$F_{exec.\ time}[d_{exec.\ time}(2,3)] = 0$$

$$F_{exec.\ time}[d_{exec.\ time}(2,4)] = 1$$

The execution time is defined in Figure 4.3 with a preference of the second type with an indifference threshold of 20 min. The preference of the service 2 over the service 3 for the execution time is 0 because the time difference is only 13 min. However, the preference of the service 2 over the service 4 is 1 because the execution time is 31 min which is longer than 20 min.

In order to establish the respective importance of alternatives, we need to define aggregated preference indexes and outranking flows.

The *aggregated preference indexes* are used to express the degree to which the service a preferred to the service b over all considered QoS Characteristics and Dimensions (written $\pi(a, b)$) and inversely, to what degree is the choice b preferred to the choice a over all considered QoS Characteristics and Dimensions ($\pi(b, a)$). Most of time, a will be of higher priority to b for some QoS properties and b will be of higher priority to a for others. Therefore, $\pi(a, b)$ and $\pi(b, a)$ are usually positive.

$\pi(a, b)$ and $\pi(b, a)$ are defined by:

$$\begin{cases} \pi(a,b) = \sum_{j=1}^{k} P_j(a,b) w_j \\ \pi(b,a) = \sum_{j=1}^{k} P_j(b,a) w_j \end{cases} \tag{4.4}$$

where w_j is the weight associated to the QoS Characteristic or QoS Dimension j derived with the AHP method and k is the number of distinct QoS properties.

For services providing the vegetation indexes, for the benefits hierarchy, the aggregated preference indexes over services 2 and 3 are the following: $\pi(2, 3) = 0,074074$ and $\pi(3, 2) = 0,888889$ meaning that the service 2 is less preferred to the service 3 than the service 3 is preferred to the service 2. These values have been obtained by multiplying values obtained from comparisons by weights derived from AHP as explained in Subsection 4.4.3. This step refers to the line 32 of the Algorithm 1.

The *outranking flows* determine how each choice a is facing the $n - 1$ other possible choices in A, the set of all possible alternatives. The positive outranking flow ($\phi^+(a)$) expresses how an alternative a is outranking all the others, the higher its value, the better the alternative. The negative outranking ($\phi^-(a)$) expresses how an alternative a is outranked by $n - 1$ other alternatives. The lower its value is, the better is the alternative. Outranking flows are evaluated for QoS properties on each side of the tree, respectively for benefits and costs hierarchies.

$\phi^+(a)$ and $\phi^-(a)$ are defined by:

$$\begin{cases} \phi^+(a) = \frac{1}{n-1} \sum_{x \in A} \pi(a, x) \\ \phi^-(a) = \frac{1}{n-1} \sum_{x \in A} \pi(x, a) \end{cases} \tag{4.5}$$

This step is presented in line 34 of the Algorithm 1. Values calculated for second service delivering vegetation indexes for the benefits hierarchy are respectively: $\phi^+(2) = 0,25308$ and $\phi^-(2) = 0,58162$. These values mean that, for the benefits QoS, other services are comparatively better than the service 2.

Once these outranking flows have been determined, several ways of ranking are available. PROMETHEE I proposes a partial ranking of alternatives authorizing equalities over alternatives while PROMETHEE II provides a complete ranking of alternatives. In our framework proposal, complete ranking offers more information than partial one, so we choose to use PROMETHEE II. The complete outranking flow of PROMETHEE II is defined by:

$$\phi(a) = \phi^+(a) - \phi^-(a) \tag{4.6}$$

The complete outranking flow refers to the line 35 of the Algorithm 1. For the benefits hierarchy, the complete outranking flow of the second service performing the vegetation indexes is: $\phi(2) = -0,32854$.

4.4.5 Benefits/costs analysis

Outranking flows on both hierarchies define the relative performance of services on positive properties (benefits) and negative properties (costs). Benefits should be maximized while costs have to be minimized. In order to aggregate both considerations into a single measure of performance, the AHP MCDM method proposes to compute the benefits/costs ratio [55]. For each alternative, the benefits/costs ratio is calculated and this score is linked to each available service as a relative measure of its performance. This process refers to lines 36-38 of the Algorithm 1. In the last step, the selector computes the outranking flows for both hierarchies. The selector then computes the benefits/costs ratio of each alternative as suggested by some AHP variations. The best service is the one with the highest final score.

Table 4.2: Final score of available services

service	final score
1	-3.99
2	-0.52
3	0.77
4	-0.51
5	1.55
6	-0.04
7	-0.14
8	0.16
9	-0.09
10	1.42

Table 4.2 presents the score of the relative performance of each service able to perform the vegetation indexes with the data provided by the MERIS instrument. These scores have been computed with the benefits/costs ratio explained in line 37 of the Algorithm 1. The service selector determines that the service whose fits the best the user requirements is the service 5. This service has the highest performance score has defined in the line 38 of the Algorithm 1.

4.5 Discussion of our Framework

4.5.1 Between criteria weighting

To fix weights to each quality properties, we use the classical AHP procedure proposed by Saaty [177]. However, numerous other methods have more recently appeared. They differ by the information that the user needs to provide, the calculation process or more specific properties. We overview here some methods that depend neither on the range of the scale nor the encoding to express the evaluation on this scale. Simos introduces a method in [194] that is reviewed later by Figueira in [56] which relies on an order relation over the considered criteria. The strength of the difference between two successive criteria is introduced by inserting 'blank' criteria. Bana E Costa introduces MACBETH in [8], a procedure that proposes a simple questioning procedure to drive the interactive quantification of values through pairwise verbal judgments of difference of attractiveness between valuable elements. Saaty's method is more appropriate to QoS priorities because the AHP method is restricted to application fields that have 'zero' as a natural limit, MACBETH authorizes concepts with a contrary opposite, the repulsiveness. The AHP solves the integration of repulsiveness more intuitively, by introducing benefits and costs hierarchies. Attractive quality properties are parts of the benefits hierarchy while repulsive quality properties are parts of the costs hierarchy. Moreover, AHP associates one real number with each QoS property while MACBETH associates an interval of $\mathbb{R}_0^+$ with each quality property that is not fixed a priori. In the field of QoS priorities, this interval is not easily interpreted and the association of one real number to each property is favored. Mousseau [145] introduces an elicitation technique for importance parameters that defines an interval on which priorities will be fixed, this method observes within criterion information to fix weights. Goldstein [63], Hokkanen [77] and Roy [176] also introduce similar methods to fix priorities. All these methods provide weights that can be used in our selection framework but the AHP method has an advantage over these methods. With AHP, the user expresses relative importance of criteria with pairwise comparisons on these criteria. This technique allows to express preference that are not necessarily transitive or even consistent [178].

4.5.2 Intra criterion comparison

To determine preferences related to Intra criterion information, several methods exist. The AHP method [177] can be used to express this information but relies on pairwise comparisons and involves $cn\frac{n-1}{2}$ comparisons provided by the service user with c the total number of quality properties and n the total number

of alternatives. If we use this technique to determine the performance of alternatives in the context of service selection, the service user has to specify an huge quantity of information.

Ratings are favored to pairwise comparisons because adding new alternatives has no effect on the rank of existing alternatives. To fix such ratings, several methods are available: Simple Additive Weighting (SAW) introduced by Hwang in [81], Simple Multi Attribute Rating Technique (SMART) proposed by Edwards in [215], outranking methods [55] or Multi-Attribute Utility Theory (MAUT) introduced by Keeney [108]. Outranking methods are of particular relevance when small differences of evaluations are not significant in terms of preference [55]. In the context of web services quality, such preferences may appear, for example: indifference threshold for the Network Time when the difference is lower than 5 ms. Different outranking methods are available: ELECTRE I, ELECTRE Is, ELECTRE III, Promethee and etc. In our selection approach, we choose to use Promethee [25]. The particularity of Promethee is its structure that is based on pairwise comparisons as the AHP method. However, these comparisons are autonomously computed and only require that the decision maker specifies the type of preference and its optional parameters. Pairwise comparisons are then easily transformed into ratings.

4.5.3 Hierarchy

The hierarchy organization of the AHP breaks down a problem into its smallest parts and then calls for only simple pairwise comparison judgments to develop priorities in each hierarchy [178]. Other MCDM techniques as the SAW [81], the SMART [215] or MACBETH [8] only permit a single level of criteria. Hierarchies utilization in the context of web services selection allows to consider both QoS Characteristics decomposition into QoS Dimensions and division of QoS Dimensions into other QoS Dimensions. The decomposition of quality properties calls for structuring the hierarchy to capture the basic elements of the problem. User priorities are then defined for concepts at different levels and really correspond to its expectations. Indeed, when comparing elements at each level, a decision maker has just to compare them with respect to the contribution of the lower-level elements to their parent element on the tree.

4.5.4 Benefits/costs analysis

The basic concept of benefits/costs analysis comes from economics. It is used in investigating alternative courses of action and to quantify all positive and negative aspects into a common currency such as dollars. Their ratio or difference can be calculated to determine whether the benefits outweigh the costs. When only one project among several options is to be taken, then the project with the highest benefits/costs ratio would be the most appropriate choice. When using AHP with benefits/costs analysis, the same approach is taken except that AHP priorities rather than dollars are used as the common currency of comparison [209]. Costs include economic costs and various intangibles, a benefits/costs analysis is used when the two are so close in value that both can be considered [55]. The benefits/costs ratio is not the only proposal to account for both benefits and costs hierarchies, Wedley et al. [214] overview different possibilities. However, in our selection approach, the benefits/costs ratio is the most easily interpretable, reflecting the overall quality in respect to the price.

4.6 Related Work

In [118], Lai et al. give a weight to the modules of software by utilizing the AHP. The priorities are based on the access frequencies of the modules. Jung and Choi report in [96] the results of a case study where the AHP benefits/costs analysis was employed to support the selection of a multi-media system in a group decision environment. However, none of these approaches refer to web services and their quality properties.

Shaikh and Mehandjiev [191] propose a negotiation of attributes in e-business process compositions involving contracts between the client organization and the organizations which are offering such process components. Bids are evaluated and selected for the whole composition with the help of the AHP and involve many attributes of importance to the negotiators, for example: price, quality and flexibility. AHP

is used in this context because of its ability to deal with the mix of quantitative and qualitative attributes and its general compatibility with the synthesis of decision making. AHP is used through a hierarchy of attributes to fix their overall weights in the composition process.

Tong and Zhang [203] present a fuzzy MCDM algorithm for web services selection based on QoS. This approach selects the service with the highest degree of membership belonging to the positive ideal solution. Negative and positive criteria are considered together and then need to be scaled. Performances are measured with weighted Euclidean distance to the positive and negative ideal solution.

Liu et al. propose in [124] a QoS computation for dynamic web service selection. Quality properties vectors are put in a matrix and are then normalized. The normalization allows for an uniform measurement of services qualities independent of units and provide an uniform index to represent service qualities for each provider. This normalization allows the authors to consider both positive and negative criteria. The approach also defines how to compute Intra criterion information for each quality property.

Naumann [153] also uses a MCDM technique to select the source of information based on quality properties. To rank possible execution plans, the SAW method [81] is applied. The user establishes a weighting of between criteria information and within criterion performance is given by the SAW scaling. This scaling is different for positive and negative criteria.

Our selection framework has several advantages over the approaches mentioned above. The model allow the users to easily express their expectations about priorities and preferences that need to be accounted for during the selection process. We also organize attributes in both separate hierarchies dedicated to benefits and costs to make appear their respective contributions to the optimization goal. We define clearly how to compute weights of quality properties and derive the quality of each service on each quality attribute with the Promethee method. The quality of each service is this way quickly measured and does not require that the user specifies significant quantities of information.

The Appendix A provides more information about service selection methods.

4.7 Conclusions

We propose a framework for service selection in this chapter. The first part of the framework is an extension of the UML QoS Framework, enabling the service requester/user to express priorities beytween QoS criteria and preferences over the values of the QoS criteria. The second part of the framework is a procedure that uses these rich specifications of QoS requirements for service selection. The constraints defined by the user on QoS criteria are used to reject available services that do not fulfill some requirements about their values. The preferences about values stated by the user are used to organize QoS criteria into positive and negative sets. The priorities specifications account in the weighting of the different criteria, realized with the AHP method. The pairwise comparisons of values of quality criteria are based on the preference type defined in the preference specification. Finally, a benefits-costs analysis is performed on each service alternative to determine the service that best fits the user's requirements.

Advantages of our approach are the matching between specifications made by the service user and information required by the service selector. As all the necessary elements are determined by the service requester, the service selector's task can be automated in a straightforward way along the lines of Algorithm 1. All steps of this algorithm are supported by different multi-criteria methods enabling to consider the user's preferences and priorities about QoS properties during service selection.

Chapter 5

QoS based Service Composition

This Chapter tackles an essential issue of services management. It proposes a quality driven composition method enabling to determine the best service composition available according to quality user expectations. A service composition assembles services to fulfill complex tasks involving multiple operations. The composition method is the extension of the selection problem presented in Chapter 4 to the simultaneous choice of multiple services providing different functionalities. As for the selection method proposed in Chapter 4, the composition method discriminates possible compositions upon basis of quality information. The presented method relies on quality information specified with the quality model defined in Chapter 3.

5.1 Introduction

Problem. A key benefit of service-oriented computing is the modularity of the services [162]. Interoperable services can be combined into service compositions, so that they can satisfy together those expectations of the users, which the services cannot individually satisfy. One important question in relation to service composition is how do we know which services should participate in a composition? In other words, given a pool of services, which can contain competing services that offer same functionality at different quality-of-service (QoS) levels, which of these services should be selected so that user's quality requirements are satisfied to the most desirable and feasible extent? The answer lies in the procedure applied to identify the appropriate selection of services, which will subsequently be composed. An associate difficult issue of interest is how to revise selections as new services become available, and those used in previously identified selections become unavailable. We shall refer to this second issue as the exploration-exploitation tradeoff: exploitation consists of relying on observed historical performance of services, while exploration consists replacing services with new ones.

Contributions. This chapter focuses on the problem of selecting services when there are potentially many QoS criteria describing the services, and the pool of services varies. In response, we propose a two-step service selection approach. (1) A method is applied to aggregate QoS level over many criteria so as to obtain a single rating of a service. (2) The computed ratings of the candidate services are input to a reinforcement learning (RL) algorithm, which finds the selection of services that maximizes the overall QoS level delivered to the user. The RL approach is capable of handling variations in the pool of available services by exploring selections other than those that historical data shows appropriate, i.e., the algorithm handles the exploration-exploitation tradeoff. Interestingly enough, the introduced RL algorithm guarantees (asymptotically) optimal exploitation for a given exploration level. Our first experiments reported here illustrate that, for a given level of exploration, our algorithm is more efficient than comparable approaches. The contributions of this chapter are the QoS aggregation and the RL algorithm, so that no commitments are made on, e.g., how composition proceeds once a selection is identified, how interoperability is ensured, and so on. This ensures that our results are generic. Said choices are left to the designer.

Organization. We start by presenting the overall abstract model of service selection, in which we define

our approach (§5.2). We then explain how to compute individual services' aggregate QoS ratings when multiple QoS criteria are defined (§5.3). The RL selection approach is then proposed (§5.4). Comments on the results of first experiments are then given (§5.5). We close the chapter with the discussion of related work (§5.6) and a summary of conclusions(§5.7).

5.2 Service Selection Model

The purpose of this section is to introduce the formalism and assumptions that we rely on in the remaining of the chapter to represent the pool of services together with the process, for which the services need to be selected. The satisfaction of a user's requirement typically requires the coordinated execution of several tasks, that is, the execution of a process. An example of process involving the execution of several tasks could be a travel planner aggregating multiple component services for flight booking, travel insurance, accommodation booking, car rental, and itinerary planning [229]. Formalisms such as, e.g., petri-nets [49] or statecharts [70] are common representations of the process that must be executed to satisfy a requirement. Given a process description and a pool of services, the service composition problem aims at determining which of the services from the pool should execute which tasks. Notice that we cannot apply our approach directly to a statechart or a petri-net. Indeed, in this work, we depict a process via a directed acyclic hypergraph (DAH).

The translation of a statechart into a DAH must keep the expressiveness of the composition process. The translations rules that we apply enable us to ensure the expressiveness initially offered by statecharts while petri-nets involve more elaborated translation rules. Indeed, statecharts allow the definition of complex processes involving loops, alternatives and concurrency. The DAH offers such advanced concepts with the help of adaptations described in proposed translation rules. The translation rules are inspired from Achbany et al. [1], see this work for more details:

1. The composition process is depicted in a statechart with sequence of states $[st_1, st_2, ..., st_m]$, such that st_1 is the initial state, st_m the final state, and for every state $st_i : (1 < i < m)$.

2. To allow concurrency, sets of concurrent transitions with a common origin state in a statechart must be labeled with an AND label.

3. If the statechart contains cycles, it must be unfolded into an acyclic statechart. Various techniques allow to unfold cyclic processes into acyclic ones [112; 160]. This rule enables to handle loops in the statechart.

4. DAH has a node for each state of the statechart.

5. DAH has an edge for each transition of the statechart. The origin node of the transition is the node corresponding to the origin state of the transition while the destination node is the node corresponding to the destination state of the transition.

6. Whenever there is an AND label on a node, the edge between the relevant nodes in the DAH are labeled with the tuple of states that must be executed concurrently, e.g., $[st_1, ..., st_n]$. This rule enables us to consider the concurrency expressed in statecharts [1].

Thus, a node in our DAH represents a state of the statechart. A link between two nodes depicts the execution of the task that drives the process between the given states, whereby the execution is performed by a single specific service from the pool. If there are m links from one node to the other, this means that there are m different services that can execute the same task: these services are therefore competing, and will need to be compared in terms of QoS properties (e.g., response time, availability, and so on).

Once translation rules have been applied, we can use the DAH to run the composition algorithm. The set of tasks that can be executed to complete the process is defined by $< t_1, t_2, ..., t_n >$ where n is the total number of tasks in the process. E.g., t_1 is the task defining the flight booking and t_2 is the

[1] The RL algorithm proposed in Section 5.4 must be extended to allow concurrency with the support of edge labels.

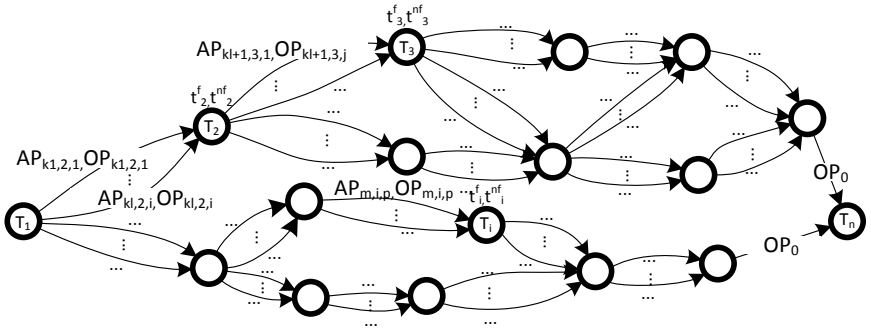


Figure 5.1: Directed Acyclic Hypergraph representation of the service composition

task defining the car rental. While the process is executed, all tasks do not need to be executed, because multiple combinations of tasks are available to fulfill the composition. Each task t_i is associated to a set of functional constraints ($t_i^f :< t_i^{f1}, t_i^{f2}, ..., t_i^{fn} >$ where n is the number of functional constraints of the task t_i) and a set non-functional constraints ($t_i^{nf} :< t_i^{nf1}, t_i^{nf2}, ..., t_i^{nfm} >$ where m is the number of non functional constraints of the task t_i) about the functionality to provide. E.g., t_1^{f1} states that the task has to provide the number of available seats and t_1^{nf1} states that the result has to be executed within 8 seconds.

A given task t_i of the process can be realized by different services. $s_{i_1}, s_{i_2}, ..., s_{i_n}$ are n services that can execute the task t_i. Each of these competing services can meet the functional requirements t_i^f of the task, and meet the threshold nonfunctional requirements, i.e., QoS criteria, t_i^{nf}. These services may be offered by different providers, or the same provider may offer same services at different QoS levels. Although all these services meet the threshold QoS levels for a given task, it is likely that some meet values that are more desirable than the threshold values defined in t_i^{nf}. E.g., the service s_{1_1} is able to execute the task within 5 seconds. We reasonably assume that the QoS surplus of which a service is capable is different among services. The user will then be interested in maximizing this surplus.

The user's QoS surplus is computed according to her QoS requirements, which are given by her priorities over QoS properties (e.g., response time is more important than availability) and her preferences over values of these properties (e.g., low response time is preferred to slower response time). The set of all QoS properties is defined by a QoS ontology. The user l – a human or another system – supplies the nonfunctional requirements by giving values for all QoS properties, that is, supplies the tuple $<\mathtt{AC}_{l1},\mathtt{AC}_{l2},...,\mathtt{AC}_{ln}>$ where n is the total number of quality properties. This tuple is discussed in detail below (§5.3.1).

QoS properties of a service are given in two distinct sets: the QoS levels advertised by the providers, and the QoS levels actually observed in past executions. The set of QoS levels advertised by the provider k of the service α fulfilling the task i is given by $<\mathtt{AP}_{ki_\alpha 1},\mathtt{AP}_{ki_\alpha 2},...,\mathtt{AP}_{ki_\alpha n}>$ where n is the number of QoS properties. E.g., $\mathtt{AP}_{11_1 1}$ states that the service s_{1_1} advertises an delay of 4 seconds for the execution time quality property. The set of QoS levels previously observed when the service α was fulfilling the task i is given by tuple $<\mathtt{OP}_{ki_\alpha 1},\mathtt{OP}_{ki_\alpha 2},...,\mathtt{OP}_{ki_\alpha n}>$, where n is again the number of QoS properties. E.g., $\mathtt{OP}_{11_1 1}$ refers to an observed execution time of 5 seconds.

Figure 5.1 illustrates the representation of a process in the described framework. Each task is associated to its functional (t^f) and nonfunctional requirements (t^{nf}). Each service edge is associated to its advertised (AP) and observed (OP) QoS levels. There are links between two nodes only if the predecessor task and the successor task belong to successive tasks of the process.

Figure 5.1 illustrates a process involving several tasks. t_1 is a fictional task standing for the beginning of the process and t_n is a fictional task standing for the end of the composition. Edges at destination to t_n have no QoS capabilities advertised and all are associated to the same neutral value ($\mathtt{OP}_0$). Remark finally that the DAH is not frozen over time. As the pool of services varies, the hypergraph obtains new

links, while others may be removed. New services can enter on the composition, old ones can disappear and, their QoS properties can change, since, e.g., the nonfunctional properties advertised and actually observed of a service can vary over time.

Service composition is the responsibility of the service selector, denoted *Selector* below. This component is assigned to a process. *Selector* receives the DAH and acts as an interface between the user of the process and the providers of the services that can be involved in the process. In selecting services, the *Selector* will favor services with more desirable QoS levels. The *Selector* also ensures an appropriate tradeoff between the exploration of possible paths through the DAH and the exploitation of results of past traversals. At each execution (equivalently, traversal), *Selector* evaluates the services used in the process according to their observed QoS levels. While a service is being executed, *Selector* observes the delivered QoS value (OP) for each of the QoS properties in the set $<\mathtt{AC}_{l1},\mathtt{AC}_{l2},...,\mathtt{AC}_{ln}>$ of the client l's QoS requirements. These values are aggregated with the service evaluation process defined below (§5.3) in order to obtain a global rating of the service performance. The better the global rating of the service, the higher its probability of being selected in future executions of the process.

Remark that the observed QoS value is obtained only after the service executes. The *Selector* consequently uses past observations in selecting services for the next execution of the process. To select services that will enter in the composition, *Selector* maintains a database of QoS levels offered in the past, for each service, each execution, and each QoS parameter.

At this point, we have a framework with DAH as the description of all possible allocations of services to tasks in the process, the QoS parameters, the concepts of required, advertised, and observed QoS levels, and the *Selector* component. We thus move to the problem of how to compute the aggregate QoS of each service in the next section.

5.3 Service Evaluation

The aggregate QoS rating of a service is computed using three inputs:

1. *User's QoS requirements. (AC)* Given a set of QoS properties, the input required by the *Selector* from the user are the user's preferences over the values of these properties, and the user's priorities over the properties [103]:
   (a) *Preferences* are used by the user l to define its expectations about values of quality properties. These expectations associate a modality (i.e., maximize or minimize) to a QoS property j ($\mathtt{AC}_{lj}^{mod}$), and give a maximal (for quality properties to maximize) or minimal (for quality properties to minimize) value to each quality property j ($\mathtt{AC}_{lj}^{opt}$). E.g., the user wishes to minimize the execution time with a 2 seconds minimal value.
   (b) *Priorities* determine the relative importance that the user l gives to QoS properties. Each QoS property j is given a weight ($\mathtt{AC}_{lj}^{prior}$) defining its importance relative to other QoS properties. E.g., the user gives more importance to the execution time than to the availability of the service.
2. *Advertised QoS values for each service.* If a service has not yet been executed, *Selector* has no information about past executions. To evaluate the quality of the service it can only use QoS information advertised by the provider of the service: i.e., the set of advertised QoS values AP.
3. *Observed QoS values for each service.* After a service executes, *Selector* can update the rating of the service from the observed QoS values OP.

We first discuss the notions of preference (§5.3.1) and priority (§5.3.2) used here. We then explain how to compute the QoS rating of a service (§5.3.3).

5.3.1 QoS preferences

A user's preferences are given in two tuples, $\mathtt{AC}^{mod}$ and $\mathtt{AC}^{opt}$:

- $\texttt{AC}^{mod}$ contains a modality for each QoS property. A modality is either *maximize* (e.g., availability), which is denoted 1 in the tuple, or *minimize* (e.g., execution time, cost), denoted 0 in the tuple.

- $\texttt{AC}^{opt}$ gives the threshold value for each QoS property. The threshold value is the user's worst acceptable value for the QoS property.

5.3.2 QoS priorities

The tuple $\texttt{AC}^{prior}$ carries a user's priorities over QoS properties. The priorities of quality properties are expressed by weights reflecting the importance that the user gives to each of them. The weights are derived from the order relationship given by the customer on quality properties. E.g., the customer gives more importance to the cost than to the execution time, the weight for the cost QoS property will be more important than the weight of the execution time QoS property. These weights are relative weights, i.e., these reflect the relative importance of each quality property in comparison with other quality properties.

To determine such relative weights, we rely on Multi-Criteria Decision Making (MCDM) methods [55]. Among existing methods [78; 145; 177; 194], we use the Analytic Hierarchy Process [177]. This method is well suited to priorities definition because it lets us fix weights to different criteria with the help of a comparison matrix of QoS properties. The user defines in this matrix the importance and the direction of the comparison for each pair of quality properties. Each quality property is compared with other criteria with an importance on a scale fixed between 1/9 and 9. The direction defines if a QoS property is more or less important than another. For a pairwise comparison x between QoS properties A and B: AxB, A is more important than B if $x > 1$ and A is less important than B if $x < 1$. E.g., the user fixes the pairwise comparison between the execution time and the availability to 5. This means that the execution time is five times more important than the availability according for the user. Similarly, the pairwise comparison between the availability and the execution time is 1/5, meaning that the availability is five time less important than the execution time QoS property. The weights of the QoS properties are obtained with the computation of the right eigenvector of the matrix. The eigenvector is computed by raising the pairwise matrix to powers that are successively squared each time. The rows sums are then calculated and normalized. The computation stops when the difference between the sums in two consecutive computations is smaller than a prescribed value. The values get are associated to QoS properties. For each QoS property j, $\texttt{AC}_{lj}^{prior}$ has one weight.

5.3.3 Computation of the QoS rating

The rating for a transaction is a function of (i) the advertised QoS level ($\texttt{AP}$) or the observed QoS level ($\texttt{OP}$) by the provider of the service s; (ii) the preferences on values of $\texttt{C}$: their modalities $\texttt{AC}^{mod}$ and their optimal values $\texttt{AC}^{opt}$, and; (iii) the priorities of $\texttt{C}$ on quality properties: $\texttt{AC}^{prior}$.

To compute the rating of a service, we use an MCDM method, namely the Simple Additive Weighting (SAW) [81]. This method is based on the weight average; we multiply for each QoS property its relative score according to user's preferences ($\texttt{AC}^{opt}$ and $\texttt{AC}^{mod}$) by their respective weights defined in $\texttt{AC}^{prior}$. Next, these products are summed for all QoS properties. The formula we apply to compute r_s according to the user l, the rating of a service s is as follows:

$$r_s = \sum_{i=1}^{n} \left(\texttt{AC}_{lj}^{mod} \texttt{AC}_{lj}^{prior} \frac{\texttt{XP}_{ki_\alpha j}}{\texttt{AC}_{lj}^{opt}} + (1 - \texttt{AC}_{lj}^{mod}) \texttt{AC}_{lj}^{prior} \frac{\texttt{AC}_{lj}^{opt} - \texttt{XP}_{ki_\alpha j}}{\texttt{AC}_{lj}^{opt}} \right)$$

Above, k is the provider of the service α fulfilling the task i, n is the number of QoS properties and $\texttt{XP}_{ki_\alpha j}$ is the advertised QoS level of the property i ($\texttt{AP}_{ki_\alpha j}$) or the actually observed QoS level of the same property ($\texttt{OP}_{ki_\alpha j}$), in case *Selector* has information about the past executions of s. When QoS properties have to be maximized ($\texttt{AC}_{lj}^{mod}$=1), then the QoS level is computed with $\frac{\texttt{XP}_{ki_\alpha j}}{\texttt{AC}_{lj}^{opt}}$. When quality properties have to be minimized ($\texttt{AC}_{lj}^{mod} = 0 \Rightarrow (1 - \texttt{AC}_{lj}^{mod}) = 1$), then the Qos level is computed with $\frac{\texttt{AC}_{lj}^{opt} - \texttt{XP}_{ki_\alpha j}}{\texttt{AC}_{lj}^{opt}}$.

The service evaluation process provides a rating $r_s \in [-1, 1]$ for a given service s with -1 the worst value and 1 the best value. This rating is used to compute the cost of an edge on the composition. The aim is to minimize the total cost of the composition, so that the cost of an edge is an inverse function of the rating, i.e., a service with a good rating will have a low cost. The cost of edges is fixed between 1 and 2, so the cost c_s of an edge representing the service $s \in [1, 2]$ The transformation function between r_s and c_s is $c_s = 1 + ((1 - r_s)/2)$.

5.4 RL-Based Composition

5.4.1 Baseline

To solve the composition problem, we use an analogy to the shortest path problem. The aim of the shortest path problem is to find a path between two nodes such that the sum of the costs of edges on the path is minimized. In our service composition approach, we instantiate the edge cost by an inverse function of the QoS rating of the service associated to this edge. The cost of an edge is then proportional to the overall quality of the service, i.e., if the service has a good rating, the service has a higher probability to be selected for a future composition than when it has a lower rating. The cost is set on edges and reflects the QoS level of services represented on their successor nodes.

To solve this shortest path problem, we use a Reinforcement Learning [200] (RL) approach; further details are given in Subsection 5.4.2. RL is concerned with how an agent ought to take actions in an environment to maximize its long-term reward. Here, RL is used to select services to execute while maximizing the client's QoS surplus of service compositions.

To find best solutions, the RL approach continuously updates the parameters of QoS performance observed after executing the process. Moreover, once the optimal solution is found, compositions continue to be explored, so that we take into account changes in the pool of available services and/or the QoS levels that the services provide. The approach ensures that either past optimal compositions will be returned, or other compositions will be explored to determine if improvements can be made to historically optimal compositions.

To adapt RL to the services composition problem, we make the following assumptions:

- Before the process has been executed for the first time, the cost of each edge is set according to the QoS level advertised by the provider of the service. However, once the process is executed, the cost of the edges are updated with the QoS level observed in these executions. The user initially knows the QoS advertised by the different providers. With executions, the different possible services are executed and the user estimates the realistic as opposed to advertised QoS values for the services. These values are updated after each execution to take into account the QoS levels to expect from the services.

- Advertised and observed QoS values can change over time. At each execution, these values are memorized and the cost of the edge is computed, and used in subsequent composition choices. This continual updating can therefore accommodate the variations in the pool of the services and their advertised and observed QoS levels.

5.4.2 Reinforcement learning based on randomized shortest paths

We introduce here a reinforcement learning framework that ensures (asymptotic) optimal, continual, exploration in a static environment based on randomized shortest paths (RSP, see [179]). The obtained policy (i.e., composition) is optimal in that it minimizes the expected cost to reach the goal node for a given exploration level. As noted above, continual exploration is particularly important in services composition as the pool of services can change, along with the services' advertised and observed QoS levels.

Let us assume we are given a weighted directed graph with costs $c_{kk} > 0$ associated to each arc. $c_{kk'}$ is either a function of QoS advertised $f(\mathtt{AP}_{ki_\alpha j})$ or actually observed $f(\mathtt{OP}_{ki_\alpha j})$ for the service α represented

with the edge kk' as described in Subsection 5.3. For nodes l, l' not connected by an arc, an infinite cost is assumed, $c_{ll'} = \infty$ so that no jump through this arc is possible. Service compositions are initiated in the network by agents sent from some initial node (assumed to be node of he initial task t_1); their main objective being to reach some goal, destination node (node of the task t_n where n is the total number of nodes) with minimal expected cost.

The RSP framework developed in [179], and following [2], tells us that the optimal policy minimizing the total expected cost for reaching the goal node while maintaining a constant exploration through the graph can easily be computed from an intermediary matrix, $\mathbf{W}$, containing as elements $w_{kk'} = \exp(-\theta c_{kk'})$ for $k \neq n$ and $w_{nk'} = 0$, where θ regulates the exploration. By policy, we mean the assignment, in each node k, of a probability distribution $p_{kk'}$ of jumping to a neighboring node k'. This policy therefore specifies how an agent walks through the graph. The exploration of the graph is quantified by the total expected entropy encountered by an agent during a trajectory or episode, $H = -\sum_k n_k \sum_{k'} p_{kk'} \log(p_{kk'})$ where n_k is the expected number of passages through node k during his trajectory (or episode) from the starting node to the destination node. The total expected cost accumulated over an infinite horizon is minimized, when starting from the initial (or source) node k_0, and following policy π, $v_\pi(k_0) = \mathbb{E}_\pi \left\{ \sum_{t=0}^{\infty} c_{s_t s_{t+1}} | s_0 = k_0 \right\}$ [179]. Since the total expected entropy is a monotonic decreasing function of the parameter θ, it is easier to control θ instead of the entropy H. When $\theta \to \infty$, the policy converges to the deterministic shortest-path policy while when $\theta \to 0$, the agent performs a blind random walk with a uniform distribution of jumping to each neighboring node (and thus not taking costs into account).

It can be shown that the optimal policy, denoted by $p_{kk'}^{\text{opt}}$, is provided by (see [179] for details)

$$p_{kk'}^{\text{opt}} = \frac{w_{kk'} z_{k'n}}{\sum_{l'=1}^{n} w_{kl'} z_{l'n}}, \text{ with } k \neq n \tag{5.1}$$

where the z_{kn} values are

$$\begin{cases} z_{nn} = 1 \\ z_{kn} = \sum_{k' \in S(k)} w_{kk'} z_{k'n} \end{cases} \tag{5.2}$$

and $S(k)$ is the set of successor nodes of node k. Equation (5.1) provides the optimal policy corresponding to a given level of exploration θ. It corresponds to the probability distribution of jumping to a neighboring node k' in each node $k \neq n$. When following this policy, each path $\wp_r$ from the initial node to the destination node, i.e., each service composition, has a probability of $\mathrm{P}(\wp_r) = \exp(-\theta C(\wp_r)) / \sum_r \exp(-\theta C(\wp_r))$ of being chosen, where $\wp_r$ is the path number r and $C(\wp_r)$ is the total cost associated to that path [179]. Thus, highly desirable service compositions have a high likelihood while undesirable service compositions are associated to a low likelihood.

Let us now recast this procedure into a reinforcement learning algorithm where the agent observes dynamically the costs and updates his policy while exploring. It suffices to transform Equation (5.2):

$$z_{kn} = \sum_{k' \in S(k)} w_{kk'} z_{k'n} \tag{5.3}$$

$$= \sum_{k' \in S(k)} p_{kk'}^{\text{opt}} \left(\frac{w_{kk'}}{p_{kk'}^{\text{opt}}} z_{k'n} \right) \tag{5.4}$$

$$= \mathbb{E} \left[\frac{w_{kk'}}{p_{kk'}^{\text{opt}}} z_{k'n} \right] \tag{5.5}$$

Reinforcement learning aims is to directly estimate the expectation from the observation of the immediate cost and the value of $z_{k'n}$ in the next node k' [200]. There is a large range of potential techniques for doing that, depending on the problem at hand (see for example [27; 197]). One could simply use exponential smoothing or, alternatively, rely on a stochastic approximation scheme by letting $\alpha(t)$ decrease

Algorithm 2 Computation of the optimal policy while fixing the exploration of the network: a simple reinforcement learning algorithm.

Require:
- Node 0 is the initial node while node n is the goal node.
- $\theta > 0$: the parameter controlling the degree of exploration.

Initialize $\hat{c}_{kk'} = f(s_{kk'}^{aQoS})$ if arc (k, k') exists; $\hat{c}_{kk'} = \infty$ otherwise
Compute $\hat{w}_{kk'} = \exp(-\theta \hat{c}_{kk'})$
Initialize $\hat{z}_{kn} = 1$
$t \leftarrow 0$ { t is the episode counter}
repeat
  $k \leftarrow 0$ { start an episode at initial node 0}
  repeat
    Choose next node k' according to probability $\hat{p}_{kk'}^{\text{opt}} = \dfrac{\hat{w}_{kk'}\hat{z}_{k'n}}{\sum_{l'=1}^{n} \hat{w}_{kl'}\hat{z}_{l'n}}$
    Follow the arc (k, k') and observe the current cost, $\hat{c}_{kk'} \leftarrow c_{kk'} = f(s_{kk'}^{oQoS})$
    Compute $\hat{w}_{kk'} = \exp(-\theta \hat{c}_{kk'})$
    Update $\hat{z}_{kn} \leftarrow \hat{z}_{kn} + \alpha(t)\left[\dfrac{\hat{w}_{kk'}}{\hat{p}_{kk'}^{\text{opt}}}\hat{z}_{k'n} - \hat{z}_{kn}\right]$
  until the goal node n is reached, $k' = n$ { episode number t is over}
  $t \leftarrow t + 1$
until convergence of the policy
return The policy (transition-probabilities) matrix containing the elements $\hat{p}_{kk'}^{\text{opt}}$

over time t,

$$\hat{z}_{kn} \leftarrow \hat{z}_{kn} + \alpha(t)\left[\frac{\hat{w}_{kk'}}{\hat{p}_{kk'}^{\text{opt}}}\hat{z}_{k'n} - \hat{z}_{kn}\right] \tag{5.6}$$

which converges for a suitable decreasing policy of $\alpha(t) \in]0, 1[$ [197]. The hats in Equation (5.6) denote estimated values. This leads to the reinforcement learning scheme detailed in Algorithm 2.

Computational complexity

The computational complexity of this algorithm is similar to the complexity of the SARSA reinforcement learning algorithm [200], and is difficult to quantify exactly for various reasons [104], among others, because of its stochastic nature.

5.5 Experimental Results

5.5.1 Comparison to classical competing methods

This subsection details the first experiments with Algorithm 2 presented. The experiments are based on the process given in Figure 5.1. For all services s and QoS properties j, $\texttt{AP}_{ki_\alpha j}$ and $\texttt{OP}_{ki_\alpha j}$ were randomly assigned with $\texttt{AP}_{ki_\alpha j} \geq \texttt{OP}_{ki_\alpha j}$, reflecting that providers usually overestimate their performances. The rating of a service s, r_s, is initialized with s^{aQoS} when the service s has not yet been executed and computed with s^{oQoS} after the execution of the service. r_s then belongs to the interval $[-1, 1]$. It has been further assumed than $c_s = 1 + \frac{1-r_s}{2}$ and that, the cost associated to the edge representing the service s, c_s belongs to $[1, 2]$.

The aim of our experiments is to show that for a given degree of exploration, our approach provides the most efficient solution. We therefore compare our approach to two similar standard methods of exploration: ϵ-greedy and Bolztmann [200]. The most efficient solution is the one that maximizes the QoS surplus offered to the client of the process. As the QoS surplus is an inverse function of the cost

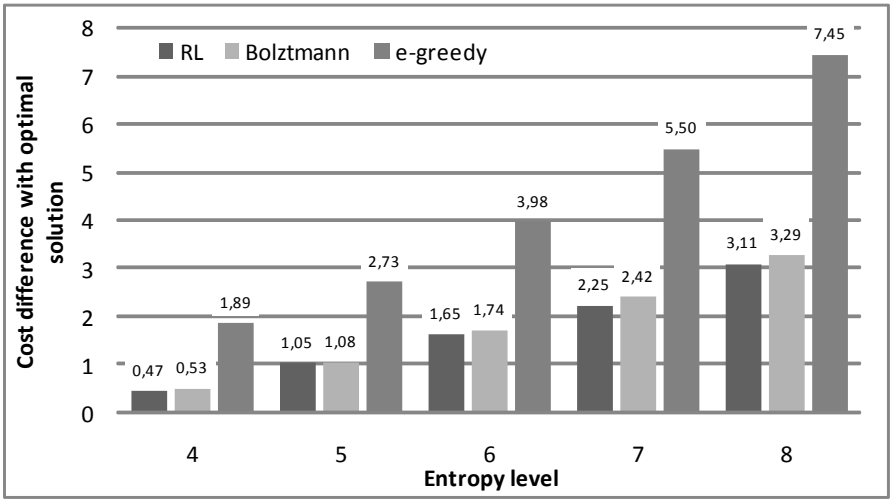


Figure 5.2: Comparison to similar methods

assigned to edges associated to services, the most efficient solution is the one that minimizes the cost of the composition.

Figure 5.2 illustrates the results of our experiments. For five given levels of entropy, we compute the main cost of composition executions and compare it to the cost of the optimal solution based on Algorithm 2. The Figure 5.2 shows that the difference between the optimal cost and the mean cost provided by the different methods is the smallest for our RL approach.

The result are based on 20,000 process executions for the following entropy levels: H = 4, 5, 6, 7, and 8. The whole procedure was launched 20 times (runs) and average results are computed on these 20 runs. The QoS level offered (OP) of all services of the process was modified after 10,000 executions to reflect changes that can occur in the services behavior. We show that for all entropy levels, our method has the smallest cost difference and is then the most efficient. We clearly observe that our method provides much better results than the ϵ-greedy (e-greedy) method and our results are slightly better than those provided by the Bolztmann method (Bolztmann).

However, the proposed composition method is not easily generalized to the stochastic shortest path problem with a stochastic action-path transition.

5.5.2 Entropy impact to variations of quality level

The second experiments involving the RL algorithm illustrate the effect of the entropy on the service composition problem. These experiments underline the adaptation abilities of the RL algorithm while the QoS offered by providers is changing over time (dynamic environment). The adaptation abilities are observed at different levels of entropy, depending on the θ parameter. The experiments are based on the process provided in Figure 5.1.

We initially define a shortest path on the process in which costs c_s of services s belonging to the shortest path are set to 3. All other services s of the process have costs (c_s) set to 6 (Configuration A). The whole simulation includes 20000 runs of the service composition. To simulate the adaptation of the RL algorithm, we modify the quality levels offered after 5000 runs. To modify the quality levels offered, we set the costs (c_s) of services s that were previously set to 6 to their new value: 1 (Configuration B). The best path identified for the firsts 5000 runs becomes the worst with all other paths outperforming its capabilities. We observe the process adaptation for different levels of entropy (i.e.: different values of the θ parameter). The Figures 5.3, 5.4, 5.5, 5.6, 5.7 and, 5.8 illustrate respectively the evolution of the average cost of the service executions with $\theta = 0.5, 1, 1.5, 2, 2.5, 3$.

We can observe that when the entropy is high (low θ value), the service composition is adapted quickly. With a small entropy (high θ value), the composition needs more time to be tuned. However, the average

cost is depending also depending on the entropy. The average cost can be smaller with a high θ value (e.g.: $\theta = 2$ in Figure 5.6) than with a low θ value (e.g.: $\theta = 1$ in Figure 5.4).

We made additional experiments to illustrate the RL algorithm behavior in a highly dynamic environment. Rather than modifying the quality levels after 5000 executions, we alternate quality levels (Configuration A and Configuration B) after each set of 2500 service compositions. Initially, the quality levels are defined with Configuration A. After 2500 executions, these are defined with Configuration B. After 5000 executions, the quality levels are again defined with the Configuration A. The configurations are continuously modified until the end of the 20000 executions. These adaptations to the dynamic environment are observed for different levels of entropy. The Figures 5.9, 5.10, 5.11, 5.12, 5.13 and, 5.14 illustrate respectively the evolution of the average cost of the service executions with $\theta = 0.5, 1, 1.5, 2, 2.5, 3$.

We can conclude that the RL algorithm adapts quickly with a low θ value. However, a low θ value implies a continual exploration of the process and the utilization of suboptimal services. The entropy level must be adapted according to the dynamic degree of the environment. A stable environment requires mainly exploitation of results while a very dynamic environment requires exploration of available services compositions. Finally, some values of θ provide better average costs than others (e.g.: $\theta = 1.5$ in Figure 5.11). The θ parameter could be adapted to the composition process in a further step of the algorithm.

5.6 Related Work

QoS-based resolution of service composition has been considered in other work [69; 91; 123; 161; 227; 229; 231]. Although these proposals rely on QoS, most them give a static definition of QoS. QoS properties accounted in the selection or composition are most of the time already determined and do not reflect choices of the user. Zhang et al. [231] minimizes the throughput and the response time while Jaeger et al. [91] aggregates a set of QoS properties (execution price, availability, reliability, duration and reputation). Our approach gives more freedom to the user and leaves open the QoS ontology, so that the user can decide which QoS properties will be accounted during composition.

Most QoS based composition methods relies on an aggregate of QoS properties. The aggregation process is, however, not always well defined. We expressed user's requirements via priorities between quality properties and preferences over the values of quality properties. Preferences have already been addressed in other approaches with a direction attribute to indicate if a property has to be minimized or maximized [91; 123; 153; 229]. However, our definition of the optimal value summarizes how to evaluate the performance of a service from the user requirements. The priority expectations is defined in some proposals with means of a weight attribute associated to quality properties. However, there are no clear instructions about how the weighting is made [30; 69]. We thereby fill the gap between the requirements of the client and the weighting of QoS properties.

Most existing QoS composition approaches aim at summing QoS values of services entering in the composition rather than computing their individual performance [91; 227; 229; 231]. Our proposal focus on the individual evaluation of each service candidate to the whole composition. Our approach also has the interesting property of being able to accommodate changes in the pool of services and the variations of their advertised and observed QoS levels. Observe that we do not need to compute all possible paths when a new service enters to determine its performance. In contrast, many composition methods propose solutions for a fixed composition schema, without the possibility of adding and removing candidate services [15; 30; 114].

Ko et al. [114] suggest a QoS-oriented web service composition planning architecture. This architecture maintains expert-made composition schemes in a service category and assists a user to choose the schema to use. Chen and Zhang [37] propose an ant colony algorithm to schedule large-scale workflows with various QoS parameters. This algorithm enables users to specify their QoS preferences as well as define the minimum QoS thresholds. The objective of their algorithm is to find a solution that meets all QoS constraints and optimizes the user-preferred QoS parameter. Kormaz and Krunz [117] formulate an algorithm to find a feasible route that satisfies end-to-end QoS requirements of a request while efficiently

using resources. They propose a randomized heuristic search to find a feasible path. All these proposed approaches to the service composition and/or selection issue involve QoS calculation to define the efficiency of a composition. However, most of these solutions are static, i.e., each time a service changes its quality possibilities, the approach must be executed again. Our approach enables a continual exploration of the dynamic environment of service compositions. In the context of an evolving environment, we provide more interesting results.

In comparison to our previous work [101], this approach gives better results. Here, the entropy is assigned globally rather than at each node. The global spread of entropy allows us to assign automatically the entropy at each node to provide better results for a given exploration rate. More details on this aspect can be found in [179].

The Appendix B provides more information about service composition methods.

5.7 Conclusions

This chapter focused on the problem of service composition in a particular, but realistic setting. The setting has the following characteristics:

1. potentially many QoS properties describe services' QoS levels;
2. the pool of available services can change over time;
3. the QoS levels advertised by the service providers can change over time;
4. the QoS levels actually observed after the execution of services can change over time;

We proposed a two-step service composition approach applicable to settings having the characteristics cited above. The first step amounts to aggregate QoS level over many criteria so as to obtain a single rating of a service. The second step consists of using the computed ratings of the candidate services as the input to a reinforcement learning (RL) algorithm, which finds the composition of services that maximizes the overall QoS level delivered to the user. The algorithm continually updates the QoS ratings to reflect changes of the kinds 2–4 above. Our first experiments reported here illustrate that, for a given level of exploration, our algorithm is more efficient than comparable approaches. Our proposal is generic, in the sense that we make as little commitments as feasible: e.g., we do not commit to a QoS ontology, how composition proceeds once a composition is identified, how interoperability is ensured, and so on. This ensures that our results are generic. Said choices are left to the designer.

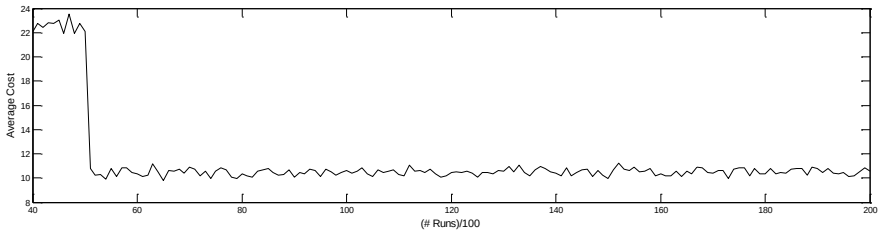


Figure 5.3: Evolution of the average cost with $\theta = 0.5$

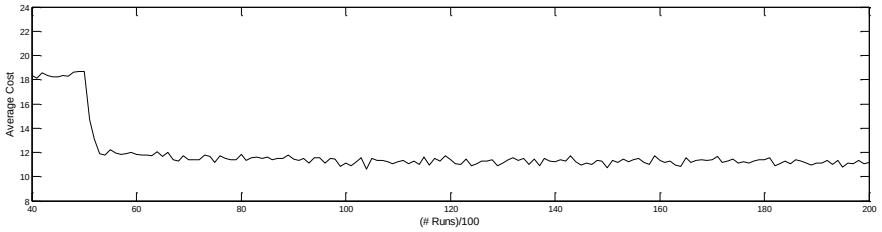


Figure 5.4: Evolution of the average cost with $\theta = 1$

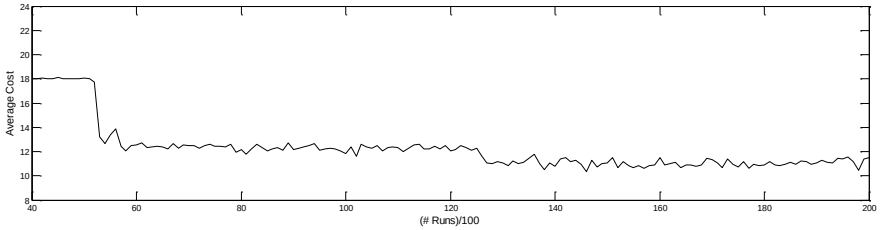


Figure 5.5: Evolution of the average cost with $\theta = 1.5$

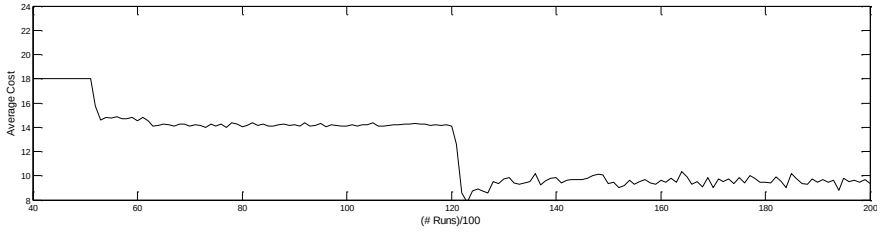


Figure 5.6: Evolution of the average cost with $\theta = 2$

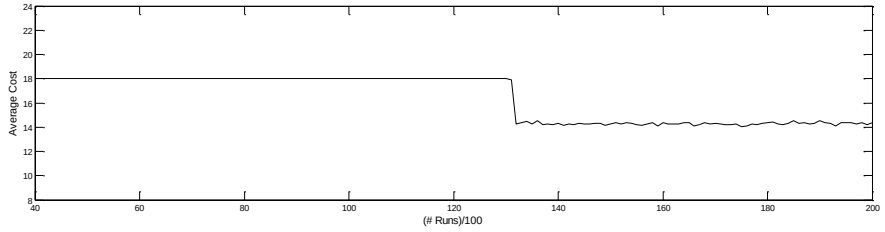


Figure 5.7: Evolution of the average cost with $\theta = 2.5$

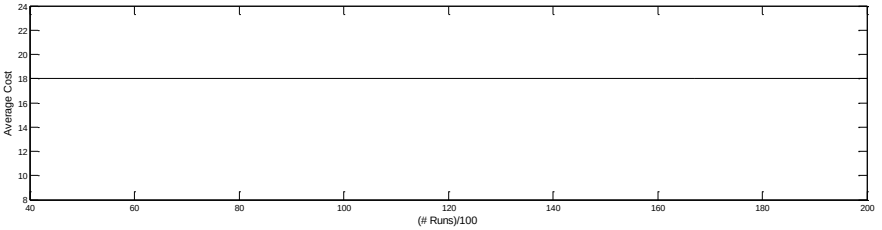


Figure 5.8: Evolution of the average cost with $\theta = 3$

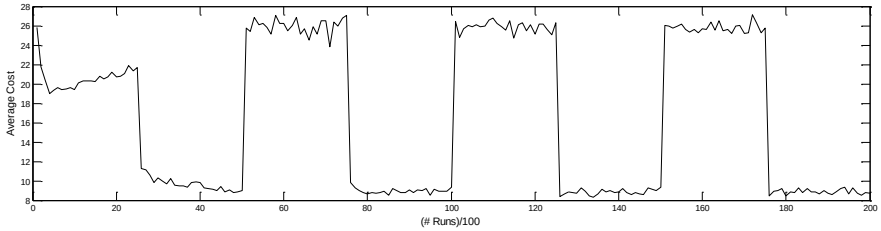


Figure 5.9: Evolution of the average cost with $\theta = 0.5$

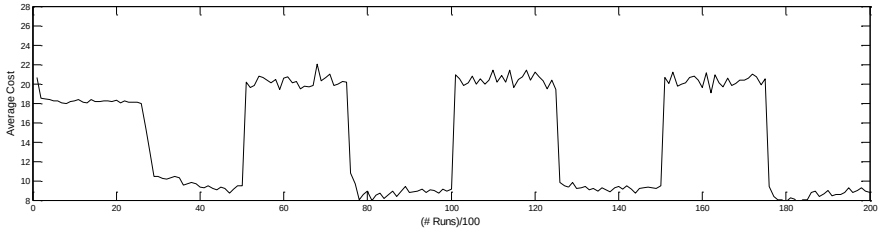


Figure 5.10: Evolution of the average cost with $\theta = 1$

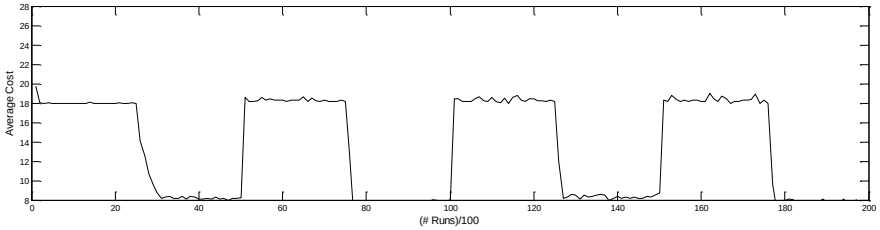


Figure 5.11: Evolution of the average cost with $\theta = 1.5$

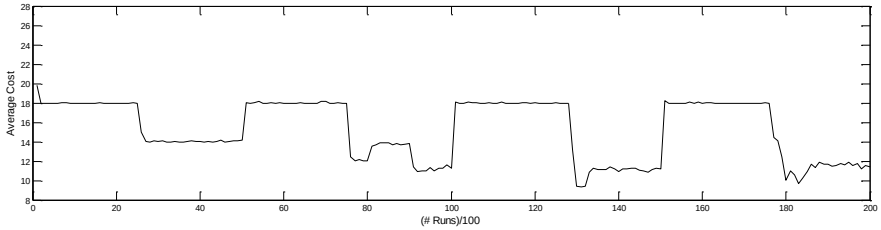


Figure 5.12: Evolution of the average cost with $\theta = 2$

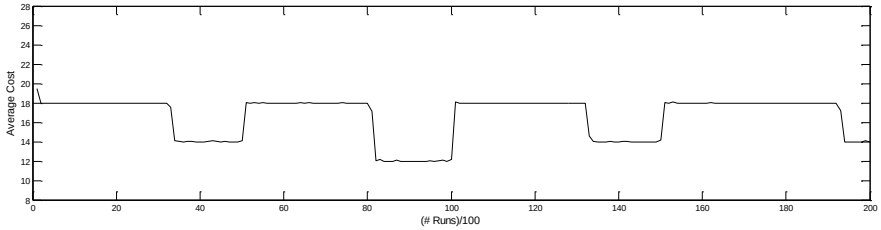


Figure 5.13: Evolution of the average cost with $\theta = 2.5$

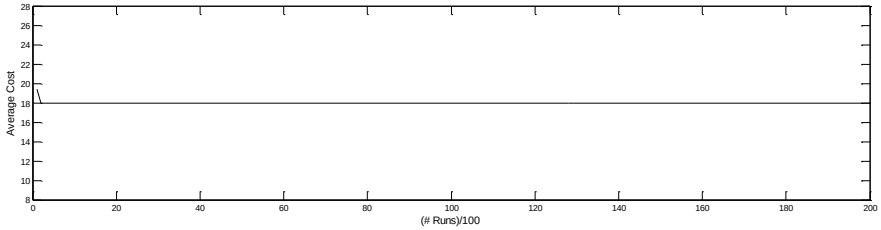


Figure 5.14: Evolution of the average cost with $\theta = 3$

Chapter 6

User Profiling

This Chapter outlines a method enabling to define user profiles of services requesters. This method relies on quality information specified with the QVDP model proposed in Chapter 3. To increase the efficiency of services management, this method proposes to automatize the service selection with results of past transactions. The Chapter 4 outlined a selection method relying on quality specification of stakeholders. The method proposed in this Chapter relies on quality information but also on other criteria as the brand image effect or the possible optimism/pessimism of services requesters. We propose to define user profiles according to such criteria and results of past transactions. The definition of user profiles is useful to predict the satisfaction of a user regarding a service transaction. The satisfaction prediction will then be used to select and predict services executions that meet requester's expectations.

6.1 Introduction

Reputation has a critical role in any system where service providers and service users meet to effect transactions. Just as a price of a service is supposed to signal the value that the market participants see in a having the service delivered, the reputation (score) of a service provider is supposed to signal the extent, to which service user can trust the providers' announcement of the quality of service that it will deliver in a transaction.

A price of a service can be unreliable, in the sense that it departs from the price that would have been set through the match between the demand and supply of that service in a given market. An unreliable price will affect the relationship between demand and supply, making the market, broadly speaking, inefficient. Since reputation is a signal similar to a price, in that it affects the choice of buying one service from one provider or another, the reliability of a reputation score is critical for the efficient operation of a market.

To pursue the analogy between prices and reputation scores, a reputation score can be "fixed", in the same way that a price can be "fixed" on a market. An important aspect, in which prices and reputations differ however, is that fixing the former is against the law, as the theory of anti-trust legislation illustrates. In absence of legislation, research surveyed further down (§6.5) focused on the design of incentive mechanisms intended to motivate the provision of honest feedback, as opposed of that which is strategically manipulated to suit the aims of particular market participants.

A service provider's reputation is a function of the feedback, given after every past transaction by the users of this provider's service. It is reasonable to expect a reliable reputation score to reflect the differences between the quality levels advertised by the provider, and those delivered in transactions. The motivation for the work presented in this chapter comes from the observation that even if it is possible to compute honest feedbacks, the resulting reputation score can be unreliable. This is due to interpersonal differences in human users of the services, and the simple observation that quality is a subjective experience. Two honest users can still provide different feedbacks, even if the service provider advertised and delivered the same quality levels to both of them.

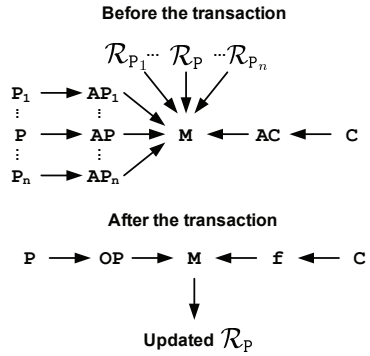


Figure 6.1: Symbols representing the actors and information involved before and after a transaction.

Our response to the potential unreliability of honest reputation scores is to identify sources of bias other than dishonesty, and explain how to compute them. In the rest of the introduction, we outline more precisely the context and some terminology that spans this chapter (§6.1.1), then state the problem (§6.1.2), and outline our contributions (§6.1.3).

6.1.1 Context

In this chapter, we are interested in a system, in which many providers and clients participate. Clients provide requirements, and providers deliver services, which satisfy these requirements. Since more than one provider may be able to deliver the same service, a third kind of participants are mediators, which choose among competing providers on the behalf of clients. A transaction involves a client, a mediator, and a set of providers. The steps of the transaction are: (1) the client provides requirements; (2) the mediator chooses among the providers the one to deliver the service according to the requirements; (3) the client observes the output of the service; (4) the client provides feedback. We assume that a client observes at no cost the reputation of the various providers.

We consider that in a transaction, the provider `P` fulfills some request of a client `C` (in other words, `P` delivers a service to `C`). Prior to the transaction, each `P` advertises its tuple `AP` of quality levels that it can deliver when performing a request. E.g., a value in the tuple may indicate that `P` achieves a response time of X milliseconds. Values advertised via `AP` need not be honest: `P` may state values that differ from its actual abilities. Also prior to the transaction, each `C` announces a tuple `AC` of quality requirements. In order for the transaction to take place, for a given `C`, an agent `M` compares alternative `P`s that can satisfy `AC`, and selects one `P` on the basis of its `AP` and its reputation score $\mathcal{R}_P$, as illustrated in Figure 6.1.

After the transaction takes place, `C` observes a tuple `OP` of delivered quality values. `C` leaves feedback, which is a unique feedback value `f`. `M` updates the reputation score for the relevant provider, whereby a reputation score of `P` is an aggregate $\mathcal{R}_P$ (henceforth denoted only $\mathcal{R}$) of feedbacks of past transactions.

$\mathcal{R}$ is important because it is an input to the decision procedure used in selecting among competing providers. It is therefore essential to ensure that the reputation score is reliable. *A reliable reputation score reflects true past differences between the announced `AP` and observed `OP` levels of quality of provider agents in each transaction.* $\mathcal{R}$ is a function of `AP` and `OP` from past transactions. Ideally, $\mathcal{R}$ will *only* depend on `AP` and `OP`.

In reality $\mathcal{R}$ depends, however, on more than `AP` and `OP` from past transactions. Since $\mathcal{R}$ affects the probability for a provider agent to be selected for future transactions, strategic manipulation of feedbacks is bound to happen in order to manipulate $\mathcal{R}$s in favor or against particular `P`s. In such a case, $\mathcal{R}$ depends on additional parameters, themselves determined by `C`'s considerations that may have nothing to do with `P`'s `AP` and `OP`. This makes $\mathcal{R}$s unreliable, for they may be manipulated to induce to error in selecting `P`s for future transactions. In response, attention has been placed on the design of incentive mechanisms that motivate honest feedback. E.g., Jurca and Faltings [98] describe an incentive mechanism that supports the equilibrium where truthful feedback is obtained in an electronic market. Dellarocas [42] proposes a set of mechanisms intended to significantly reduce and ideally eliminate the

negative effects of fraudulent behavior, such as when conspiring buyers intentionally give unfair scores to sellers, or when sellers discriminate on the quality of service they provide to buyers.

6.1.2 Problem

Effectively, reputation scores never reflect the aggregation of honest feedbacks. In other words, even if we manage to ensure that feedback is honest via incentive mechanisms or otherwise, *we cannot ensure that the human users who provide feedback via* `C`*s are homogeneous*. The main source of the problem is that *quality is a subjective experience*: two human users may give different honest feedback for exactly the same true difference between `AP` and `OP`. This is an uncontroversial claim, well established in management science (for surveys, see [165; 175]) and noted in software engineering (e.g., [113]). In our terminology, an honest $\mathcal{R}$ varies across human users, for whom `C`s are proxies. With this in mind, the pressing question is: *How to account for human users' interpersonal differences in order to determine the origin of unreliable* $\mathcal{R}$*?*

6.1.3 Contributions

In response to the problem, we propose to observe the reliability of reputation scores by accounting for differences in human users' *feedback profiles*. To this aim, we describe a method for computing users' feedback profiles. The method computes feedback profiles by comparing, at each transaction and for each `C`, "true" feedback $\bar{\texttt{f}}$ with the feedback `f` given by `C` after the transaction, whereby `f` is elicited from `C`.

True feedback $\bar{\texttt{f}}$ is an estimate of the feedback that abstracts from the user's feedback profile. True feedback $\bar{\texttt{f}}$ is a function of (i) `C`'s preferences on values of quality properties (e.g., lower values of response time are preferred to higher ones), (ii) `C`'s priorities (equivalently, importance) over quality properties (e.g., availability is more important than response time), (iii) `AP`, which gives the values of quality properties advertised before the transaction (i.e., quality levels proposed by `P`), (iv) `OP`, which gives the values of quality properties observed after the transaction (i.e., quality levels delivered by `P`). We reasonably assume that feedback reflects a user's perception of quality, which leads us to compute $\bar{\texttt{f}}$ according to the established conceptualization of *perceived quality* in management science. According to the standard definition from Parasuraman et al. ([166]: p.46), "the quality that a client perceives in a service is a function of the magnitude and direction of the gap between expected service and perceived service." True feedback $\bar{\texttt{f}}$ therefore amounts to an unbiased quantification of the gap between user's expressed expectations – i.e., `AC`, preferences, and priorities –, expected values of quality properties before the transaction – i.e., `AP` –, and observed values of quality properties after the transaction – i.e., `OP`.

Assuming that users' feedback is honest, a feedback profile of the user is the source of difference between $\bar{\texttt{f}}$ and `f`. On the basis of differences between $\bar{\texttt{f}}$ and `f` for a given user, our profiling method computes the feedback profile for that user. A feedback profile characterizes a user on three orthogonal dimensions, called *outlook*, *sensitivity to deception* and *sensitivity to brand image*, as illustrated in Figure 6.2. *Outlook* describes a user's degree of optimism or pessimism in feedbacks. *Outlook* reflects the fact that some users consistently give more favorable or defavorable feedback. We then have two kinds of sensitivity. The *sensitivity to deception* characterizes a user's degree of deception when delivered quality varies from expected quality. Deception occurs while the quality level delivered is lower than the quality level expected by the user. Users can be slightly or strongly sensitive to deception and can be averse if the user has an aversion effect to the deception. Deception sensitive users will react negatively to deception (i.e., they will decrease their feedbacks) while deception averse ones will positively react (i.e., they will increase their feedbacks) when the observed level of quality is lower than the quality level advertised by the provider. In other words, for the same deception related to the same gap between quality expected and delivered, users will react differently. The *sensitivity to brand image* characterizes the effect of provider's brand image on the user. Users will react differently to provider's image, their feedbacks can be sensitive to the image of the provider but also adverse if the user has an aversion to image of providers. Users that are brand image sensitive will improve their feedbacks for providers with a notorious brand image while users averse to brand image will decrease their feedbacks for such users.

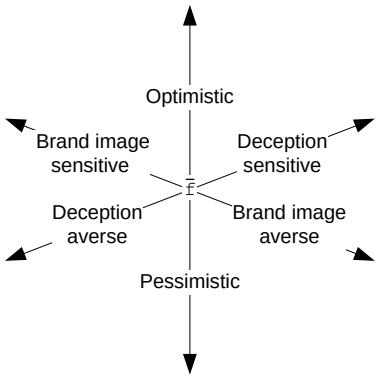


Figure 6.2: Dimensions of feedback profiles.

Figure 6.2 illustrates the three dimensions of feedback profiles. The figure points out that any feedback has a position in relation to the true feedback. It is by computing the displacement of the feedback from the true feedback over transactions that we estimate a feedback profile. In case a user's feedback profile changes over transactions, our profiling method continually adjusts the computed feedback profile to account for any changes in outlook and sensitivities.

Organization. Our profiling method is used by the mediator M, which selects one among many competing Ps on behalf of an agent C. The method proceeds in two steps at each transaction. The first step (§6.2) occurs before the transaction: M transforms user's expectations (i.e., AC, preferences, and priorities) so that they can be used after the transaction to compute the true feedback $\bar{\mathtt{f}}$. The second step (§6.3) takes place after the transaction: M computes the true feedback and obtains the user's feedback f, then updates the estimate of the user's feedback profile (i.e., the estimate of the user's outlook and sensitivity). After we explain the two steps, we report the results of our experiments in computing feedback profiles (§6.4). We discuss related work (§6.5) and summarize conclusions (§6.6).

6.2 Computing True Feedback

The true feedback $\bar{\mathtt{f}}$ of a transaction i occurring between a client l and a provider k is defined by x_{kli}. x_{kli} is derived from user l expectations (i.e., $\mathtt{AC}_l$: l's hard constraints, preferences, and priorities) and from the gap observed between the expected quality values ($\mathtt{AP}_{ki}$) advertised by the provider k at transaction i, and the observed quality values ($\mathtt{OP}_{ki}$) delivered by the provider k at that same transaction i. $\mathtt{AC}_l$ has three components:

- Hard constraints in $\mathtt{AC}_l$ express the quality requirements of the service user. These requirements act as constraints during the selection of the provider k. For each quality property requirement in $\mathtt{AC}_l$, the quality proposed in $\mathtt{AP}_{ki}$ by provider k must meet the level required.

- Preferences in $\mathtt{AC}_l$ are used by the client l to define its expectations about values of quality properties. These expectations associate a modality (i.e., maximize or minimize) and a maximal (for quality properties to maximize) or minimal (for quality properties to minimize) value to each quality property defined in $\mathtt{AC}_l$.

- Priorities in $\mathtt{AC}_l$ determine the relative importance of quality properties according to client l. Each quality property of $\mathtt{AC}_l$ is related to a weight defining its impact in comparison to other quality properties.

6.2.1 Preparing preferences

The preferences of a client l about quality properties defined in $\mathtt{AC}_l$ are defined via two tuples: $\mathtt{AC}_l^{mod}$ and $\mathtt{AC}_l^{opt}$.

$\mathtt{AC}_l^{mod}$ contains the modality of the different quality properties of $\mathtt{AC}_l$. The client l will desire to *maximize* some quality properties (e.g., availability, security) and to *minimize* others (e.g., execution time, cost). The maximization or the minimization is the modality associated to each quality property. For each quality property in $\mathtt{AC}_l$, $\mathtt{AC}_l^{mod}$ has a related value in $\{-1, 1\}$. This value expresses the modality of the quality property, i.e., -1 if the property is to be minimized and 1 if the property is to be maximized.

$\mathtt{AC}_l^{opt}$ contains the optimal values for each quality property defined in $\mathtt{AC}_l$ according to the client l. The client l has a reference optimal value for each quality property. Quality properties are expressed on defined scales and the optimal value is the maximal value on the scale. E.g., on a percentage scale, this value is 100. While the modality of a quality property is *minimize*, the optimal value of client l is its maximal reference. E.g., client l expects a maximal execution time of 20 sec. For each quality property in $\mathtt{AC}_l$, $\mathtt{AC}_l^{opt}$ has a related optimal value.

6.2.2 Preparing priorities

The priorities of a client l about quality properties defined in $\mathtt{AC}_l$ are defined in a tuple: $\mathtt{AC}_l^{prior}$.

The priorities of quality properties in $\mathtt{AC}_l$ are expressed by weights reflecting the importance that the client l gives to each of them. The weights are derived from the order relationship given by l on quality properties of $\mathtt{AC}_l$. E.g.; the client l gives more importance to the cost than to the execution time, the weight of the cost will be more important than the weight of the execution time. These weights are relative weights, i.e., these reflect the relative importance of each quality property in comparison with other quality properties of $\mathtt{AC}_l$.

To determine these weights, we rely on Multi-Criteria Decision Making (MCDM) methods [55]. Among existing methods [56; 78; 145; 177; 194], we use the Analytic Hierarchy Process [177]. This method is well suited to priorities definition because it enables to fix weights to different criteria with the help of a comparison matrix of quality properties. The client l defines in this matrix the importance and the direction of the comparison for each pair of quality properties in $\mathtt{AC}_l$. Each quality property is compared with other criteria with an importance on a scale fixed between 1/9 and 9. The direction defines if a quality property is more or less important than another. For a pairwise comparison x between quality properties A and B: AxB, A is more important than B if $x > 1$ and A is less important than B if $x < 1$. E.g.; the client l fixes the pairwise comparison between the cost and the execution time to 5. It means that the cost is five times more important than the execution time according to client l. Similarly, the pairwise comparison between the execution time and the cost is 1/5, meaning that the execution time is five time less important than the cost for the client l. The weights of the quality properties are obtained with the computation of the right eigenvector of the matrix. The eigenvector is computed by raising the pairwise matrix to powers that are successively squared each time. The rows sums are then calculated and normalized. The computation is stopped when the difference between these sums in two consecutive calculation is smaller than a prescribed value. The values get are associated to quality properties of $\mathtt{AC}_l$. For each quality property in $\mathtt{AC}_l$, $\mathtt{AC}_l^{prior}$ has a related weight.

6.2.3 Feedback evaluation

Eliciting preferences and priorities allows the client l to perform an evaluation of the quality level received according to its expectations. The true feedback x_{kli} of client l for a transaction i is a function of (i) the expected quality level initially advertised by the provider k for transaction i: $\mathtt{AP}_{ki}$; (ii) the observed quality level offered by k during transaction i: $\mathtt{OP}_{ki}$; (iii) the preferences on values of the client l: their modalities $\mathtt{AC}_l^{mod}$ and their optimal values $\mathtt{AC}_l^{opt}$, and; (iv) the priorities of the client l on quality properties defined in $\mathtt{AC}_l$: $\mathtt{AC}_l^{prior}$.

To compute the true feedback x_{kli}, we use a MCDM method, the Simple Additive Weighting (SAW) [81]. This method is based on the weight average; we multiply for each quality property of $\mathtt{AC}_l$ the difference

between expected quality levels ($\mathtt{AP}_{ki}$) and observed quality levels ($\mathtt{OP}_{ki}$) by their respective weights defined in $\mathtt{AC}_l^{prior}$. Next, these products are summed for all quality properties defined in $\mathtt{AC}_l$. However, the difference between quality levels of $\mathtt{AP}_{ki}$ and $\mathtt{OP}_{ki}$ is computed according to client l preferences.

The formula allowing to compute the true feedback x_{kli} is given here:

$$x_{kli} = \sum_{j=1}^{n} \mathtt{AC}_{lj}^{mod} \mathtt{AC}_{lj}^{prior} \frac{|\mathtt{OP}_{kij} - \mathtt{AP}_{kij}|}{\mathtt{AC}_{lj}^{opt}}$$

Where n is the total number of quality properties defined in $\mathtt{AC}_l$. When quality properties have to be maximized ($\mathtt{AC}_{lj}^{mod}$=1), the gap between the quality level expected and observed is computed with $\mathtt{OP}_{kij}-\mathtt{AP}_{kij}$. When quality properties have to be minimized ($\mathtt{AC}_{lj}^{mod} = -1 \Rightarrow |\mathtt{AC}_{lj}^{mod} - 1| = 1$), the gap between the quality level expected and observed is computed with $\mathtt{AP}_{kij}-\mathtt{OP}_{kij}$. $(\mathtt{OP}_{kij} - \mathtt{AP}_{kij})/\mathtt{AC}_{lj}^{opt}$ quantifies the direction and the magnitude of the gap between the observed quality level and the advertised quality level, whereby that gap is compared to the optimal desired level of quality for the given quality property. $\mathtt{OP}_{kij} - \mathtt{AP}_{kij}$ is proportional to client l's perceived quality, as perceived quality is usually conceptualized in marketing. We interpret $(\mathtt{OP}_{kij} - \mathtt{AP}_{kij})/\mathtt{AC}_{lj}^{opt}$ as follows:

1. $(\mathtt{OP}_{kij} - \mathtt{AP}_{kij})/\mathtt{AC}_{lj}^{opt} \in [0, 1]$ is the provider's overperformance, as the proportion of the optimal quality level delivered by the provider above the announced quality level;
2. $(\mathtt{OP}_{kij} - \mathtt{AP}_{kij})/\mathtt{AC}_{lj}^{opt} = 0$ the provider delivered as promised;
3. $(\mathtt{OP}_{kij} - \mathtt{AP}_{kij})/\mathtt{AC}_{lj}^{opt} \in [-1, 0]$ is the provider's underperformance, as the proportion of the optimal quality level delivered by the provider below the announced quality level.

6.3 Computing a Feedback Profile

6.3.1 Description of the basic model

The feedback ($\mathtt{f}$) given by a user to a provider after a transaction is not the true feedback($\bar{\mathtt{f}}$). The true feedback is determined by the user's actual feedback, characterized by the feedback profile.

The feedback profile of a client is defined over multiple transactions occurring with different providers. Assume we have a set of n_p providers k and n_c clients l. Each provider k has a latent brand image, q_k, that is hidden to the external world. We define the brand image score associated to provider k as the estimate of q_k based on empirical data. The total number of transactions is denoted by N. Therefore, each provider k is characterized by one single feature, his brand image q_k.

On the other hand, the client l who asks for the transaction i rates it based on the inspection of its quality x_{kli}. Indeed, each time the provider k realizes a transaction i, its quality in accordance to user l expectations ($\mathtt{AC}_l$): x_{kli} is observed. This quantity provides a measure of the difference between the observed quality of the service and the promised quality. Here, we assume that the client l can be characterized by four different features: (i) his sensitivity with respect to the brand image of the provider, a_l^q, (ii) his sensitivity with respect to the deception related to the quality of the provided service, a_l^x, (iii) his bias, b_l, and (iv) his stability in providing constant ratings for a fixed observed quality, σ_l. The Definition 8 details the components of the feedback profile.

Definition 8. Feedback Profile. The feedback profile of a user l is a tuple

$$\mathcal{F}_l = (\mathtt{AC}_l^{pref}, \mathtt{AC}_l^{prior}, a_l^x, a_l^q, b_l)$$

where:

- $\mathtt{AC}_l^{pref}$ is an n-tuple of pairs $(\mathtt{AC}_l^{opt}, \mathtt{AC}_l^{mod})$, with
  - $\mathtt{AC}_{lj}^{opt}$ is the optimal quality level desired by the client l for the quality property (i.e., criterion) j; and

  - $\texttt{AC}_{lj}^{mod}$ is the modality for the value of the quality property j that indicates if the value of that quality property should be minimized or maximized;

- $\texttt{AC}_{l}^{prior}$ is an n-tuple of the client l's priority (i.e., importance) weights $\texttt{AC}_{lj}^{prior}$ given for each of the n quality properties;

- a_l^x is an estimate of the client l's sensitivity to the deception induced by the gap between the expected and delivered levels of quality properties;

- a_l^q is an estimate of the client l's sensitivity to brand image; and

- b_l is the client's outlook, i.e., tendency to be systematically more or less strongly optimistic or pessimistic in feedbacks.

We will assume that the client's outlook (b_l), sensitivity to the deception (a_l^x) and, sensitivity to the brand image (a_l^q) are independent. We further assume that there is no interaction between these parameters of the client profile. We understand the outlook to be a constant, which is positive if optimism prevails, and negative otherwise. That constant is added to an outlook-free feedback level. The idea is that the bias introduced by outlook is completely independent from the client's expectations, the behavior of the provider, and the brand image of the provider.

A generative model of ratings, taking the form of a linear regression model for each client, and accounting for all these features is assumed. More precisely, the uncensored rating, y_{kli} is the feedback $\texttt{f}$ given by a client l after a transaction i with the provider k is assumed to be given by

$$y_{kli} = a_l^q q_k + a_l^x x_{kli} + b_l + \varepsilon_{kli} \tag{6.1}$$

where the random variable ε_{kl} is normal centered and ε_{kli} is a realization of this random variable appearing in transaction number i, $\varepsilon_{kl} \sim N(0, \sigma_l)$. Above, x_{kli} integrates the information about the client's preferences ($\texttt{AC}_{l}^{pref}$) and priorities ($\texttt{AC}_{l}^{prior}$). ε_{kli} makes the feedback vary randomly around the feedback level obtained by accounting for the client's feedback profile, and the difference between the expected and advertised quality levels. Notice that the ratings y_{kli} are not observed; more realistically, only a censored version of the rating is directly observed – see next section.

Finally, since, as for a one-way analysis of variance, the q_k are only defined up to an additive constant, we constrain the q_k parameters to sum to zero, $\sum_{k=1}^{n_x} q_k = 0$ and unit variance [26]. The q_k are therefore standardized.

6.3.2 Accounting for truncation

Ratings are often expressed on a limited scale. The proposed model therefore assumes that the ratings y_{kli} are truncated in order to obtain a final rating z_{kli} in the interval $[-c, +c]$. We assume, without lack of generality, that the ratings are normalized in the interval $[-1, +1]$, and thus $0 \leq c \leq 1$. In other words, only the truncated ratings z_{kli} are observed while the y_{kli} are unobserved. Therefore,

$$z_{kli} = \text{trunc}(y_{kli}, c) \tag{6.2}$$

where trunc is the truncation operator defined as

$$\text{trunc}(y, c) = \delta(y \geq 0) \min(y, c) + \delta(y < 0) \max(y, c) \tag{6.3}$$

The function $\delta(y \geq 0)$ is equal to 1 if the condition $y \geq 0$ is true and 0 otherwise. Thus, the truncation operator saturates the variable y by constraining its range in the interval $[-c, +c]$.

This model thus considers that we directly observe the truncated ratings for the N transactions, $\{z_{kli}\}$, as well as the quality level according to client l quality expectations, $\{x_{kli}\}$. The objective is to estimate (i) the quality of the providers based on these ratings, and (ii) the different biases of the clients/raters.

6.3.3 The complete likelihood function of the model

We now consider the problem of estimating the different parameters of the model, namely $\mathbf{\Theta} = \{q_k, a_l^q, a_l^x, b_l, \sigma_l\}$. For this purpose, the set of observed values is considered as incomplete and a Expectation-Maximization algorithm will be used, as proposed in [57]. The complete set of variables is $\{y_{kli}, z_{kli}\}$, $k = 1 \dots n_p$, $l = 1 \dots n_c$, $i = 1 \dots N$, and since only the $\{z_{kli}\}$, $k = 1 \dots n_p$, $l = 1 \dots n_c$, $i = 1 \dots N$ are observed, all the other variables, $\{y_{kli}\}$, $k = 1 \dots n_p$, $l = 1 \dots n_c$, $i = 1 \dots N$, are considered as unobserved. Assuming independence between the observations and exact knowledge of the x_{kli}, the likelihood for the complete (observed and unobserved) data is

$$\begin{aligned} \mathcal{L}(\mathbf{\Theta}) &= \prod_{k=1}^{n_p} \prod_{l=1}^{n_c} \prod_{i \in (k,l)} \mathrm{P}(y_{kli}, z_{kli} | x_{kli}) && (6.4) \\ &= \prod_{k=1}^{n_p} \prod_{l=1}^{n_c} \prod_{i \in (k,l)} \mathrm{P}(z_{kli} | x_{kli}, y_{kli}) \mathrm{P}(y_{kli} | x_{kli}) && (6.5) \\ &= \prod_{k=1}^{n_p} \prod_{l=1}^{n_c} \prod_{i \in (k,l)} \mathrm{P}(y_{kli} | x_{kli}) && (6.6) \end{aligned}$$

where $\mathbf{\Theta}$ is the vector containing all the parameters of the model, $\mathbf{\Theta} = \{q_k,\ a_l^q,\ a_l^x,\ b_l,\ \sigma_l\}$, and (k, l) denotes the set of transactions involving provider k and client l, with n_{kl} being the total number of transactions $\in (k, l)$. Thus, in the previous equation, the product on i is taken on the set of n_{kl} transactions belonging to (k, l). The last equality is due to the fact that the value of z_{kli} is known exactly if we know the value of y_{kli}. Taking the logarithm of both sides and denoting the log-likelihood by $l = \log \mathcal{L}$ provides

$$\begin{aligned} l(\mathbf{\Theta}) &= \sum_{k=1}^{n_p} \sum_{l=1}^{n_c} \sum_{i \in (k,l)} \log(\mathrm{P}(y_{kli} | x_{kli})) && (6.7) \\ &= -\frac{1}{2} \sum_{k=1}^{n_p} \sum_{l=1}^{n_c} \sum_{i \in (k,l)} \left\{ \left[\frac{y_{kli} - (a_l^q\, q_k + a_l^x\, x_{kli} + b_l)}{\sigma_l} \right]^2 \right. \\ &\quad \left. + \log(\sigma_l^2) + \log(2\pi) \right\} && (6.8) \end{aligned}$$

This complete log-likelihood function serves as basis for the parameters estimation algorithm.

6.3.4 Estimating the parameters

In this section, we develop an expectation-maximization (EM) based algorithm (the so-called "one-step-late" algorithm ([66]; see also [141])) for the estimation of both the parameters of the model and the unobserved variables. As already stated, we assume that we observe only the rating z_{kli} and the quality level according to client l quality expectations, x_{kli}, for each transaction. Notice that the estimates of the parameters of interest are denoted by a hat.

A few notations are needed before stating the reestimation formulas. The standard normal distribution and the standard normal cumulative distribution function are denoted by

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \text{ and } \phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du \qquad (6.9)$$

Let us now turn to the design of the reestimation algorithm.

The expectation step

The expectation step of the EM algorithm aims to compute the expectation of the log-likelihood function (6.8) given the observed variables and the current estimate of the parameters, that is, $\mathrm{E}[l | x_{kli}, z_{kli}, \widehat{\mathbf{\Theta}}]$.

In order to compute these expectations, we make use of a well-known decomposition related to the bias-variance decomposition; see [6] for a similar computation for estimating the parameters of a tobit model with the EM algorithm:

$$\begin{aligned} \mathrm{E}\left[(\alpha y+\beta)^2\right] &= (\alpha \mathrm{E}[y]+\beta)^2 + \alpha^2 \underbrace{\mathrm{E}\left[(y-\mathrm{E}[y])^2\right]}_{\mathrm{V}[y]} && (6.10)\\ &= (\alpha \mathrm{E}[y]+\beta)^2 + \alpha^2 \mathrm{V}[y] && (6.11)\end{aligned}$$

where $\mathrm{V}[y]$ is the variance of the random variable y. The first term consists in replacing the random variable y by its expectation $\mathrm{E}[y]$. This amounts, when computing the expectation of the likelihood in Equation (6.8), to *simply replace the unobserved values of y_{kli} by their conditional expectations*. The second term involves the variance of the random variable, $\mathrm{V}[y]$.

Let us first deal with the estimate of the unobserved variable, denoted as $\widehat{y}_{kli} = \mathrm{E}[y_{kli}|x_{kli}, z_{kli}, \widehat{\mathbf{\Theta}}]$. It can easily be shown (from the standard theory of the tobit model [66; 67]; see [57] for details) that the estimate of the unobserved variable y_{kli} for transaction i depends on the three possible truncation cases: $z_{kli} = -c$, $-c < z_{kli} < +c$ and $z_{kli} = +c$:

First case: $z_{kli} = -c$:

$$\widehat{y}_{kli} = (\widehat{a}_l^q\,\widehat{q}_k + \widehat{a}_l^x\,\widehat{x}_{kli} + \widehat{b}_l) + \widehat{\sigma}_l^y \lambda(\widehat{\gamma}_{kli}) \quad (6.12)$$

$$\text{with} \quad \begin{cases} \widehat{\gamma}_{kli} = \dfrac{-c - (\widehat{a}_l^q\,\widehat{q}_k + \widehat{a}_l^x\,\widehat{x}_{kli} + \widehat{b}_l)}{\widehat{\sigma}_l^y} \\ \lambda(\widehat{\gamma}_{kli}) = -\dfrac{\varphi(\widehat{\gamma}_{kli})}{\phi(\widehat{\gamma}_{kli})} \end{cases} \quad (6.13)$$

Second case: $-c < z_{kli} < +c$:

$$\widehat{y}_{kli} = z_{kli} \quad (6.14)$$

Third case: $z_{kli} = +c$:

$$\widehat{y}_{kli} = (\widehat{a}_l^q\,\widehat{q}_k + \widehat{a}_l^x\,\widehat{x}_{kli} + \widehat{b}_l) + \widehat{\sigma}_l^y \lambda(\widehat{\gamma}_{kli}) \quad (6.15)$$

$$\text{with} \quad \begin{cases} \widehat{\gamma}_{kli} = \dfrac{c - (\widehat{a}_l^q\,\widehat{q}_k + \widehat{a}_l^x\,\widehat{x}_{kli} + \widehat{b}_l)}{\widehat{\sigma}_l^y} \\ \lambda(\widehat{\gamma}_{kli}) = \dfrac{\varphi(\widehat{\gamma}_{kli})}{1 - \phi(\widehat{\gamma}_{kli})} \end{cases} \quad (6.16)$$

Second, the variance term in (6.11) will simply be neglected. Indeed, it has been observed in [57] that this term (i) adds considerable complexity to the updating rules, and (ii) does not improve significantly the performances of the model. However, the interested reader could easily adapt the EM algorithm from [57] in oder to compute the exact expectation. Consequently, the expected log-likelihood is simply approximated by

$$\begin{aligned} \mathrm{E}[l|x_{kli}, z_{kli}, \widehat{\mathbf{\Theta}}] \approx -\frac{1}{2}\sum_{k=1}^{n_p}\sum_{l=1}^{n_c}\sum_{i\in(k,l)} \Bigg\{ &\left[\frac{\widehat{y}_{kli} - (a_l^q\,q_k + a_l^x\,x_{kli} + b_l)}{\sigma_l}\right]^2 \\ &+ \log(\sigma_l^2) + \log(2\pi)\Bigg\} \end{aligned} \quad (6.17)$$

that is, the log-likelihood where we substituted y_{kli} by $\widehat{y}_{kli}$.

The maximization step

For the parameters associated to the providers, when maximizing Equation (6.17) (M-step) by computing the partial derivatives with respect to the different parameters and setting the result equal to zero, we

easily obtain, for the provider-related parameters,

$$q_k = \frac{\sum_{l=1}^{n_c} \sum_{i \in (k,l)} \left(a_l^q \left(\widehat{y}_{kli} - a_l^x x_{kli} - b_l\right)\right) / \sigma_l^2}{\sum_{l=1}^{n_c} \left(n_{kl} a_l^{q2}\right) / \sigma_l^2} \tag{6.18}$$

For the parameters associated to the clients,

$$a_l^q = \frac{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} q_k \left(\widehat{y}_{kli} - a_l^x x_{kli} - b_l\right)}{\sum_{k=1}^{n_p} n_{kl} q_k^2} \tag{6.19}$$

$$a_l^x = \frac{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} x_{kli} \left(\widehat{y}_{kli} - a_l^q q_k - b_l\right)}{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} x_{kli}^2} \tag{6.20}$$

$$b_l = \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i \in (k,l)} \left(\widehat{y}_{kli} - a_l^q q_k - a_l^x x_{kli}\right) \tag{6.21}$$

$$\sigma_l^2 = \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i \in (k,l)} \left(\widehat{y}_{kli} - a_l^q q_k - a_l^x x_{kli} - b_l\right)^2 \tag{6.22}$$

The one-step-late EM-based algorithm [66; 141] aims to replace the parameters in the right-hand side of the equations by their current estimate. This leads to the following updating rules

$$\widehat{q}_k \leftarrow \frac{\sum_{l=1}^{n_c} \sum_{i \in (k,l)} \left(\widehat{a}_l^q \left(\widehat{y}_{kli} - \widehat{a}_l^x x_{kli} - \widehat{b}_l\right)\right) / \widehat{\sigma}_l^2}{\sum_{l=1}^{n_c} \left(n_{kl} \widehat{a}_l^{q2}\right) / \widehat{\sigma}_l^2} \tag{6.23}$$

$$\widehat{a}_l^q \leftarrow \frac{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} \widehat{q}_k \left(\widehat{y}_{kli} - \widehat{a}_l^x x_{kli} - \widehat{b}_l\right)}{\sum_{k=1}^{n_p} n_{kl} \widehat{q}_k^2} \tag{6.24}$$

$$\widehat{a}_l^x \leftarrow \frac{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} x_{kli} \left(\widehat{y}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{b}_l\right)}{\sum_{k=1}^{n_p} \sum_{i \in (k,l)} x_{kli}^2} \tag{6.25}$$

$$\widehat{b}_l \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i \in (k,l)} \left(\widehat{y}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{a}_l^x x_{kli}\right) \tag{6.26}$$

$$\widehat{\sigma}_l^2 \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i \in (k,l)} \left(\widehat{y}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{a}_l^x x_{kli} - \widehat{b}_l\right)^2 \tag{6.27}$$

The updating rules (6.23)-(6.27) are iterated together with the equations providing the expected ratings $\widehat{y}_{kli}$ (Equations (6.12)-(6.14)) until convergence. Initially, the parameters are set to $\widehat{q}_k = \text{mean}_{li}(z_{kli})$, $\widehat{a}_l^q = 0$, $\widehat{a}_l^x = 1$, $\widehat{b}_l = 0$ and $\widehat{\sigma}_l = 0.1$.

6.4 Experiments

The experimental section answers the following three research questions:

1. Is the Feedback Profile model able to estimate the parameters of the client profile in an accurate way?

2. Do the suggested model compare favorably in terms of residuals with respect to similar models (Simple Average, Brockhoff [26] and Iterative Refinement [121])?

3. What are the applications of the Feedback Profile model and does the model compete with similar methods in these applications?

In order to investigate these questions, we performed the experiments described in this section.

6.4.1 Experimental setup

The first experiments simulate a simple provider-client interaction model. Remember that transaction is the execution of a requested service by a provider for a client. The realization of a transaction leads the client to give a rating (i.e., feedback) to the provider. In accordance with previous notations, a provider k is characterized by his brand image q_k. A client l is characterized by his sensitivity to the image of the provider a_l^q, his sensitivity to the deception induced by the gap between expected and delivered quality level as defined according to its expectations a_l^x and his outlook b_l and his stability in providing constant ratings for a fixed level of quality σ_l. A transaction i occurring between a provider k and a client l has a quality level in accordance to l's expectations defined by x_{kli} and a rating of y_{kli}. n_k is the total number of providers, n_l is the total number of clients. In the following experiments, parameters will be randomly generated in the following intervals according to a uniform distribution:

$$\begin{cases} a_l^q \in [0,1] \\ a_l^x \in [0,1] \\ b_l \in [-0.5, 0.5] \\ q_k \in [-1,1] \\ x_{kli} \in [-1,1] \end{cases}$$

6.4.2 Predictability of a Feedback Profile

We outline here the accuracy with which the Feedback Profile model estimates feedback profiles. We generate clients and providers, and the ratings for transactions. We then attempt to determine clients and providers profiles with these ratings and compare them to the generated profiles.

6.4.2.1 Estimation of clients' and providers' parameters

In this first, preliminary experiment, we illustrate the estimation power of our Feedback Profile model. We first generate clients and providers profile parameters and try to estimate them with our model. We then compare the estimation results with generated values.

Parameters are generated with values given in Subsection 6.4.1. Once the parameters are generated, the ratings given by the client l to the provider k for the transaction i (y_{kli}) are computed according to the Equations (6.1) and (6.2) of the model. x_{kli} and z_{kli} values are put in our Feedback Profile model that allows to estimate the providers and the clients profiles. For this experiment, the data set contains 100 providers and 100 clients and a maximum of 100 transactions for each couple of provider-client. The model updates the parameters associated to the providers and the clients at each iteration until convergence. Figures 6.3, 6.4, 6.5 and 6.6 respectively represent the results get for the parameters q_k, a_l^q,a_l^x, and b_l. For each parameter, we illustrate the value initially generated and the result of its estimation. Figures 6.3, 6.4, 6.5 and 6.6 highlight the accuracy of the Feedback Profile model as predicted parameters are close from their initial values. The continuous line is the baseline (generated parameter = estimated parameter) and marks are the results of the experiment. The closeness of our results with the baseline outlines that parameters are estimated in an accurate way.

6.4.2.2 Predictability of clients profiles

This second experiment outlines the prediction of client profiles. To this aim, we generate clusters of client profiles and observe how the model can estimate them. We initially generate four different clusters of client profiles with the following characteristics:

1. *Brand image sensitive clients* with $a_l^q \sim N(0.6, 0.1)$, $a_l^x \sim N(0.05, 0.1)$, $b_l \sim N(0.1, 0.1)$;
2. *Deception sensitive clients* with $a_l^q \sim N(0.05, 0.1)$, $a_l^x \sim N(0.6, 0.1)$, $b_l \sim N(0.1, 0.1)$;
3. *Optimistic clients* with $a_l^q \sim N(0.1, 0.1)$, $a_l^x \sim N(0.1, 0.1)$, $b_l \sim N(0.7, 0.1)$;

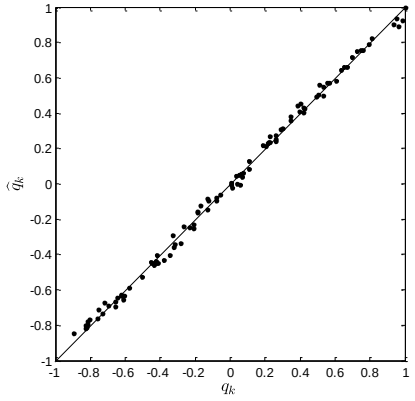


Figure 6.3: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of q_k .

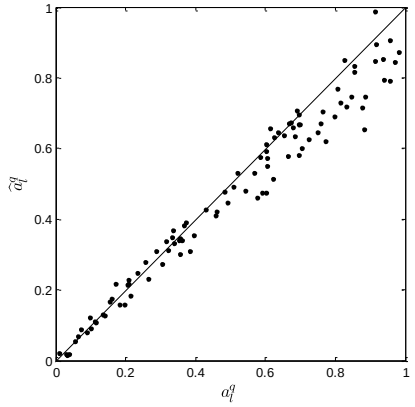


Figure 6.4: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of a_l^q .

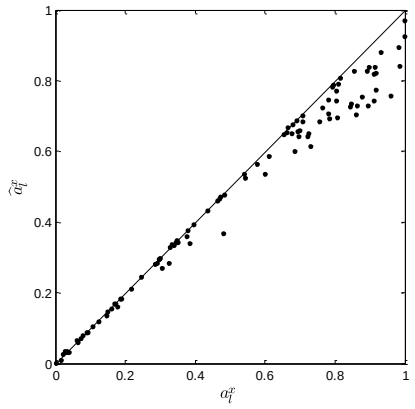


Figure 6.5: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of a_l^x .

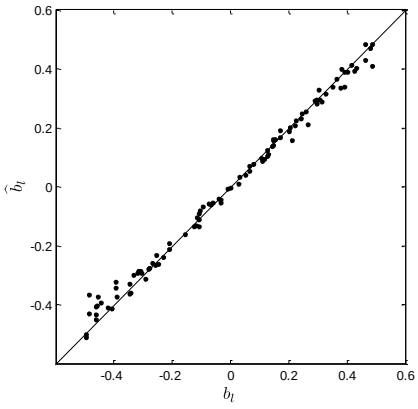


Figure 6.6: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of b_l .

4. *Pessimistic clients* with $a_l^q \sim N(0.1, 0.1)$, $a_l^x \sim N(0.1, 0.1)$, $b_l \sim N(-0.5, 0.1)$.

The parameters q_k and x_{kli} are generated $\in [-1, 1]$. Figure 6.7 illustrates the generated clusters (left) and Figure 6.8 illustrates the predicted clusters (right). In Figure 6.7 and 6.8: the brand image sensitive clients are tagged with '•'; the deception sensitive clients are tagged with '○'; the optimistic clients are tagged with '∗'; and, the pessimistic clients are tagged with 'x'.

We clearly observe that predicted clusters are close from those generated.

6.4.3 Studies of the residuals

Experiments below illustrate the residuals obtained with the Feedback Profile method in comparison with residuals obtained with similar methods.

6.4.3.1 Ratings estimation residuals

In this experiment, we study the ratings residuals induced by the estimation of the parameters. The ratings residual is the result of: $\sum_{k,l,i} |y_{kli} - \widehat{y}_{kli}|$ where $\widehat{z}_{kli} = trunc(\widehat{y}_{kli}) = \widehat{a}_l^q \widehat{q}_k + \widehat{a}_l^x x_{kli} + \widehat{b}_l$. We compare the results of the Feedback Profile with those get with the method introduced by Brockhoff in [26]. The Brockhoff method estimates ratings with the following formula: $\widehat{z}_{kli} = a_l q_k + b_l$ (the model

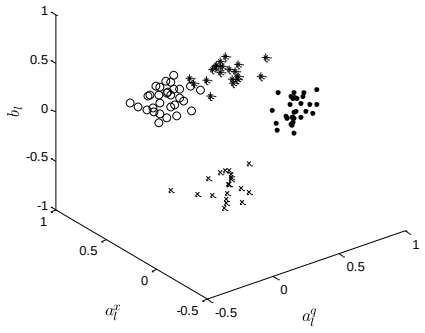


Figure 6.7: Real (generated) clusters of client profiles.

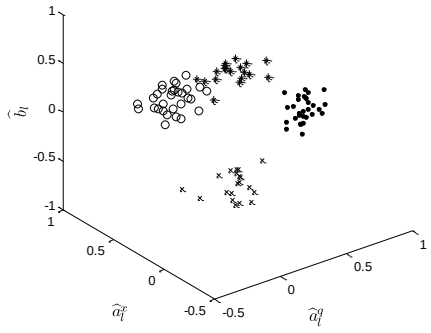


Figure 6.8: Predicted (estimated) clusters of client profiles.

does not account for truncation). The estimation of parameters of the Brockhoff model [26] is given by:

$$\widehat{q}_k \leftarrow \frac{\sum_{l=1}^{n_c} \sum_{i\in(k,l)} \left(\widehat{a}_l^q \left(\widehat{z}_{kli} - \widehat{b}_l\right)\right) / \widehat{\sigma}_l^2}{\sum_{l=1}^{n_c} \left(n_{kl} \widehat{a}_l^{q2}\right) / \widehat{\sigma}_l^2} \tag{6.28}$$

$$\widehat{a}_l^q \leftarrow \frac{\sum_{k=1}^{n_p} \sum_{i\in(k,l)} \widehat{q}_k \left(\widehat{z}_{kli} - \widehat{b}_l\right)}{\sum_{k=1}^{n_p} n_{kl} \widehat{q}_k^2} \tag{6.29}$$

$$\widehat{b}_l \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i\in(k,l)} \left(\widehat{z}_{kli} - \widehat{a}_l^q \widehat{q}_k\right) \tag{6.30}$$

$$\widehat{\sigma}_l^2 \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_p} \sum_{i\in(k,l)} \left(\widehat{z}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{b}_l\right)^2 \tag{6.31}$$

As before, the generated data set contains 100 providers and 100 clients and a maximum of 100 transactions for each provider-client pair. Parameters are generated with values given in Subsection 6.4.1.

The results obtained for 100 runs of both methods are illustrated in Table 6.1. Table 6.1 shows the average ratings residual for the Feedback Profile and the Brockhoff methods. These results highlight that the Feedback Profile method outperforms the Brockhoff method as the average ratings residual is smaller.

	Feedback Profile	Brockhoff
Average ratings residual	0.0436	0.1352

Table 6.1: Comparison between average residuals get with the Feedback Profile model and the Brockhoff model.

6.4.3.2 Residuals for brand image estimation

This experiment estimates the residuals of the parameter q_k (i.e., the brand image of the provider k) obtained via the Feedback Profile method and other comparable methods. The brand image residual is the result of: $\sum_k |q_k - \widehat{q}_k|$.

Once more, the data set contains 100 providers and 100 clients and a maximum of 100 transactions for each couple of provider-client. Parameters are generated with values given in Subsection 6.4.1.

We compare our results with those of comparable methods: the Iterative Refinement model (IR) [121] ; the Brockhoff model [26]; and a Simple Additive (SA) method.

The estimation of parameters for the IR model [121] is provided by:

$$\widehat{q}_k \leftarrow \frac{\sum_{l=1}^{n_c} \sum_{i\in(k,l)} \widehat{z}_{kli}}{\sum_{i\in(k,l)} n_{kl}\widehat{w}_l} \tag{6.32}$$

$$\widehat{V}_l \leftarrow \frac{\sum_{k=1}^{n_p} \sum_{i\in(k,l)} \left(\widehat{z}_{kli} - \widehat{q}_k\right)^2}{n_{\bullet l}} \tag{6.33}$$

$$\widehat{w}_l \leftarrow \frac{1}{\widehat{V}_l^{\beta}} \tag{6.34}$$

The estimation of q_k for the SA method is given by:

$$\widehat{q}_k \leftarrow \frac{1}{n_p} \sum_{l=1}^{n_c} \sum_{i\in(k,l)} \widehat{z}_{kli} \tag{6.35}$$

Table 6.2 respectively represent the results on 100 runs get for our Feedback Profile model, the IR model, the Brockhoff model, and the SA model. Figures 6.9, 6.10, 6.11 and 6.12 illustrate the q_k parameters

	Feedback Profile	IR	Brockhoff	SA
Average brand images residual	0.0135	0.0286	0.0321	0.1301

Table 6.2: Comparison between average brand images residual get with the Feedback Profile model, the Brockhoff model, the IR model and the SA method.

initially generated and the $\widehat{q}_k$ estimated by, respectively, our Feedback Profile model, the IR model, the Brockhoff model and the SA model. Results of Table 6.2 and Figures 6.9, 6.10, 6.11 and 6.12 outline that the Feedback Profile model produces smaller brand image residuals than similar methods.

6.4.4 Applications

We present here some applications of our Feedback Profile model. We use the model to detect the collusion that can exist between particular clients and providers. We also apply the model to real data about movie ratings and predict ratings (i.e., users' feedbacks).

6.4.4.1 Collusion detection

The collusion between providers and clients acts towards manipulating client perceptions by posting ratings that praise the provider products [44]. Similarly, some clients can systematically decrease the ratings given to a particular client to lower its overall score. The Feedback Profile model allows to detect such collusion that can occur between providers and clients. In order to detect collusion, we introduce a new parameter: c_{kl} which is the collusion level occurring between a provider k and a client l. We outline the collusion with the Feedback Profile model and with the Brockhoff model. For this experiment, the data set contains 100 providers and 100 clients and a maximum of 100 transactions for each couple of provider-client. Parameters are generated with values given in Subsection 6.4.1.

For simulating collusion, we add a systematic cheating between the 40th provider and the 3rd client: the ratings given to the 40th provider by the 3rd client are systematically increased by 1: $\widehat{y}_{40,3,i} \leftarrow \widehat{y}_{40,3,i} + 1$.

Once parameters of the models have been estimated, the collusion of all couples of provider-client are computed. The collusion formula of the Feedback Profile model is given by

$$c_{kl} = \frac{\sum_{i\in(k,l)} (\widehat{y}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{a}_l^x x_{kli} - \widehat{b}_l)}{n_{kl}} \tag{6.36}$$

The collusion formula of the Brockhoff model is given by

$$c_{kl} = \frac{\sum_{i\in(k,l)} (\widehat{z}_{kli} - \widehat{a}_l^q \widehat{q}_k - \widehat{b}_l)}{n_{kl}} \tag{6.37}$$

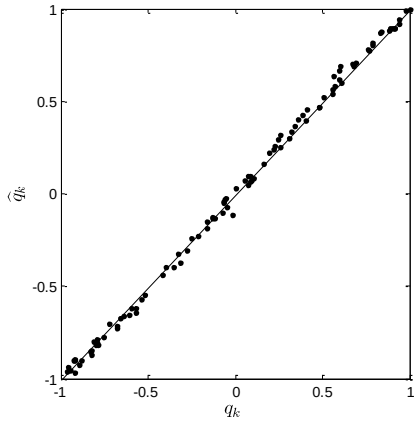


Figure 6.9: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of q_k for the Feedback Profile model.

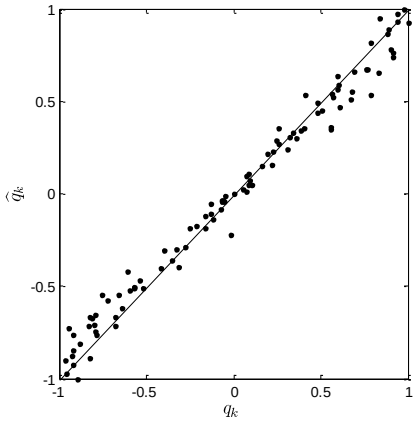


Figure 6.10: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of q_k for the IR model.

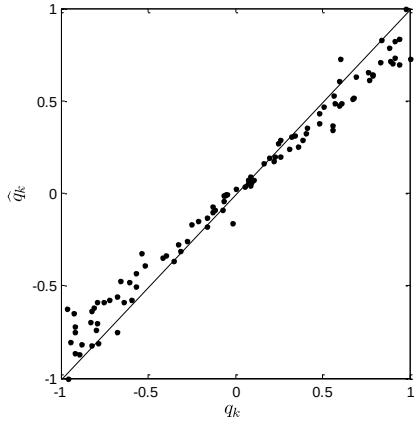


Figure 6.11: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of q_k for the Brockhoff model.

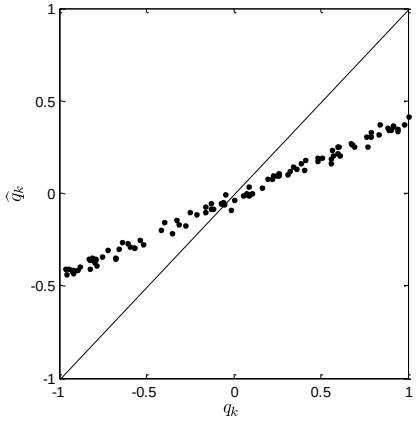


Figure 6.12: Comparison between the real (generated, x-axis) and the predicted (estimated, y-axis) value of q_k for the SA method.

Figures 6.13 and6.14 illustrate the collusion value for all providers for the client 3. In both figures, the collusion is significant for the 40th provider.

The Feedback Profile model allows a more accurate detection of collusion than the Brockhoff model. The collusion values are restricted in the interval [-0.2,0.2] in the Feedback Profile model and in the interval [-0.8,-0.2] in the Brockhoff model. The collusion between provider 40 and user 3 is detected with a 1.1 value in the Feedback Profile model and a 0.6 value for the Brockhoff model. The Feedback Profile model better detects collusions with small values than the Brockhoff model because the collusion values dissipate more in the Brockhoff model .

Figures 6.15 an 6.16 illustrate the collusion detection if the ratings given to the 40th provider by the 3rd client are increased only by 0.3. The Feedback Profile model locates easily the collusion while the Brockhoff model is unable to detect the collusion, i.e., $c_{3,40}$ is not more significant than the collusion between other couples of provider-client. These results outline that our model is more appropriate in this case than the Brockhoff model.

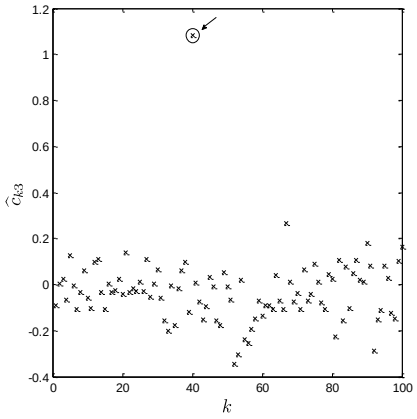


Figure 6.13: Collusion corresponding to a rating bias of 1.0 detected with the Feedback Profile model.

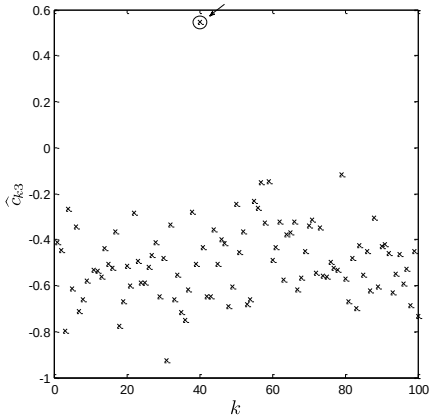


Figure 6.14: Collusion corresponding to a rating bias of 1.0 detected with the Brockhoff model.

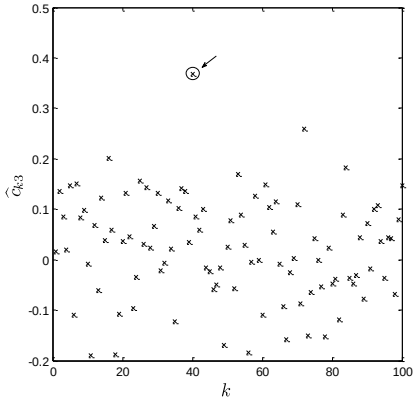


Figure 6.15: Collusion corresponding to a rating bias of 0.3 detected with the Feedback Profile model.

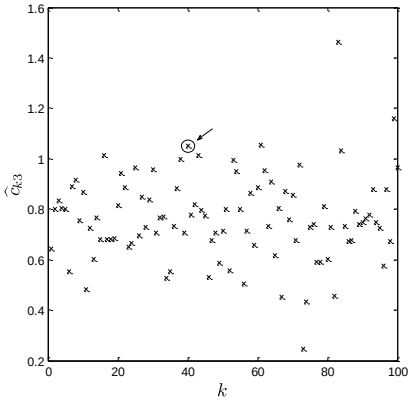


Figure 6.16: Collusion corresponding to a rating bias of 0.3 detected with the Brockhoff model.

6.4.4.2 Rating prediction from real data

MovieLens dataset. The Feedback Profile model can also be used to predict the ratings (i.e., feedback) that the clients will give to providers after a transaction. We can therefore predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users [183].

We made experiments on the MovieLens dataset[1] to predict movie ratings with our Feedback Profile model. We use the dataset with 100.000 ratings over 943 users and 1682 movies. A rating is a value taken from the set $\{1, 2, 3, 4, 5\}$. To add the quality level information (x_{kli}), we define the quality of a movie simply as the mean of the ratings for this movie.

To adapt the dataset to our Feedback Profile model, we consider here that the movies are the providers and the users are the clients. There is at most one transaction between a client and a provider, i.e., each user rates a movie at most once.

We compare the Feedback Profile (FP) model with other comparable methods: the Feedback Profile model without quality information (FP\x); the Brockhoff model [26] (B); some average methods, and; some nearest neighbors methods. The average methods are: a simple average method where a rating

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

estimation is given by the mean of all existing ratings on the dataset (Average); an average by provider method where a rating estimation for a provider is given by the mean of all existing ratings for this provider (Av. P), and; an average by client where a rating estimation of a client is given by the mean of all existing ratings given by this client (Av. C). The nearest neighbors methods are based on a weighted sum of movie similarity computation [183]. The similarity computation is performed with three different methods: the Pearson correlation similarity (Corr.); the adjusted cosine similarity (Adj. Cos.), and; the cosine based similarity (Cos) [183].

We test these different methods on the dataset with a ten-fold cross validation: for 10 different runs, we take a training set of 90% to adjust parameters and we generate the 10% remaining ratings and compare them to the test set. For each method, we then observe its Mean Absolute Error (MAE). The MAE is a measure of the deviation of predictions from their true user-specified values [183]. The results are given in the first column of Table 6.3.

	MAE for the prediction methods without truncation	MAE for the prediction methods with truncation
Feedback Profile (FP)	0.7212	0.6824
Feedback Profile without x (FP\x)	0.7310	0.6908
Brockhoff (B)	0.7302	0.6900
Average (Average)	0.9448	0.8942
Average by provider (Av. P)	0.8154	0,7842
Average by client (Av. C)	0.8346	0.8018
Pearson correlation (Corr.)	0.8162	0.7864
Adjusted Cosine (Adj. Cos.)	0.7526	0.7256
Cosine (Cos)	0.8124	0.7868

Table 6.3: Performances of the different models on the MovieLens dataset, without and with truncation.

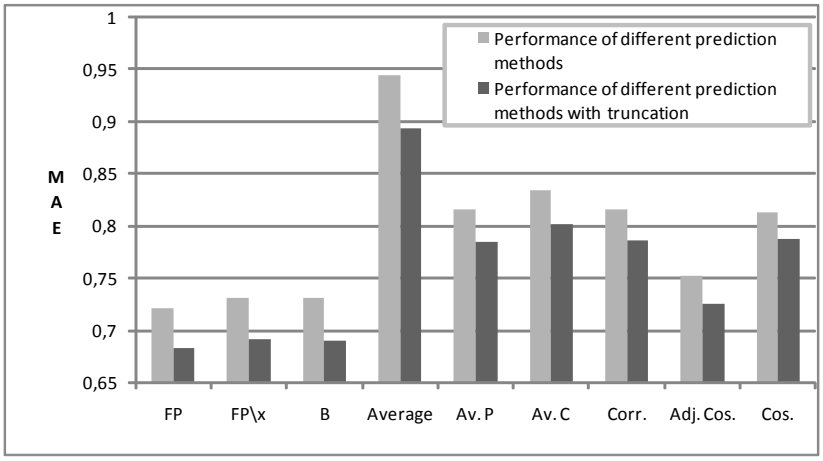


Figure 6.17: Performances of the different models, on the MovieLens dataset, without (gray bar) and with (black bar) truncation.

To reduce the MAE, we truncate the predicted rating to the nearest allowable integer value. Indeed, ratings are specified by users in the set $\{1, 2, 3, 4, 5\}$. Predicted ratings are in interval $[1, 5]$. To reduce the error induced by continual values in predicted ratings, we map them to their nearest integer in the set $\{1, 2, 3, 4, 5\}$. E.g., a predicted rating of 3.8 will be truncated to 4. The Table 6.3 shows the results of this truncation on the different methods. Experimental results are illustrated in Figure 6.17. It can be observed that the Feedback Profile method (FP) shows better performances than all the other investigated methods (FP\x, B, Average, Av. P, Av. C, Cos, Adj. Cos., Corr) with and without truncation. Figure 6.17 outlines that the truncation of predicted ratings improves results for all tested methods.

Students dataset. We also use the Feedback Profile model to predict the ratings given by professors to students about the achievement of their courses. For this purpose, the grades of 99 students (second-

year bachelor students at the University of Louvain) were collected for their 12 courses. A rating is the value given by a professor to a student within the interval $[0, 20]$. In this experiment, the students are the providers and the professors are the clients. There is at most one transaction between a client and a provider, each professor rates a student once. To add quality level information (x_{kli}), we define the quality of a student as the mean of the ratings for this student.

We compare the Feedback Profile (FP) model with the same methods than for the MovieLens dataset. We test these different methods on the dataset with a ten-fold cross validation. The MAE results without and with truncation are given in Table 6.4. The truncation is made by mapping predicted ratings to their nearest integer value in the interval $[0, 20]$.

	MAE for the prediction methods without truncation	MAE for the prediction methods with truncation
Feedback Profile (FP)	2.035	2.015
Feedback Profile without x (FP\x)	1.901	1.906
Brockhoff (B)	2.029	2.020
Average (Average)	2.612	2.667
Average by provider (Av. P)	2.239	2.206
Average by client (Av. C)	2.397	2.408
Pearson correlation (Corr.)	2.304	2.300
Adjusted Cosine (Adj. Cos.)	1.988	1.971
Cosine (Cos)	2.401	2.406

Table 6.4: Performances of the different models on the students dataset, without and with truncation.

It can be observed that, for the students dataset, the Feedback Profile (FP) model shows better performance than most other investigated methods (B, Average, Av. P, Av. C, Cos, Corr) and is competitive with results provided by remaining methods (FP\x,Adj. Cos.).

6.4.5 Discussion of the results

With the MovieLens dataset used for the prediction, we conclude that adding information about quality improves the predictability of ratings. Moreover, the Feedback Profile model is the only one tested which allows us to define a client profile through different parameters: its outlook (b_l), its sensitivity to deception (a_l^x) and its sensitivity to brand image (a_l^q). We are thus able to specify the profiles of users of the dataset. Figures 6.18, 6.19, 6.20 and 6.21 illustrate such profiles with the MovieLens dataset. The first figure (top left - Figure 6.18) shows the profiles through all 3 dimensions: b_l, a_l^x, a_l^q. The second figure (top right - Figure 6.19) illustrates the clients' profiles through dimensions a_l^x and b_l. The third figure (down left - Figure 6.20) outlines the user profile through dimensions a_l^q and b_l. The last figure (down right - Figure 6.21) represents the dimensions a_l^x and a_l^q.

We can notice on these figures that some users adopt an unusual behavior (outliers) with parameter values far from most users. We looked at their ratings and concluded that such these act weirdly. Indeed, some of them give good ratings to movies with a low rating average and some others rate good movies with bad scores. The Feedback Profile model is thus able to detect users with a strange behavior.

To evaluate the efficiency of the Feedback Profile model for the prediction of movie ratings, we choose to set x_{kli} to the mean of ratings for each movie. Notice that features related to movies (such as actors, producers, etc.), could be used as well. This is left to further work.

Figures 6.22, 6.23 and 6.24 illustrate Feedback Profiles get with the students dataset. The first figure (top left - Figure 6.22) illustrate the professors' sensitivities to students' brand image. The second figure (top right - Figure 6.23) illustrates the professors' sensitivities to deception. The last figure (down right - Figure 6.20) represents the professors' outlook.

Similarly to the MovieLens dataset, we choose to set x_{kli} to the mean or ratings for each student. However, other features related to students quality level could be used as well.

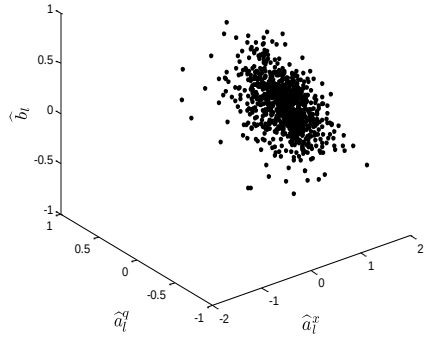


Figure 6.18: Users profiles represented in dimensions b_l, a_l^x and a_l^q.

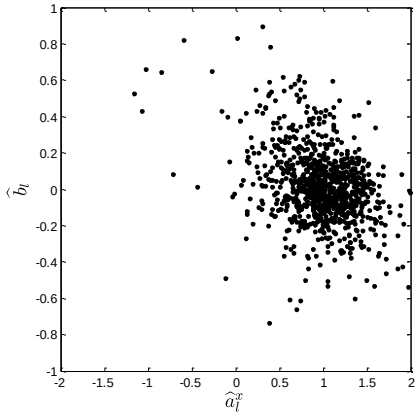


Figure 6.19: Users profiles represented in dimensions a_l^x and b_l.

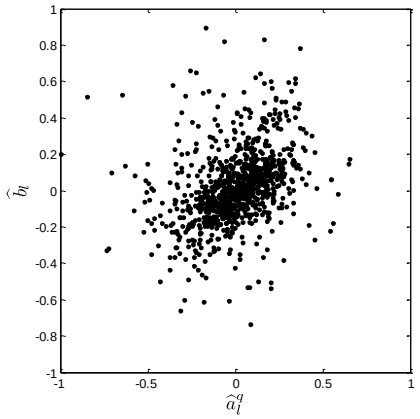


Figure 6.20: Users profiles represented in dimensions a_l^q and b_l.

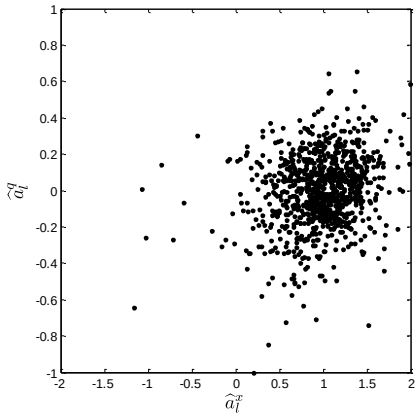


Figure 6.21: Users profiles represented in dimensions a_l^x and a_l^q.

6.5 Related Work

As Dellarocas [42] observes, mechanisms for computing reputation have emerged as an important means for risk management in online communities where parties can change partners from one transaction to the next. When MAS involve self-interested agents, there are incentives for client agents to provide dishonest (also, unfair) feedback if this contributes to the realization of their personal goals. This amounts to intentional, strategic manipulation of feedbacks, which leads to unreliable reputation scores. The response of choice is to design reputation mechanisms that motivate the provision of honest feedback. Such mechanisms can control who participates in the MAS, the kind of feedback information solicited from participants, how is such information aggregated, and how it is presented to the clients [43]. In settings involving moral hazard clients face the risk of providers reneging on advertised quality levels. Dellarocas [43] studied reputation mechanisms for such cases. Dellarocas and Wood [45] study reporting bias, which distorts the distribution of public feedback relative to the underlying distribution of private (hidden) feedback, where it is critical to understand how to interpret the decision of a party not to post feedback. Poston [171] outlines the potential sources of such biases. Biases sources are classified into different types according to their characteristics. Biases can be involved by error measurement; by personal preferences (unintentional bias) or by; intentional feedback, (manipulated feedback). To motivate honest feedback, Jurca and Faltings [98] suggest a mechanism, which elicits feedback from the client only when the provider claims good service; before asking for feedback from the client, the mechanism allows the provider to

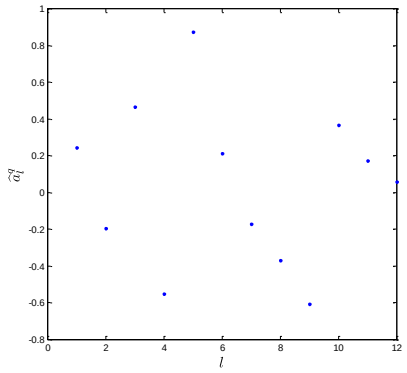


Figure 6.22: Professors' sensitivities to students' brand image.

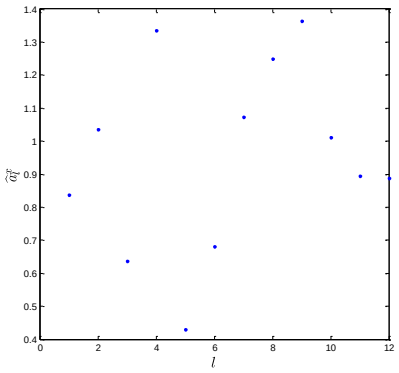


Figure 6.23: Professors' sensitivities to deception.

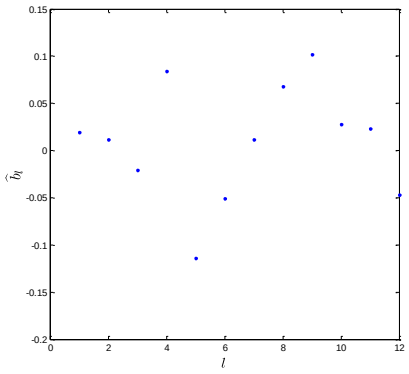


Figure 6.24: Professors' outlook.

acknowledge failure and reimburse the client. If the provider claims good service and the client disagrees, both are sanctioned: the provider suffers from a negative report, while the client pays a small fine.

Our profiling method complements incentive-based reputation mechanisms. Such mechanisms improve the honesty of the user's feedback $\mathtt{f}$, allowing us to blame the remaining difference between $\mathtt{f}$ and $\bar{\mathtt{f}}$ not on intentional manipulation, but mainly on the user's feedback profile. In absence of an incentive mechanism that favors the provision of honest $\mathtt{f}$s, our profiling method will of course still estimate (and pick up changes in) feedback profiles. What changes compared to the honest $\mathtt{f}$s is the interpretation of the feedback profile. Part of the estimated outlook and both sensitivities will in fact be due to the intentional strategic manipulation. Our profiling method is able to distinguish the part of the difference between $\mathtt{f}$ and $\bar{\mathtt{f}}$ arising from strategic manipulation from the part of that difference, which arises from unintentional outlook and sensitivity characteristics. The feedback profile model allows to detect the strategic manipulation that can occur while feedbacks are given. Ratings manipulations are observed by the feedback profile model with the collusion that can exist between some providers and clients.

In contrast to the incentive-based approach, Yu and Singh [224; 225] elicit reputation information by traversing the social network involving the provider's neighbors. They understand the difference between true feedback and observed feedback as evidence for deception. They deal with deception by reducing the effect of deceptive witnesses on the aggregate reputation of a provider. Our approach is more cautious, in the sense that the feedback profile we estimate is not intended to discredit feedbacks, but to adjust them for the effect of outlook and sensitivities. To capture the trustworthiness of a provider, Park et al. [167] uses two vectors to define how optimistic a peer is. The first is the sensitiveness of the client to the trust vector and the second is its sensitiveness to the penalty vector. The ratio over both vectors defines the optimism of the client to trust and penalty values. This ratio concept is similar

to our outlook value, according to the rater of the service, providers will be systematically optimist or pessimist. However, our approach takes into account other attributes of the client profile: sensitivities to deception and to brand image. We allow a more precise conceptualization of the client behavior effect in a rating. In dealing with inaccurate feedback Teacy et al. [201] assume that the providers of opinion act consistently. We do not make this assumption, as our method continually adjusts the computed feedback profile. In more general terms, two additional observations are important. Firstly, our approach takes a clear position with regards to incentive-based reputation mechanisms intended to favor feedback honesty. This is not the case with non-incentive based approaches, so that it is unclear how they fit within MAS that implement incentive-based reputation mechanisms for honest feedback. Secondly, our method more thoroughly studies the divergence between (honest) observed feedback and reliable feedback than those from Yu and Singh [224; 225], and Teacy et al. [201]. More precisely, we explain this difference via *three* parameters – outlook, sensitivity to deception and sensitivity to brand image – that have intuitive interpretations. By doing so, we illustrate that additional insight can be gained about user's feedback, and be used to inform the research and practice of reputation mechanism design.

6.6 Conclusions

Reputation scores influence cooperation, because they help to build trust among agents. It is essential that reputation scores be reliable, i.e., be computed only on the basis of true past differences between a provider's advertised quality levels before the transaction and the delivered (i.e., observed) quality levels after the transaction. In other words, reliable reputation scores aggregate client's feedback, which reflects only this difference at each transaction. In reality feedback, and therefore the reputation score of a provider agent, depends on more than the gap between advertised and delivered quality levels. In response, incentive-based reputation mechanisms have been suggested to improve the honesty of feedbacks, so as to avoid strategic manipulation of reputation scores. We argued in this chapter that honest feedback does not equate to "true" feedback because honesty does not change the fact that feedback is based on a subjective experience of the gap between the expected and delivered quality levels. We suggested that the departure of observed honest feedback from true feedback is due to the user's feedback profile. A feedback profile characterizes a user on three orthogonal dimensions: outlook; sensitivity to deception, and; sensitivity to brand image. Outlook describes a user's degree of optimism or pessimism in feedbacks. Sensitivity to deception characterizes a user's effect of the gap between the expected and the delivered quality level. Sensitivity to brand image defines how the user reacts to the image of the provider. We proposed a two-step method for computing users' feedback profiles and tracking their change over transactions. In the first step, user's announced preferences and priorities are processed to allow true feedback to be computed. In the second step, true feedback is computed, user's feedback is elicited, and the estimate of the user's feedback profile is updated. We reported the results of experiments on hypothetical data to illustrate the performance of our method in approximating users' true feedback profiles. These experiments highlight the ability of the model to accurately estimate feedback profiles. We also presented how the feedback profile model can be used for collusion detection and the prediction of ratings from a real dataset. Results illustrate that the feedback profile model is more efficient than similar methods for these applications.

Part III

SLA Management

Outline of Part III

The third Part of this thesis presents contributions related to the management of Service Level Agreements. Part II has presented models related to the web service management issue. This part tackles a specific issue of web services management: the management of SLAs. SLAs are critical to the management of web services as they define contracts between stakeholders of the services executions. Such contracts ensure the achievement of services from both functional and non functional considerations. More specifically, SLAs define the quality level to be achieved during the service execution. To manage SLAs, it is then essential to monitor quality information to ensure the conformance between the contract defined and the quality level provided. The models proposed in this Part propose to improve the services management through the enhancement of SLAs management. The proposed models are detailed here:

Chapter 7 proposes a normative management of service level agreements defined between web services. Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. SLAs define these obligations, including for instance the expected service levels to be delivered by the provider, and the payment expected from the client. The obligations of the parties must be made explicit prior to the transaction, and a mechanism should be available to supervise the interaction, in order to ensure that the obligations are met. We propose in this Chapter an architecture to that aim. More precisely, we outline a norm-oriented multiagent system (NoMAS) architecture that is combined with the service-oriented architecture in order to support the definition, management, and control of SLAs between the service clients and service providers.

Chapter 8 proposes to use context to manage SLA during service executions. Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. SLAs define the service users' Quality of Service (QoS) requirements that the service provider should satisfy. Requirements defined once may not be satisfiable when the context of the web services changes (e.g., when requirements or resource availability changes). Changes in the context can make SLAs obsolete, making SLA revision necessary. We propose a method to autonomously monitor the services' context, and adapt SLAs to avoid obsolescence thereof. This chapter has been published in the International Conference on Service oriented Computing'08 proceedings [73].

Chapter 7

Normative Management of Service Level Agreements

This Chapter proposes to manage Service Level Agreements defined between requesters and providers with normative agents. Normative agents offer usual advantages of multi-agent systems and benefit from a norm-oriented reasoning. The SLA control is a critical activity of service management, it enables to define execution requirements and to control the conformance of the execution to these requirements. SLAs allow to manage web services from quality information as they specify the quality level that should be met during the service fulfillment. SLA management requires beforehand quality specifications made with a quality model as the one proposed in Chapter 3 to insure the compatibility between user's requirements and provider's capabilities. Moreover, the SLA management is complementary to quality-driven management activities proposed in Chapters 4, 5 and 6.

7.1 Introduction

We focus in this paper on the critical task of ensuring that the contractual obligations of the parties – the service providers and the service clients – involved in a transaction are respected by these parties within a service-oriented system. Their obligations are typically outlined in a service-level agreement (SLA). An SLA is a contract between the said parties, which specifies the quality-of-service (QoS) levels that should be met [109]. QoS is a combination of several quality properties, e.g., availability, reliability, cost, response time [143]. A provider can propose the same service at different quality QoS levels. When a service client requests the execution of a given functionality, it advertises its QoS expectations. The service selected for the service execution will be the one that best satisfy client expectations about QoS properties. Prior to the transaction, the client and the provider enter into a contract by signing an SLA, and thereby specify quality levels to be observed during the service execution [109]. SLAs are used in the QoS management context in order to know what clients requirements to meet, how to manage clients expectations, how to regulate resources and to control costs [180].

The use of SLAs in managing the transaction between a provider and a client requires appropriate conceptual foundations and associated computational mechanisms. SLAs require an architecture if they are to enable the interactions between stakeholders. This architecture must support a specification language used by the stakeholders to communicate their expectations and capabilities. Similarly, a specification language to define the elements of the SLA is needed. Beside the architecture, the SLA management needs an incentive mechanism. To stimulate the correct behavior of stakeholders, these must have mutual obligations. E.g., the provider has the obligation to meet a given QoS level and the client has the obligation to pay according to that quality level. However, the client pays for the service after its execution by the provider. It follows that the client's payment can be adapted to the QoS level delivered by the provider. Finally, stakeholders can behave in an opportunistic manner, i.e., the client can underevaluate the QoS level perceived and the provider can exaggerate the QoS level offered. To prevent

such situations, the architecture must introduce a third-party controller to monitor the SLA execution.

The architecture must support the adaptation and supervision of SLA during the service execution. This architecture needs to be responsive, flexible and autonomic. Norm-oriented Multi-Agent Systems (NoMAS) provide these characteristics. Normative agents refer to agents conforming to norms. The core idea of this paper is to adapt the analogy of norms and agents to the issue of SLA and stakeholders. An SLA will be described as a set of norms to be fulfilled by the different agents of the system. The stakeholders of the service execution will be represented by normative agents complying to norms that restrict their behavior. The architecture supports a language enabling the communication between stakeholders. This language allows to express norms to be followed by the normative agents of the MAS. The elements constitutive of the SLA are defined by obligations norms regulating the stakeholders.

Contributions. We propose an architecture based on normative agents in order to: (i) enable the communication between stakeholders involved in the SLA with a common language; (ii) define SLAs that meet provider capabilities and client requirements; (iii) manage the service execution and check the conformance of the quality level expected and observed; (iv) ensure the execution of the mutual obligations according to the SLA contract. We propose to achieve the SLA compliance through two particular mechanisms: *mutual obligations*, which motivate the fulfillment of respective obligations of the involved stakeholders; and a *supervised interaction* with a third-party controller, which monitors and evaluates the SLA execution and penalizes the agents that does not fulfill their obligations.

Organization. Section 7.2 presents the conceptual foundations of the approach. Section 7.3 outlines the management architecture and the management of SLA. Section 7.4 proposes an evaluation of the proposed approach. Section 7.5 summarizes the related work and the Section 7.6 concludes this paper.

7.2 Case Study and Conceptual Foundations

This Section covers the conceptual foundations or our SLA management approach. We first describe the case study used to illustrate the approach proposed in this paper. We briefly introduce the Service Level Agreement concept. We also outline the mutual obligations and the supervised interaction used throughout the approach.

7.2.1 Case Study

We refer in this paper to a case study coming from the European Space Agency (ESA) program on Earth observation. This program allows researchers to access and use the infrastructure operated and the data collected by the agency[1]. The data and infrastructure of the ESA are accessed through web services. In order to facilitate the discussion and delimit our example, we focus on one part of the overall system. The MERIS/MGVI service is a service able to use the MERIS instrument data provided by the Envisat satellite of the ESA to compute the vegetation indexes for a given period of time and region of the world. A vegetation index measures the amount of vegetation on the Earth's surface. The data on the vegetation index can be obtained for any time range and it is possible to delimit the region of the world that is of interest. This service is subject to one particular QoS characteristic: the latency is initially situated between 4 and 6 hours by day of the selected period. E.g.: if the time range selected is from October 24th 2009 to October 26th 2009, the execution time needed to compute the vegetation index is set between 12 to 18 hours. The length of the selected period impacts then strongly the time needed to fulfill the request. The SLA specification between stakeholders of this service must clearly constrain the execution time prior to all remaining QoS properties. The different concepts presented in this paper will be illustrated with the MERIS/MGVI service and, specially about the execution time of this service.

[1] http://gpod.eo.esa.int

7.2.2 Service Level Agreement

A Service Level Agreement is a contract between the service provider and the service client specifying mutually agreed obligations of the provision of a service [31; 222]. The SLA concerns the non-functional properties of the service [109], i.e., quality properties. When clients can choose among a set of functionally equivalent web services, Quality of Service (QoS) considerations become the key criteria for service selection. As a consequence, SLA about nonfunctional properties must be defined and managed between service clients and providers [109].

The specification of QoS obligations of a SLA starts from a set of Service Level Objective (SLOs) [180]. A Service Level Objective is a guarantee of a particular state of the SLA parameters in a given time period [84]. All quality properties advertised by the provider are associated to an SLO as illustrated in Example 6. Each SLO has a functional part that refers to the QoS concerned and a guarantee part (italicized in Example 6) applied on the functional part. With SLOs, the SLA covers all quality properties defined in the QoS request of the service client.

Example 6. The provider of the MERIS/MGVI service shall execute the service *within 5 hours by day of the selected period.*

Example 6 is the SLO stating the maximum execution time of the agreement defined between the provider and the client. As referred in Section 7.2, the execution time of the MERIS/MGVI service is very important and needs to be clearly defined in the SLA. The SLA definition is communicated between the different stakeholders of the service execution. To assure the interoperability of SLA definitions, their specifications need to be written in a language common to providers and clients. The Web Service Level Agreement (WSLA) language [84] is one of the main standard specifying SLAs. The Example 2 illustrates how the SLO agreement of the Example 6 is specified with WLSA.
Example 2.

```
<ServiceLevelObjective name=''exectime''>
  <Obliged>provider</Obliged>
  <Validity>
    <Start>2009-10-25T08:00:00.000-05:00</Start>
    <End>2009-10-30T08:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Predicate xsi:type=''wsla:Less''>
      <SLAParameter>ExecutionTime</SLAParameter>
      <Value>ExecutionTimeThreshold</Value>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
<ServiceLevelObjective>
```

The `ExecutionTimeThreshold` used in Example 1 is a constant that assigns a name to a simple value that can be referred in other definitions [84]. This threshold corresponds to the maximal execution time expected by the client, i.e., 5 hours by day of the selected period to compute.

7.2.3 Mutual Obligations

Delivering the service at the quality level specified in the SLA is an obligation for the service provider. However, the service client has an obligation to provide all the information needed for the service execution (i.e.: inputs needed for web service execution), but also to pay for the service execution. Interactions between the provider and the client involve mutual obligations [125]. Such bilateral obligations motivate the SLA conformance. Indeed, breaches to some obligations of one party can compromise the fulfillment of obligations of the other party. Both parts have interest in achieving their obligations to meet the contract. Goodin [64] outlines the possible structures of mutual obligations. The SLA of web services can be defined as mutually conditional obligations. With mutual conditional obligations, each party is

obliged to discharge his obligations if and only if the other party discharges his obligations. E.g., if the service provided does not meet the contracted execution time, the client has not to pay the amount initially set. The SLA defines mutual obligations compelling the respective behavior of stakeholders.

The service execution is made of bilateral obligations, i.e., unilateral obligations from the provider about the service level execution and unilateral obligations of the client about payment or rating [107; 144]. We consider in the remainder of this paper that the client obligation is only about payment. However, other contractual obligations can be used as feedback rating as requests frequency.

The execution of obligations occurs sequentially: obligations of one of the stakeholders are executed before the obligations of the other. E.g., the provider's obligations are executed before the client's and the level of payment can consequently be adapted to the degree, to which the provider conforms to the obligations. Adaptations of the client obligations according to the observed quality level must be specified in the initial SLA. In the classification of mutual obligations [64], SLA contracts are diachronic mutual obligations, because one party is supposed to discharge its obligations before the other party does the same. The consequence is that initial contract must specifies the expected penalties if the defined quality level is not met [107]. The SLA contract implies that the penalty is initially accepted by the provider. Clearly, the efficiency of the relationship existing between the client and the provider improves if specifications of penalties for cases of contract breaches are present.

7.2.4 Supervised Interaction

Stakeholders of the service execution must achieve their respective obligations to conform to the initial SLA. If they are not supervised, they can adopt an opportunistic behavior, i.e., not fulfill their obligations or fulfilling them at a level lower than expected. E.g., the service client can reduce the payment even if the quality level provided meets is expectation. To prevent such situations, we propose to monitor the service execution with a third-party. This third-party will act as a controlling authority and allows to ensure the correct execution of the SLA. It is a witness of the service execution and stimulates the conformance to SLA for both involved parts. The third-party allows deterrence-based trust, i.e., you trust the other party because there is a very strict rule normative or legal system of rules, and the agent is punished for any violation of rules [35]. The third-party is the controller that controls the compliance of both parts to rules defined in the SLA, it measures the efficiency of the stakeholders transactions and computes their respective reputations [146]. An analogy of the third-party controller is the ebay online auction website [1]. The evaluation system of ebay prevents the opportunistic behavior of the stakeholders of the transaction.

This authority has an additional role in managing the SLA. Namely, it is in charge of collecting and computing metrics. It collects and stores metrics defined in the SLA and computes them to compare observed and expected results. Such metrics are used to establish the trust value of stakeholders involved in transactions. The measurement of quality values is allowed by existing metrics such as those discussed in [38]. If an SLA is breached, the third-party controller sends notifications to the involved stakeholders. The third-party is independent of the parties involved in the actual transaction, given that its aim is to prevent opportunistic behavior.

7.3 The Architecture and the Process for SLA Management

To solve the issues of SLA definition and its management during the service execution, we propose to use a normative MAS. Our proposed system will allow the SLA management and stimulates the SLA compliance through the respect of mutual obligations and the supervision of an authority. We first introduce our agent architecture that monitors the SLA through the service stakeholders in Subsection 7.3.1. We then explain how SLAs are managed with this architecture through the definition of norms associated to the stakeholders in Subsection 7.3.2.

[1] http://www.ebay.com

7.3.1 SLA management architecture

We chose to use a normative multi-agent system to monitor the execution of SLA. A normative multiagent system (MAS) involves normative mechanisms, which allow agents to adopt norms and specify how agents can modify these norms [19]. Norms can increase the efficiency of agent reasoning while their explicit representation supports reasoning about a wide range of behaviour types in a single framework [50]. Agent norms describe the obligations, permissions and prohibitions of a norm addressee to pursue certain activities, either to achieve a state of affairs or to perform an action [115]. The behaviour of an agent is monitored by its norms defining its permissions and obligations. Such a normative system allows deterrence based trust, the agent is punished for any violation of rules of the normative system [35].

The stakeholders of the service execution and the third-party controller are managed by normative agents. Norms condition the behavior of agents, the SLA is defined by obligations and prohibitions restraining the set of possible actions. We manage SLAs with normative agents coordinated within a suitable architecture. Three kinds of normative agents step in this architecture: the provider agents, the client agents, and the cluster agents. These are illustrated in Figure 7.1.

A cluster agent (A_{Clus}) is dedicated to each existing cluster of web services. A cluster of web services gathers functionally equivalent web services by providing several web services inside a unique wrapper. This wrapper is used by service clients as a standard web service. Services in a same cluster can be offered by different providers. The cluster selects the service that best satisfies the QoS expectations of the client in the cluster with an appropriate selection method [74]. This method relies on QoS advertisements of the provider and QoS expectations of the service client. These advertisements and expectations can have be made with WSLA [84] or another common appropriate language.

A service provider agent (A_P) is dedicated to each existing provider in the service cluster. A provider can offer several services in the cluster, i.e., the same functionality at different quality levels. This is illustrated in Figure 7.1 with the provider 1 and 3 each offering services with the same functionality and different quality characteristics. Providers can also offer services into different clusters, as they provide different functionalities. Provider agents advertise their QoS possibilities to the cluster agent.

A client agent (A_{Cli}) is assigned to each client requesting a service. The service request includes particular QoS expectations of the service client about the service execution.

Once the SLA has been negotiated [222] between stakeholders of the service execution it can be defined with an appropriate language. The cluster agent is responsible of the stakeholders conformance to the SLA defined. The cluster agent is also in charge of the collection and evaluation of metrics of the service execution. It collects information about QoS observed at each service transaction. It is able to compute statistical data on the service and able to determine if the service execution met the defined SLA. It is the third-party controller of the service execution, it controls the SLA compliance of the provider and the client agent. Agents of the SLA architecture are normative agents, conforming to norms derived from the SLA initially defined between the client and the provider. This architecture can be easily supported by existing normative MAS frameworks [50; 115]. In the remainder, we focus on how normative agents can support the management of SLA. The normative MAS infrastructure and communication is out of scope of this paper.

In the context of our case study, an agent is dedicated to each provider able to propose a service functionally equivalent to the MERIS/MGVI service introduced in the case study. A client agent is dedicated to the requester of the MERIS/MGVI service. A cluster agent is responsible for the supervision of interactions occurring between the client agent and the providers offering the functionality.

7.3.2 SLA Management Process

The management of SLAs with the proposed architecture covers several steps: (1) the definition of the SLA between the client and the provider; (2) the control of provider obligations, i.e., the service execution; (3) the penalties to apply to the provider if the SLA is not met, and; (4) the control of the client obligations, i.e., payment or evaluation. To fulfill these steps, the agents of the architecture introduced in Subsection 7.3.1 will take on different roles.

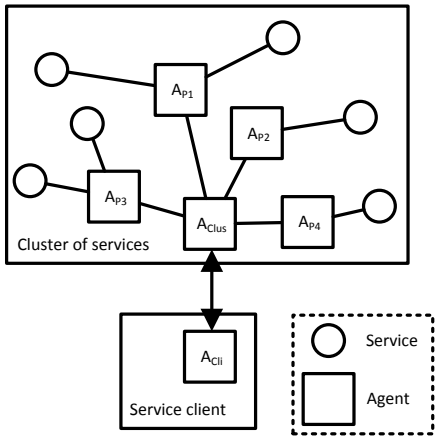


Figure 7.1: SLA management architecture

7.3.2.1 Step 1: Definition of the SLA.

The SLA is defined between the service client and the service provider from WSLA specification advertised by the provider. The WSLA specification is extended to include the mutual obligations of stakeholders. The defined SLA must cover expectations about the quality level of the service execution but also the penalties associated to breaches of SLA. These penalties are defined according to the importance of quality properties involved and according to the importance of breaches. Moreover, initial SLA specifications can also be enriched with complex rules, dependent rules or normative rules [168]. Such extensions allow the definition of enriched contracts, e.g., graduated rules are rules sets which specify graduated range for certain parameters so that it can be evaluated whether the measured values exceed, meet or fall below the defined service levels. To define these extended SLAs and include mutual obligations of service providers and clients, usual languages as WSLA [109] or WSOL [205] need to be enriched. To this aim, we choose to express the different SLOs of the initial SLA with obligation norms associated to involved agents of the architecture to benefit from the information added by more complex rules. To express SLO with normative obligations, we refer to the work of Kollingbaum [115; 116] about supervised interaction. Each SLO of the SLA contracted between the provider and the client is expressed with the NoA language [115] interpretable by all agents of the architecture. Moreover, complex conditions and penalties associated to SLO failures are also expressed with this language in further steps. The Example 3 illustrates the conversion of the SLO specified with WSLA in Example 2 into an obligation norm of the service provider agent specified with the NoA language.
Example 3.

```
obligation(
ServiceProvider,
achieve ServiceExecutionUnderTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold),
ServiceExecuted (ServiceProvider, Service) and
  ExecutionTime (Service) <= ExecutionTimeThreshold
ServiceExecutionUnderTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold))
```

This obligation states that the provider must achieve the execution of the MERIS/ MGVI service under the time limit (`ServiceExecutionUnderTimeThreshold`) specified in the initial WSLA specification (`ExecutionTimeThreshold`). The normative agents of the architecture monitor the execution of services through their norms. However, these agents can make a choice whether to obey the norms in specific cases. If the service provider is not able to achieve all SLOs of its contract, it can violate some of them to assure the fulfillment of remaining norms. This situation arises due to unexpected events (i.e.: additional requests, hardware failures) or because the provider amplified its capabilities to be selected. Among all

SLOs defined with obligations norms between the client and the provider, some can be met and some can not.

7.3.2.2 Step 2: Control of provider obligations.

Control is enabled through mechanisms of normative agents. Each agent of the architecture fulfills one or several roles in the contract management. The SLA contract is then monitored through these different roles: the addressee commits an obligation defined in the contract; the counter-party is the recipient of the obligation fulfilled, and; the authority is a witness of the contract. The authority is in charge of the correct execution of the contract and imposes sanctions in case of defective behavior of the addressee. The different roles of client, provider and cluster agents are illustrated in Figure 7.2. The provider illustrated in the service cluster of this example is the provider 1 among those proposed in Figure 7.1.

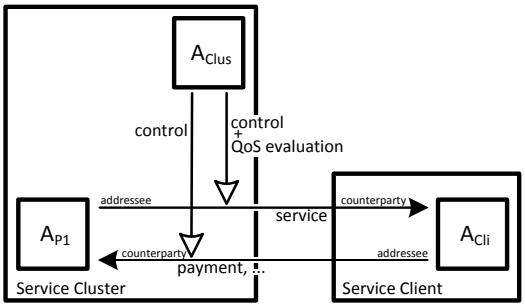


Figure 7.2: Roles fulfilled by normative agents

The interactions between the service provider and the service client (i.e., the service execution and its payment) are restrained by obligation norms associated to these roles. The SLOs specifying the expected QoS level of the MERIS/MGVI service appear as norms. The provider agent is the addressee in these norms, while the client is the counter-party in the transaction. To control the achievement of this contract, the cluster agent acts as an authority. As stated in Subsection 7.3.1, the cluster agent is responsible for collecting and computing the metrics in order to control the SLA execution. It is then able to determine if the service provider meets the SLOs defined through obligation norms. The cluster agent will impose sanctions when the quality level provided does not meet the level contracted in the SLA. Such sanctions appear as penalties applied to the provider. As stated before, these penalties are part of the initial SLA. In the MERIS/MGVI instance, the decreasing of payment is proportional to the reduction of quality level provided. Sanctions are expressed by obligations norms to be followed by the cluster agent. When the provider chooses or is forced to breach a norm specifying one of its SLO, the cluster agent captures it and activates a specific penalty. There can be several norms specifying different penalties corresponding to the spreading of the breach. The Example 4 illustrates a specification of one such penalty.
Example 4.

```
sanction(
ServiceCluster,
perform EvaluationTime (ServiceProvider, Service, ExecutionTimeThreshold,
  ExecutionTimeThreshold 2, AmountPenalty),
ServiceExecuted (ServiceProvider, Service) and
  ExecutionTime (Service) > ExecutionTimeThreshold and
  ExecutionTime (Service) <= ExecutionTimeThreshold2
TimePenalty(ServiceProvider, AmountPenalty))
```

When one of the SLOs of the MERIS/MGVI service is not met, a sanction is applied by the cluster agent according to the importance of the breach. As stated in Section 7.2, the execution time is a critical issue for the MERIS/MGVI service, sanctions to apply must penalize all provider weaknesses about delays. The Example 4 illustrates one sanction: if the execution time observed is above

the SLA time limit (`ExecutionTimeThreshold`) but is under the second time limit of the breach scale (`ExecutionTimeThreshold2`), the decreasing of payment (`AmountPenalty`) applied is proportional to the observed level on the breach scale. The cluster agent independently estimates the equality of the quality level provided and the amount to pay. Moreover, according to characteristics of mutual obligations in SLAs, the user must discharge its obligations only if the provider has discharged its owns obligations.

7.3.2.3 Step 3: Penalties to apply.

When the cluster agent observes that the SLA is not fulfilled by the provider agent, it notifies the service client through the application of a sanction. The client agent will then reflect this sanction on its own behavior. The mutual obligations of SLAs are diachronic; the client obligations are adapted to the provider fulfillment of its owns obligations. In the Example 5, the `TimePenalty` is a constant defining the payment reduction of the MERIS/MGVI service initiated by the sanction of the Example 4. There can be several payment reduction to apply, corresponding to different level of breach or to different QoS properties involved in the SLA. According to the importance of the breach, the client agent follows the norm defining the corresponding penalty. The obligation of the Example 5 is the client obligation to pay for the MERIS/MGVI service execution, i.e., the contractual obligation of the client. However, the initial payment amount (`Amount`) is reduced by the penalty (`AmountPenalty`) induced by the time sanction illustrated in Example 4. The payment of the service is an obligation norm in which the client agent is the addressee and the provider agent is the counter-party as illustrated in Figure 7.2.
Example 5.

```
obligation(
ServiceClient,
achieve ServicePayment (ServiceClient, ServiceProvider,
  Amount - AmountPenalty),
ServiceExecutionUnderExecutionTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold)) and
  TimePenalty(ServiceProvider, AmountPenalty)
achieve ServicePayment (ServiceClient, ServiceProvider,
  Amount - AmountPenalty))
```

7.3.2.4 Step 4: Control of client obligations.

The third-party controller checks the execution of the unilateral obligations of the service provider as detailed in Step 2. Similarly, the third-party controller must verify the obligations of the service client, i.e.: its payment to the provider after the service execution. To control the right execution of the payment obligation illustrated in Example 5, the cluster agent acts as the third-party controller. It is the authority of the payment transaction as illustrated in Figure 7.2. It must check that the right amount has been deposited to the provider. If the client fails to pay or deposits a bad amount, the cluster agent must apply a penalty. The Example 6 illustrates the sanction applied by the cluster agent to the client of the MERIS/MGVI service when the payment obligation is not met. With such sanctions, the cluster agent avoids the non payment of the service client. Indeed, if the payment is not made or if it is insufficient, the client is labeled as a bad payer (`PaymentPenalty(ServiceClient)`) by the third-party controller. The cluster agent can then reject future requests of bad payers on its cluster.
Example 6.

```
sanction(
ServiceCluster,
perform CheckPayment ServiceClient, ServiceProvider, Amount, AmountPenalty),
not ServicePayment (ServiceClient, ServiceProvider, Amount - AmountPenalty)
PaymentPenalty(ServiceClient))
```

7.4 Evaluation

Supervised interaction and mutual obligations improve services executions from both user and provider sides. We conduct some experiments in order to evaluate the effect of these mechanisms. These experiments simulate services transactions between users and providers and measure their utility with and without the utilization of such mechanisms. The utility denotes the abstract quality whereby an object serves our purposes, and becomes entitled to rank as a commodity [93]. We suppose here that the utility increasing is constant for each new transaction initiated. Each transaction initiated by a client involves a cost decreasing of its cumulated utility while each successful transaction increases its cumulated utility. The ratio over the increasing induced by the success of the transaction and the decreasing due to the cost of the transaction must be positive. E.g.: in our simulations, the increasing of utility is set to 1 while the service execution succeeds and the utility decreasing of the service payment is 0.8. The net utility of a service transaction is then 0.2. We generate 200000 transactions from 10000 different providers to 100 different users. Each service is executed 20 times by each service client. To simulate the opportunistic behavior of providers, we define 30% of opportunistic providers that do not fulfill their transactions 70% of time. Without mutual obligations and supervised interaction, the decreasing of client utility involved by the service payment occurs even while services executions fail.

To simulate the supervised interaction effect, we introduce a simple trust model. The trustworthiness of each provider is collected by the third-party controller. The third-party controller monitors all services executions and dismisses providers that fail 10 services executions previously supervised. While services executions occur without supervised interaction, the clients collect themselves information about past executions and dismiss providers that failed 3 of their own previous transactions.

To simulate the interest of mutual obligations, we introduce a variable payment model. The service client can reduce the initial payment while the provider obligations are not met. The utility decreasing of the service client can be less important when the service execution fails. E.g.: in our simulations, the utility decreasing involved by the payment is 0.8 when the service execution succeeds and is reduced to 0.2 while the service execution fails. Without mutual obligations, the payment has to be done and the decreasing of the client utility is fixed to 0.8.

However, the third-party controller offering such monitoring mechanisms has to be payed. We designed two different scenarii to simulate the payment of the third-party controller: a variable and a fixed remuneration. The variable remuneration implies a decreasing of the client utility at each service execution. This variable cost must be proportional to benefit of a transaction. E.g.: if the net utility before the remuneration of the third-party controller is 0.2, the third-party fee of each transaction can be 0.02. The fixed cost allows clients to benefit from third-party mechanisms after a single payment. It implies an important decreasing of the client utility. E.g.: in our simulations, we set the initial utility of the client to -1000 while the third-party controller relies on a fixed cost.

To evaluate benefits form supervised interaction and mutual obligations, we observe the mean cumulated utility of users during 200000 services executions. To highlight the benefits of third-party mechanisms, we design 7 models: (1) services executions without supervised interaction (*s.i.*) and without mutual obligations (*m.u.*); (2) services executions without *s.i.* and with *m.u.* at a variable cost; (3) services executions without *s.i.* and with *m.u.* at a fixed cost; (4) services executions with *s.i.* and without *m.u.* at a variable cost; (5) services executions with *s.i.* and without *m.u.* at a fixed cost; (6) services executions with *s.i.* and *m.u.* at a fixed cost, and; (7) services executions with *s.i.* and *m.u.* at a variable cost. We then measure the difference between the optimal cumulated utility and the cumulated utility of our different models (i.e., the optimal client utility is get while the service client never pays for the services executions that fail).

The results of our experiments are highlighted in Figure 7.3. The model nearest to the optimal client utility is the model (7) that provides both supervised interaction and mutual obligations with an initial fixed cost. However, this model becomes the best only when the initial cost is balanced by its profitability (after approximatively 64500 services executions) while at the beginning the most profitable model is the model (6) that provides both supervised interaction and mutual obligations at a variable cost. The profitability of each model is dependent from the third-party controller payment scenario but

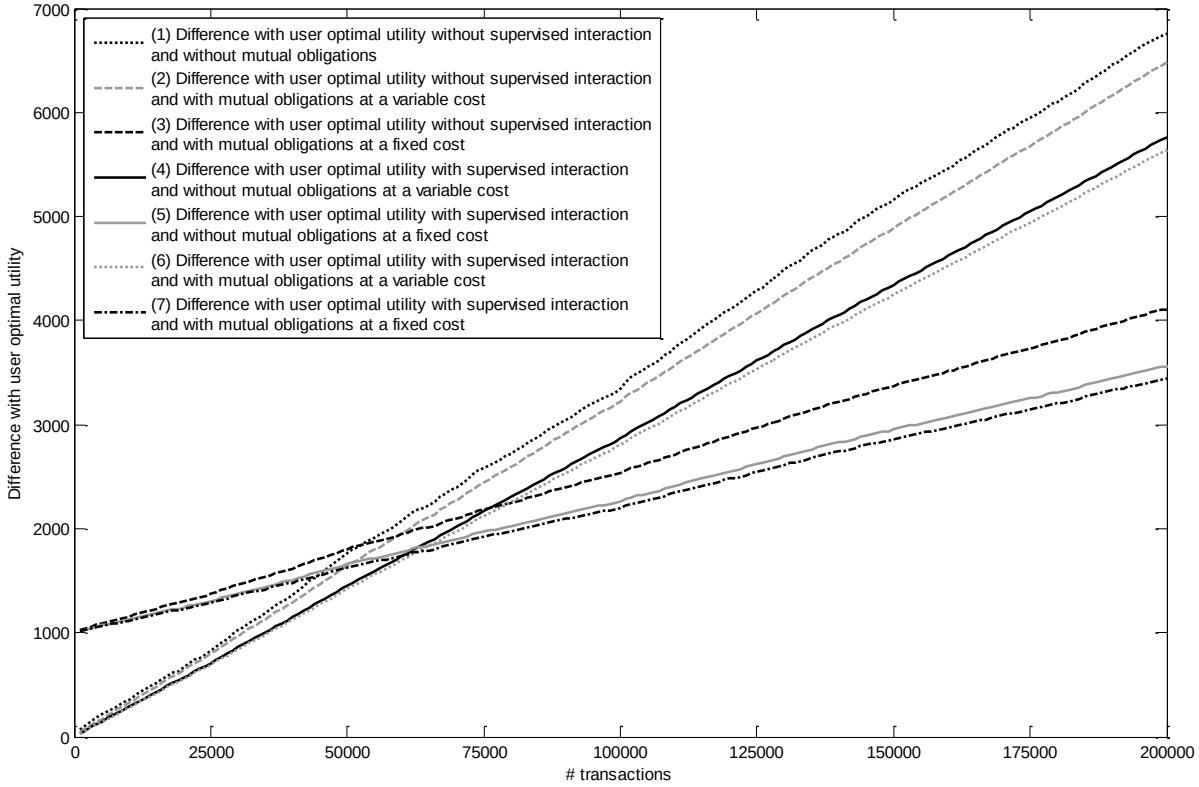


Figure 7.3: Simulation Results

the utilization of third-party mechanisms always improve the client utility. The less profitable model is the (1) that provides neither supervised interaction neither mutual obligations. Models offering only mutual obligations ((2) and (3)) improve lightly the client utility while models providing only supervised interaction ((4) and (5)) ameliorate strongly the client utility. The combination of both mechanisms (models (6) and (7)) outperforms other models and highlight the interest of normative agents to control SLA of stakeholders transactions. The experiments conducted here to evaluate the client utility can be transposed to the provider utility. We can also simulate opportunistic client that do not fulfill their obligations and evaluate the mean utility of providers.

7.5 Related Work

QoS properties and SLA management need appropriate architectures to be handled during the service execution. Campbell et al. [29] propose the Quality of Service Architecture (QoS-A) incorporating the notion of flow, service contract and flow management through QoS properties. Barbosa et al. outline in [9] different architectural configurations to enable the auditing of SLA and to evaluate their efficiencies. The WSLA framework [109] introduces a runtime architecture comprising several SLA monitoring services. Some services may be outsourced to third parties to increase the objectivity in the evaluation of the services. The QoS Mission-Action-Resource (Q-MAR) model [83] and the Grid Quality of Service Management (G-QoSM) framework [3] also propose to distribute the SLA monitoring to the different components of the system. Paschke et al. [168] introduce a Rule-Based Service Level Management (RBSLM) architecture in which SLAs are represented with declarative rules and managed through logical concepts and rule languages. Although all these architectures allow one to observe when a contract is violated, most of them do not prevent such violations and do not clearly define corrective actions to take. In our proposal, the third-party monitors stakeholders behaviors and the mutual obligations of the stakeholders

define penalties to apply while the contract is not fulfilled. The BREIN project [1] offers an architecture enabling the management of SLAs through their whole lifecycle [28]. The SLA management is enabled by taking into account the policies of the parties and their respective business goals. The BREIN SLA management offers preventive monitoring to react to upcoming violations and a prioritization of SLAs. Our normative management of SLAs adapt contracts at runtime in response to unexpected violations in order to maximize stakeholders satisfaction.

Agents systems are well fitted to monitor activities requiring negotiation between stakeholders as SLA management or e-commerce mediation [71; 193]. Other existing SLA architectures relies on multi agent systems [207]. Yan et al. [222] introduces a MAS architecture supporting the negotiation of services involved in a composition. In comparison with existing MAS architectures, our proposal is supported by normative agents. Normative agents allow to constrain the stakeholders behavior with norms defining the SLA to be achieved. They are particularly relevant to the SLA management issue. Normative agents are also used by Pitt et al. [169]. They propose a framework for QoS management which combines events, metrics and parameters with organizational intelligence offered by norm-governed multi-agent systems. Although their proposal monitors QoS information, they did not tackle the SLA conformance issue. One of the strongest point of our work is that the agreements between clients and providers are defined and monitored through norms associated to roles of agents and not to agents or components of the architecture. These roles allow the architecture to offer more flexibility, e.g., the provider can be easily substituted when unexpected failures occurs. The Appendix C provides more information about SLA management.

7.6 Conclusions

We propose in this paper an architecture enabling the management of SLA. This architecture relies on a MAS and supports a normative definition of SLA. The MAS enables the communication between stakeholders involved in the SLA. Each party of the SLA is defined with an obligation norm that constrains stakeholders behaviors. The architecture checks the conformance of the stakeholders to the SLA. To stimulate the proper execution of the SLA, its execution is driven by mutual obligations and supervised by a third-party controller. The architecture benefits from the potential autonomy assured by normative agents. The normative architecture enables the interactions between provider and client and also the evaluation of the quality level of such interactions.

[1] http://www.eu-brein.com

Chapter 8

Context driven Adaptation of SLAs

This Chapter has been published in the International Conference on Service oriented Computing'08 proceedings [73]. The Chapter 7 has presented a normative agents architecture enabling the management of web services level agreements. This architecture gives foundations for the control and monitoring activities performed by agents to ensure the contract achievement. However, this architecture tracks inconsistencies but does not prevent the failure of contracts. In this Chapter, we propose a model enabling to control SLAs upon basis of context information. The model we propose here also allow to autonomously adapt SLAs contracts to ensure the achievement of the service execution.

8.1 Introduction

Web services are a response to growing needs of responsive and configurable applications on the Internet. A service is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network [162]. Web services are supported by technologies such as SOAP, UDDI and WSDL [211] and are accessed via a Uniform Resource Locator.

Given the growing number of available web services on the Internet, different service providers may offer services that provide the same functionality to the users. Such competing services can be distinguished by comparison over nonfunctional characteristics, which take the form of Quality of Service (QoS). QoS is a combination of several qualities or properties of a service, e.g., availability, security, response time or throughput [143]. When a user requests a service to perform some given task, a service is selected that fits the user's QoS requirements. The selected service is the one that meets the most adequately user's preferences over quality attributes that go into QoS. Once the service is selected, it is assigned by the definition of a contract that defines a Service Level Agreement (SLA) between the user and the provider [127; 148]. SLAs are used to meet user's requirements, manage user's expectations, regulate resources and control costs [180]. In short, SLAs are used to set the QoS level offered by the service provider to the service user; SLAs result from a negotiation initiated between these parties [106].

However, offered and requested QoS may both vary over time. We say in this chapter that such variations occur because of changes in the *context* of services. Given that the term "context" can be widely understood, a definition local to this chapter is in order: *context* is any information about the interaction between users and a web service, for which an SLA is specified.

Changes in the context should be reflected in the SLA governing the interaction. Both the offered and the requested QoS levels may vary over the course of the interaction. Moreover, there are dependencies across different context elements that indicate a propagation of variations from one element to multiple context elements. To keep the SLA unchanged in such conditions is to make the SLA obsolescent.

Contributions. We propose an approach that enables an autonomic adaptation of SLA to respond to occurring context modifications. We illustrate our approach with a case study based on European Space Agency (ESA) services used to process information provided by the Envisat satellite. We provide

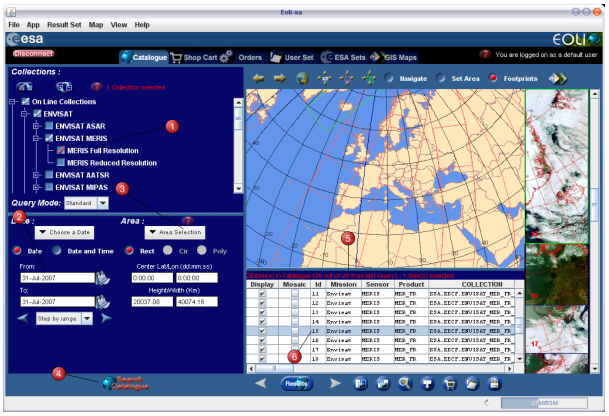


Figure 8.1: Graphical interface of the EOLI-SA service

conceptual bases necessary for SLA adaptation. We classify context elements into five distinct categories: user, provider, resource, environment and web service. We also introduce dependencies existing between elements of context enabling to propagate context modifications. We then present our SLA adaptation approach. We propose an architecture relying on an SLA manager to drive the autonomic adaptation based on context elements. Our adaptation uses context modifications and dependencies to enable an autonomic adjustment of existing SLAs to ensure the service conformance to user expectations. Adaptation involves the following steps:

1. Context modifications are reported to the SLA manager that identifies changes and starts the adaptation process.

2. Observed context variations are propagated through context dependencies existing over different elements of context by the SLA manager.

3. Once context variations have been propagated to all context categories, the SLA manager checks the compatibility between user expectations and provider capabilities.

4. Upon base of the result of the compatibility checking, the SLA manager keep the existing SLA, set up a new SLA between the user and the same provider or select another service fitting better to user expectations.

Organization. Section 8.2 introduces the ESA case study used throughout the chapter. Section 8.3 presents the conceptual elements used to drive the SLA autonomic adaptation and illustrate these concepts with the case study. Section 8.4 propose our SLA management architecture and assesses the different steps of our adaptation process. The case study illustrates the adaptation process. Section 8.5 presents the related work; Section 8.6 draws conclusions.

8.2 Case Study

The case study proposed here refers to the gpod ESA program described in Subsection 3.2.2. In relation to MERIS, a large set of web services is made available by the ESA for access to the data the instrument sends and access to the provided computing resources.

We are interested in the remainder about two specific services. The first provides the vegetation indexes for a given period of time and region of the world. A vegetation index measures the amount of vegetation on the Earth's surface. The graphical interface used by the requester of the service is shown in Figure 3.1. An illustration of the output provided for the whole world map is given in Figure 4.1. The data on the vegetation index can be obtained for any time range and it is possible to delimit the region of the world that is of interest. This service is subject to particular nonfunctional properties: the

latency is initially situated between 4 and 6 hours by day of the selected period due to the quantity of data to process. Thus, the service user expects to minimize the execution time. For a service facing such significant latency, service reliability is another critical QoS aspect. Indeed, in case of failures, all execution steps must be started over. So, maximizing the reliability reduce risks to have to start over. The second web service used to illustrate our approach is the EOLI-SA service: this service is used to calculate metadata on the products to process. For example: when you submit a zone to process the MGVI with the 'bounding box' argument, these coordinates need to be transformed into the technical data of the satellite at the time of the acquisition of the zone to process (start/stop time, orbit, lat/long, azimuth angle, etc.). The graphical interface of the EOLI-SA service is given in Figure 8.1. This service presents different nonfunctional characteristics: while it is used by other services as the MERIS MGVI/Regional, the availability of this EOLI-SA must be maximized in order for these services to execute successfully.

8.3 Conceptual Foundations

This section introduces the concepts used to drive our autonomic SLA adaptation. We present our notion of context, and subdivide it into categories in Section 8.3.1. Section 8.3.2 introduces the concept of dependencies over context elements. All the concepts are illustrated through services introduced in Section 8.2.

8.3.1 Context categories

Unexpected events can modify the current execution context and have an impact on the performance of the services. These modifications can breach the SLA. To adapt existing SLAs to context modifications, context elements need to be accurately defined by services providers and users. We classify context elements into several categories, shown schematically along with potential between-category interactions in Figure 8.2.

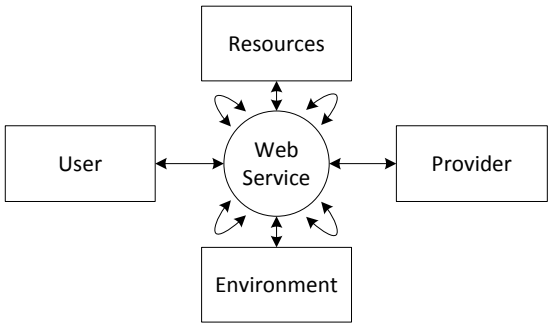


Figure 8.2: Context categories and between-category interactions

User context: The *user context* covers the user's QoS requirements. These requirements are expressed with help of preferences over QoS values that the service must achieve but also with QoS priorities specifying which QoS properties will be maximized over others [172]. The user context also carries information on past executions of services, along with advertised and observed QoS values during these executions [147]. Changes in user context may eventually induce the definition of new SLAs between the user and the provider. The user specifies and updates the user context.

Resources context: Web services executions are influenced by the availability of the resources that concern the network connection between the provider and the user but also the hardware used in executing the service and/or retrieving its results [148]. It is clear that resource availability has a direct impact on delivered QoS, thereby affecting the satisfaction of SLA. Both the service user and the service provider specify the resource context by providing their respective resource-related information. They also update this information when changes occur.

Table 8.1: Context particularities of MERIS/MGVI and EOLI-SA services

Category	MERIS/MGVI Regional	EOLI-SA
user context	maximize reliability and minimize execution time the execution time must be inferior to 7 hours by day of the selected period	maximize availability
resources context	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN
environment context	service user is an human	service user is another web service
provider context	current execution charge of the computing cluster	current execution charge of the computing cluster
web service context	execution time: 4 and 6 hours by day of the selected period reliability: upper than 95% availability: upper than 92%	execution time: inferior to 1 min reliability: upper than 98% availability: upper than 98%

Environment context: The environment context contains information about where the user is located [47] and about its surrounding environment like the current weather or date [172]. This information also includes about the network, which is not within direct control of service user or provider [95; 148]. The network that is not under the user's or provider's responsibility can have an immediate impact on the service performance. The environment context depends on the user of the service and is specified by the SLA manager.

Provider context: Provider context covers, among others, information about the provider's current execution load, the duration of its current opened sessions and announced intended length of usage by the application requesting access [95]. All activities performed by the web service and its execution charge have a direct impact on the service's QoS. Increasing or decreasing the computation charge may require changing the SLA. Provider context is specified and updated by the service provider.

Web service context: The Web Service context refers to nonfunctional characteristics of the service. It provides information about possible ranges of execution time, levels of security, expected best reliability, and so on [130; 172]. Latency or security are determined by the service's implementation, while metrics like mean availability or reliability are obtained from its past executions. Any changes in the web service context will affect QoS levels, leading to SLA adaptations. The web service context is specified by the service provider.

Provider, web service and some part of the resource categories are related to elements of the provider side and define the level of service that can be offered. User, environment and the other part of the resource categories concern items of the user side and determine the expected level of service. The modifications of all elements of context categories are performed either by the service provider, or by the service user, except for the environment context that can be affected by external events.

Context illustration. We illustrate in Table 8.1 the context elements for services from the case study. Both services are offered by the same provider and are executed on the same computing cluster. Their provider and resources elements are consequently similar.

The MERIS/MGVI service has an important execution time. To prevent failures and potential restart of the execution, the user wishes both to minimize the execution time and maximize the reliability of the web service. Moreover, the user adds a hard constraint on the execution time, stating that it must be inferior to 7 hours by day of the selected period. This constraint prevents an accumulation of unfulfilled requests by the MERIS/MGVI service. The EOLI-SA service has a faster execution, its reliability is not so critical. As it is used by other services to compute data, its availability must be maximized to increase reliability of these other services.

8.3.2 Context dependencies

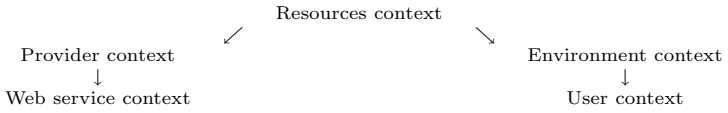
Context dependencies refer to relationships that exist between distinct context elements. For example, QoS properties supported by a service provider can be interrelated [212] e.g.; change in the execution time can affect reliability. These relationships can also occur over elements in different context categories – e.g., the execution charge of computing resources (provider context) affects various quality characteristics

Table 8.2: Examples of Context Dependencies

Common dependencies of MERIS/MGVI and EOLI-SA services		
Dep 1		Resources context - Web Service context
	Coefficient:	parallel
	Direction:	→
	Strength:	10
Dependencies of the MERIS/MGVI service		
Dep 2		Execution charge of the computing cluster (provider context) - Execution time (web service context)
	Coefficient:	parallel
	Direction:	→
	Strength:	6
Dep 3		Execution charge of the computing cluster (provider context) - Availability (web service context)
	Coefficient:	opposite
	Direction:	→
	Strength:	8
Dep 4		User bandwidth (environment context) - Transfer Time (user context)
	Coefficient:	parallel
	Direction:	→
	Strength:	5
Dependencies of the EOLI-SA service		
Dep 5		Availability (web service context) - Reliability (web service context)
	Coefficient:	parallel
	Direction:	↔
	Strength:	9
Dep 6		Execution charge of the computing cluster (provider context) - Availability (web service context)
	Coefficient:	opposite
	Direction:	→
	Strength:	2

of the service (web service context). To highlight such dependencies between elements allows us to propagate failures and performance modifications to all context categories related to the initial variation. Dependencies are defined over context elements with an associated *coefficient*, *direction* and *strength*. The *coefficient* attribute specifies that context elements involved in the dependency are parallel or opposite, meaning that their coefficient is positively or negatively correlated. The *direction* determines, which of the two considered context elements induces the value of the other; a dependency can be directed both ways, meaning that both context elements impact each other. The *strength*, represented by a value between 1 and 10, corresponds to the importance of the influence.

While all context dependencies occurring on the same context category are allowed, between-category dependencies are subject to some restrictions. We specify in Section 8.3.1 that part of the resources context, the provider context and the web service context are defined by the service provider. The other part of the resources context, the environment context and the user context are delimited by the service user. Dependencies can not involve influence of provider context categories to user context categories and vice versa. Dependencies are also constrained by the attribute direction over different categories. An improvement of the computing resources (resources category) can induce the service quality performance (web service context). Nevertheless, the service quality performance (web service context) has no influence on the computing resources (resources category). The impact direction of dependencies in context categories is given below:



Examples of context dependencies. Context dependencies are specified by the SLA manager to indicate existing interactions between context elements. Table 8.2 gives examples of context dependencies for MERIS/MGVI and EOLI-SA services.

Dep 1 states the relationship between the resources used by the service provider and web service performance. Indeed, the capability of the web service is immediately linked to the resource used to compute the web service. If the server used to compute MERIS/MGVI is down, all its performance indicators will be affected. The EOLI-SA service is also subject to that dependency. *Dep 2* underlines the impact of the execution charge of the cluster on the execution time needed to execute the service.

Table 8.3: Examples of SLAs

SLA established for the MERIS/MGVI service	
SLO 1:	the execution time must be between 5h and 6h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
SLA established for the EOLI-SA service	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 92%

Increasing the execution charge of the provider decreases resources allocated to the execution of the service and increases its execution time. *Dep 3* states that the increasing of the execution charge of the cluster will decrease the availability of the service: if the cluster charge is full, the MERIS/MGVI service that requires an important resources utilization will not be given a high priority. Consequently, its availability will be reduced. *Dep 4* is about the influence of the user's bandwidth on the user's network capacities. If the bandwidth provided by its Internet Service Provider decreases, the service user will amend its expected total transfer time. The EOLI-SA service is less subject to context variations because it does not consume that much resources. It is subject to the dependency linking its availability to its reliability: *Dep 5*. This dependency is directed both ways meaning that the increasing/decreasing of one of the quality property affectes the other. It is also subject to the *Dep 6* stating that the increasing of the execution charge of the cluster causes a diminution of the availability. This dependency is the same that the one observed for the MERIS/MGVI service but its strength is not so prominent because EOLI-SA does not require a long execution time and is less subject to the provider's utilization.

8.4 Dynamic SLA Adaptation

We outline here our adaptation process that fits SLA established between service user and provider to context elements variations. Section 8.4.1 gives a SLA description and presents our SLA management architecture. Section 8.4.2 details the different steps of our adaptation process.

8.4.1 Managing Service Level Agreements

Contracts between a service provider and a service user are given by SLAs [127]. An SLA covers the functional side of the service (the provided service corresponds to the requested service in terms of input, output, pre- and postconditions) and concerns also the nonfunctional properties of the service. When users can choose among a set of functionally equivalent web services, QoS considerations become the key criteria for service selection. As a consequence, SLAs must be defined and managed between service users and providers [31]. The contract about non-functional properties is defined for each QoS property through a Service Level Objective (SLO) [180]. SLOs are defined over QoS values and appropriate metrics. Definitions of metrics include the description of their calculation mode and are provided by the party in charge of measurement and aggregation, i.e.; either the service provider, the service user or a third tier manager [31; 222]. An SLA is then a contract between the service user and service provider about a set of SLOs. These SLOs refer to web service and user context elements. Web service context elements are QoS capabilities of the provider while user context elements are quality requirements of the service user. A complete definition of a SLA and its component is available in [181]. We work on the assumption that an initial SLA has already been negotiated between the service user and the service provider with a negotiation process such as the one specified in [222]. We propose in Table 8.3 examples of SLAs established between providers and users for both services presented in the case study section.

To manage SLAs and their adaptation, we introduce a third-part service: the *SLA manager*, in charge of the mediation between the service user and the service provider. The adaptation process refers to automatic monitoring, enforcement and optimization of SLAs between the services's user(s) and provider. The SLA manager is also responsible of the assignation of services to users. Our management

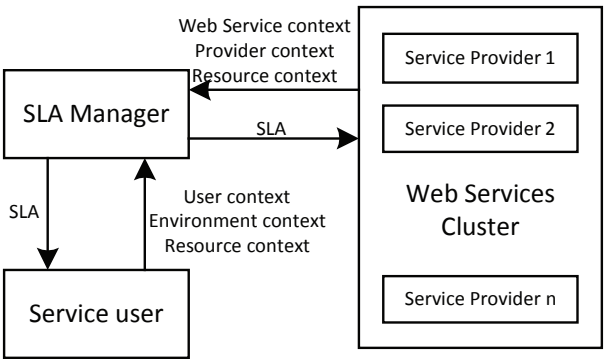


Figure 8.3: SLA Management Architecture

architecture is illustrated in Figure 8.3. We dedicate one SLA manager for each existing cluster of web services (i.e., services that offer the same functionality). Gathering of functionally equivalent web services is ensured by means of clusters of web services, that provide several web services inside a unique wrapper, used by the clients as a standard web service [210]. We suppose than an initial SLA has already been negotiated between the service user and the service provider chosen with an adequate selection method [134].

The role of the SLA manager is to continuously check the conformance of the web service to the SLA established between the user and the provider. This monitoring requires a constant verification of SLOs compliance between the service user and the service provider. To achieve this verification, context information about the web service, the provider and the part of the resources information are given by the provider while information related to the user context, its environment and its resources are communicated by the service user. The SLA manager records information about all context elements and builds execution statistics about mean observed latency, reliability or availability. The SLA manager also monitors context dependencies with help of information provided by the user and the provider. With such statistics and information about the execution context and dependencies, the SLA manager is able to check the conformity of SLOs established between the service user and the service provider. If some SLOs are breached, the SLA manager processes adaptation mechanisms to adjust the SLA, as explained in Section 8.4.2.

8.4.2 Adapting Service Level Agreements

The SLA manager is designed to respond to eventual SLA breaches or QoS variations through different mechanisms of adaptation. *Adaptation* usually refers to the alteration of an application's behavior or interface in response to arbitrary context changes [22]. For web services, the adaptation must consider all context particularities introduced in Section 8.3.1 as well as existing dependencies over context elements presented in Section 8.3.2. The aim of such adaptation mechanisms, referred as *SLA adaptation*, is to adjust the initial SLA to context variations reported to the SLA manager. If the initial web service provider is no longer able to perform its task to the quality level requested by the user, the SLA manager proceeds to *select a new provider*. It establishes a new contract between the service user and a service provider selected in the cluster of available services.

The SLA manager process this adaptation through four steps:

1. Modification notification. The SLA adaptation process is driven by the observation of a modification in at least one context category. Such changes are highlighted by information provided by users and providers and statistics made by the SLA manager. The adaptation is initiated differently following the category of the context variation. Provider, web service and some part of the resources context come from the service provider and their changes will modify the service offered, while the user, the environment and the other part of the resources context are defined by the service user and will affect the service level expected.

2. Modification spreading. The second step of the SLA adaptation is the propagation of observed

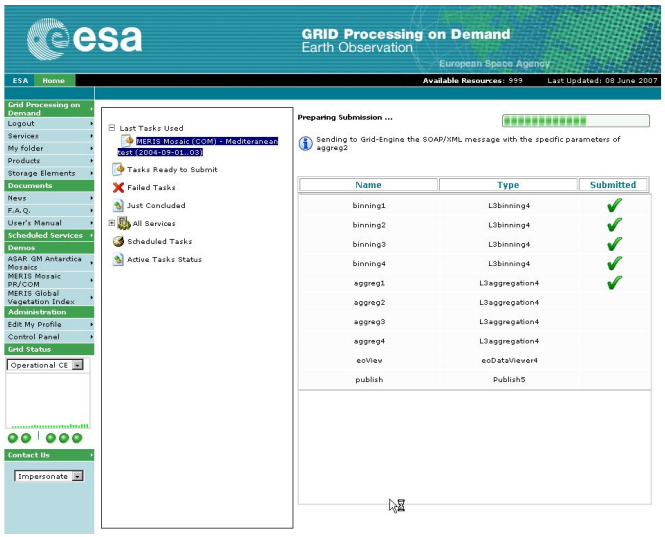


Figure 8.4: Current tasks of the provider

context variation to elements of the same category and to other relevant context categories. Spreading the modification is subject to rules presented in Section 8.3.2, which define the direction of the allowed dependencies. The impact of the context variations is governed by the coefficient, direction and strength attributes and reflects changes to all elements of the concerned context categories. Context dependencies allow the SLA manager to propagate the impact of context categories until their influence to related QoS: all context variations are converted to elements used in the SLA contract (i.e., user and web service context).

3. Compatibility checking. Once all dependencies have been propagated, the SLA manager is able to determine the final quality expectations of the user and the web service QoS offered by the service provider. To ensure their compatibility, the SLA manager checks context elements accounted for in the SLA – i.e., the user and web service context. If the web service context presents abilities that meet the expectations of the user context, these are compatible.

4. Adaptation. The last step of the process is the adaptation resulting from compatibility checking. Three different scenarios are possible. (1) The compatibility is present between web service and user context and the initial SLA is still applicable. In this instance, the SLA is preserved between stakeholders. (2) The compatibility is verified between web service and user context but the initial SLA no longer applies. The SLA initiates the set up of a new SLA between the current provider and the service user. To achieve the negotiation between the user and the provider, the manager uses a negotiation process such as the one proposed in [222]. (3) The last possibility occurs while the compatibility between the user and the provider is not verified. The SLA manager then select another service able to meet the quality requirements of the user context in the web services cluster. We do not review here details of selection mechanisms but various existing approaches [75; 134; 229] can be applied by the SLA manager. The SLA is negotiated between the new provider and the service user by the SLA manager.

Adaptation illustration. We illustrate here adaptation steps through a particular situation involving services introduced in the case study.

The adaptation process described here occurred with an increase of the execution charge in the computing cluster. The execution charge of the computing cluster belongs to the provider context category. The services accessing the computing cluster offered by the ESA are monitored and managed through a particular access interface illustrated in Figure 8.4. The execution charge can significantly increase with entrance of new requests in the computing cluster. The adaptation mechanisms initiated in response to these new requests will differ with the extent of the increasing. The adaptation process initiated by this increasing is described through its four steps here.

The *first step* is the modification notification. The provider charge is monitored through the applica-

Table 8.4: Web Service context of MERIS/MGVI and EOLI-SA services after an increasing of the execution charge

Increasing of the execution charge of 20%	
MERIS/MGVI Regional	EOLI-SA
execution time: 5h and 7h hours by day of the selected period reliability: upper than 95% availability: upper than 85%	execution time: inferior to 1 min 10 sec reliability: upper than 97% availability: upper than 97%
Increasing of the execution charge of 50%	
MERIS/MGVI Regional	EOLI-SA
execution time: 8 and 10 hours by day of the selected period reliability: upper than 95% availability: upper than 78%	execution time: inferior to 1 min 30 sec reliability: upper than 80% availability: upper than 80%

tion illustrated in Figure 8.4. With this application, the provider is able to notice the growth of the cluster utilization. The cluster is allowed to work without delays within the execution duration advertised at a fixed level of charge. When the charge moves beyond this level, the provider notifies the SLA manager. We observe the effect produced by two different increases: the first case is an increase of the charge of the computing cluster for 20%; the second involves an increase of 50%.

The *second step* of the adaptation process amounts to spread context modifications. The dependencies are directly related to an increase of the execution charge, i.e., *Dep 2, Dep 3, Dep 6. Dep 2* induces an increase of the execution time and *Dep 3* leads to a decrease of the availability of the MERIS/MGVI service. The *Dep 6* leads to a decrease of the EOLI-SA availability. This decrease enables *Dep 5*, which refers to a decrease of reliability. The effects of an increase of the execution charge on service context of both services are: an increase of the execution time and a decrease of the availability for the MERIS/MGVI service; and a decrease of the availability and the reliability for the EOLI-SA service. The web service contexts of both services resulting from increases of 20% and 50% are illustrated in Table 8.4.

The *third step* of the SLA manager's process is the compatibility checking between user requirements and the provider's capabilities. With an increase of 20%, the MERIS/MGVI capabilities still meet user requirements. The EOLI-SA is also facing the user expectations with this increase of the execution charge. With an increase of 50%, MERIS/MGVI does not meet the constraint on the maximum allowed execution time, so that the compatibility does not verify. In contrast, the EOLI-SA service is still facing the user requirements and does not break any constraint of the user.

The *fourth step* of the SLA manager is the adaptation. With an increase of 20%, the MERIS/MGVI service is in scenario 2; it is compatible with user requirements but breaches the initial SLA: its execution time is above 6 hours by day of the selected period. The SLA between the user and the provider must be renegotiated. This new SLA is illustrated in Table 8.5. The EOLI-SA service respects the scenario 1; it is still compatible with user requirements and the initial SLA is still applicable. The initial SLA is preserved between the user and the provider. With an incresing of the execution charge of 50%, the MERIS/MGVI service follows the scenario 3. The service fails to meet the constraint stating that the execution time must be inferior to 7 hours by day of the selected period. The SLA manager selects another service in the services cluster that is able to meet user requirements. The EOLI-SA is in the scenario 2; it is compatible with user expectations but the *SLO 3* of its SLA is breached, the availability is inferior to 92%. A new SLA is negotiated between the service user and the provider, it is illustrated in Table 8.5.

8.5 Related Work

Adaptation to failures and SLA violations has received attention [17; 83; 154; 196]. However, the influence of context on SLA adaptation has not been studied in depth. Analyzing the impact of context variations on software behavior is a problem outlined in various other areas such as computer human interaction [47], pervasive computing [150; 223] and autonomic systems [22; 196]. Context-sensitivity is usually defined as an application software system's ability to sense and analyze context from various sources. It lets appli-

Table 8.5: SLAs resulting from the increasing of the execution charge

New SLA established for the MERIS/MGVI service with an increasing of the execution charge of 20%	
SLO 1:	the execution time must be between 6h and 7h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
New SLA established for the EOLI-SA service with an increasing of the execution charge of 50%	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 80%

cation software take different actions adaptively in different contexts [150]. In response to these changes, several adaptation strategies exist [40; 129]. Among them, In et al. [83] outline the problem caused by QoS of situation-aware applications. The relationships between changes of situations and resources required to support the desired level of QoS is not clear. They solve this problem with a situation-aware middleware able to predict all QoS requirements of the applications and to analyze tradeoff relationships among the different QoS requirements. The resource availability may be changed according to dynamically varying situations. Such changes in QoS requirements and QoS constraint violations are identified by their middleware that resolves conflicts by rescheduling resources for supporting high priority missions. In contrast to this model, our proposal relies on an existing definition of dependencies between context elements. Moreover, in the web service area, all resources cannot be modified or rescheduled or are even out of the scope of the service provider or the service user. Our model adapts SLA to context changes and does not intervene on the context elements to comply to QoS requirements. Tosic [204] proposes an alternative to custom-made SLA, the utilization of Web Service Offerings which is supported by an infrastructure (WSOI) and a specific language (WSOL) [206]. Each service is proposed with some classes of service that differ in usage privileges, service priorities, response time guaranteed and verbosity of response information. Their approach cuts off the negotiation problem between the service provider and the service user. However, such predefined classes of service only allow a discrete variation of QoS offered to the service user. Classes of services are predefined, limiting their number and therefore the adaptation possibilities. The Appendix C provides more information about SLA management.

8.6 Conclusions

The management of SLAs between an user and a provider in the context of web services is essential to enable autonomy of web service executions. It allows an automatic resolution of conflicts occuring after web service failures or updated expectations of the user. The first advantage of our method is the reliance on the identification of context elements and existing dependencies between these context elements. The context and dependencies allow the SLA manager to anticipate problems. The modifications of context elements are reported to the SLA manager by the provider and the user before the service is executed by the service user. Thus, the SLA manager is able to anticipate and adapt consequently the existing SLA or establish a contract with a new provider. The second advantage of our method is that the SLA manager tries to preserve the existing contract between the service provider and the service user. Long term collaborations between stakeholders are protected from the continuous switching over existing services and new services are selected only when the current provider is not able to meet the user expectations.

Part IV

Conclusions

Chapter 9

Conclusions

This chapter presents the conclusions of this work. Section 9.1 outlines the main ideas proposed in this thesis. Section 9.2 presents the main contributions of this thesis. Section 9.3 outlines the limitations of this work and Section 9.4 proposes some directions for future work.

9.1 Summary

Service-Oriented Computing (SOC) is now an accepted and known paradigm. SOC is supported by several specification languages and technologies enabling its adoption by scientists and practitioners. However, SOC is an emergent paradigm with large ongoing research activities. Among others, the management and monitoring of services could be improved. Service management spans a range of activities, from installation and configuration to collecting metrics and tuning, to ensure responsive service execution. It typically involves gathering information about the managed-service framework, services and business processes, and managed-resource status and performance via root-cause failure analysis, SLA monitoring and reporting, service deployment, and life-cycle management and capacity planning [163]. To manage services, one possibility is to investigate the quality of services and to use them in the web services monitoring. QoS allow to discriminate services and to observe the performance of a service. We propose models and methods enabling the management of web services through their QoS characteristics. This thesis involves different steps:

- The definition of a **Quality of Service Specification Model** enabling services stakeholders to express their expectations and capabilities about QoS to meet and to offer. The specification model proposed provides modeling constructs about the measurement of quality properties, the desired values of quality properties, the interdependencies between values of distinct quality dimensions and, the priorities between QoS properties to take into consideration during optimization.

- The suggestion of some **QoS-driven management methods** enabling to improve the management of web services before the service execution. Proposed management methods (i.e., selection, composition and user profiling) rely on specifications made with the quality model previously defined. The selection, the composition and the user profiling methods are driven by quality expectations and capabilities specified by involved stakeholders. Such methods allow to customize the services execution from user requirements. The service proposed to the service user are those that best fill its expectations.

- The proposition of **processes enabling the management of service level agreements** during the service execution. Service Level Agreements specify contracts between stakeholders of the composition about quality levels expected. SLA are essential to guarantee the right execution of services and to prevent from opportunistic behavior. Proposed processes enable to monitor and control the contract achievement. These processes also allow to take self-healing measures if the initial contract fails.

9.2 Main Contributions

This work contributes to the improvement of web services management by offering different models and methods supporting a quality driven management. The main contributions are:

- A quality model to allow requesters to specify quality expectations, providers to advertise service qualities and management third parties to compare alternative services. Upon basis of observed similarities between various existing quality models, we review these and integrate them into a single quality model called QVDP. (Chapter 3)
- A service selection method based on quality properties of a service. By drawing on multi-criteria decision making techniques, we suggest a service selection framework that uses information previously specified with the QoS model in order to select the most appropriate service at runtime. (Chapter 4)
- A service composition method based on the QoS users' expectations and providers' capabilities. The composition method proposes an approach to the computation of individual services' aggregate QoS ratings when multiple QoS criteria are given, and a reinforcement learning algorithm which uses these ratings in order to find the selection of services that maximizes the overall QoS level delivered to the users. (Chapter 5)
- A user profiling method relying on the formulation of QoS users' expectations. A service provider's reputation is a function of the feedback, given after every past transaction by the users of the service. To obtain a reliable reputation score from honest feedback, it is necessary to account for the bias arising from each user's outlook (i.e., tendency to provide overly optimistic or pessimistic feedbacks), sensitivity to deception (tendency to react positively or negatively with weak or strong importance to differences between the quality level expected and delivered), and sensitivity to brand image (tendency to react positively or negatively with weak or strong importance to the image of a provider). These form together the feedback profile specific to a user. We propose a probabilistic model able to compute the feedback profile of each user. (Chapter 6)
- A normative management of service level agreements defined between web services. Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. We outline a norm-oriented multiagent system (NoMAS) architecture that is combined with the service-oriented architecture in order to support the definition, management, and control of SLAs between the service clients and service providers. (Chapter 7)
- We propose a method using context to manage SLA during service executions. SLAs define the service users' Quality of Service (QoS) requirements that the service provider should satisfy. Requirements defined once may not be satisfiable when the context of the web services changes (e.g., when requirements or resource availability changes). Changes in the context can make SLAs obsolete, making SLA revision necessary. We propose a method to autonomously monitor the services' context, and adapt SLAs to avoid obsolescence thereof. (Chapter 8)

9.3 Limitations

Beyond contributions proposed in this thesis, our work presents some limitations described here:

- The different models and methods outlined do not provide a complete framework of service management. Papazoglou enumerates services management activities as gathering information about the managed service platform, services and business processes and managed-resource status and performance via root-cause failure analysis, SLA monitoring and reporting, service deployment, life-cycle management and capacity planning [163]. The service management is confronted to numerous issues and we choose to address some of them in this thesis. The models and methods presented offer a

consistent set as they refer to the management of web services from an user approach. The different activities are driven by quality and by user expectations about quality values. All the contributions presented in this work aim at maximizing the user satisfaction about services executions.

- The overall work proposed in this thesis suffers from a lack of integrated validation. However, it is possible to validate most proposed methods with help of comparative analyzes with other existing models or through the use of appropriate case studies. Most methods have then been individually validated, although some proposed validations lack of applied experimentations. The quality model proposed in Chapter 3 of Part I has been validated through a case study developed in collaboration with the European Space Agency. Moreover, we have compared the expressiveness of the proposed model with other existing quality model in Table 3.1. The methods improving the management of web services presented in Part II propose quantitative analyzes. The composition method (Chapter 5) and the user profiling method (Chapter 6) outline a quantitative validation with comparisons to similar state-of-the-art methods. The selection method (Chapter 4) has not been compared to existing methods. The selection method proposed relies on Multi-Criteria-Decision-Making which is subject to a particular context of utilization. Indeed, existing MCDM methods are usually differentiated by theirs inputs, each method relying on different information. E.g.: to fix criteria priorities, some methods use weights determined by users, other computes weights with order relationships or quantitative comparisons. It is then difficult to compare alternative methods to determine the most efficient. The most adapted method is the one that best fits to the problem. Some existing selection approaches introduce quantitative experiments. However, the proposed experiments concern simulations about execution time [120; 229] or comparisons between the proposed service selection method and a random selection [69; 120; 203]. Conducting such experiments do not allow us to conclude that our proposed selection model is the one that maximize user satisfaction. The models proposed to improve the management of Service Level Agreements in Part III (Chapter 7 and 8) suffer from a lack of validation. They propose an illustration of the introduced concepts through a case study but do not evaluate them. To properly validate proposed models, they should be implemented but especially integrated into existing web services platform. This way, their performance and efficiency could be appropriately measured. Finally, the integration of proposed methods into a complete framework has not been evaluated. Practically, the development needed to enable experiences and get significant results is important and involves several development resources. The validation strategy to conduct a such evaluation is detailed in Subsection 9.3.1. The lack of existing validation for the whole proposed models is also induced by the difficulty to find appropriate data. Some existing datasets propose quality information of web services [5; 232]. However, these datasets do not provide us relevant information to perform an evaluation of our methods. It is then difficult to conduct significative experiments about service quality and requester behavior.

9.3.1 Validation strategy

To set up a global validation of our work, we describe here the validation strategy that should be adopted. The aim of a such validation is to highlight the efficiency of proposed methods to improve the management of Web services. To carry out this validation, we should conduct some surveys comparing the utilization of our management methods with usual methods. The aim of this survey is to evaluate the set of contributions according different criteria such as usability, efficiency, improvement, conformance to requester's expectations, etc. The validation should be performed in several steps:

1. **Development of the Web service management framework.** To allow an integrated validation of this work, proposed contributions should be programmed into an integrated framework. The development should be in accordance to Web service standards to assure interoperability. Methods proposed in this thesis should be offered as a framework enabling selection, composition, profiling and SLA management. This framework will act as layer above the Web services managed by the

framework. Rather than accessing directly to Web services, Web services are accessed through the framework which monitors their execution capabilities and behavior.

2. **Utilization of the framework.** The framework has to be evaluated by some testers (people with relevant experience in Web service management and people with low experience in Web services utilization) assuming the role of services requester's. The framework enables service requesters to specify their quality expectations about executions and to use proposed management methods. Testers follow a set of predefined scenarios of Web services utilizations. Some scenarios relies on the framework while the others are driven by usual or manual management methods. The design of scenarios must take into account several contexts and volumes of Web services utilization.

3. **Surveys on frameworks users.** Once testers have tried the framework through scenarios, they should answer to a questionnaire enabling to evaluate the framework. Testers should respond to issues related to the usability, the efficiency and the improvements offered by the framework. The survey results will be get through a statistical evaluation of answers.

A such validation should allow to highlight the improvement of our method in comparison to usual or manual management methods. However, a such validation depends on the framework development. The experiments could be biased by the design and the usability of the framework. The survey has to be conducted on the underlying methods and not on the framework development.

9.4 Future Work

The work proposed in this thesis covers some of the existing issues of web services management. However, the proposed framework does not address all issues related to the quality management of web services. Among remaining issues, the following are highlighted:

- Ensuring the specification compatibility between users requirements and providers capabilities. With the QoS model proposed, providers and users are both able to define their own quality metrics. However, to ensure the compatibility between providers and users specifications, a semantic match is needed. Confronting offered possibilities to expected values requires a compatibility between specifications of values. The measurement functions used by stakeholders must compute the same values and quality characteristic must have the same semantic. Upon basis of the proposed QoS model, we could propose a framework enabling semantic match between users requirements and providers capabilities.

- Automatic generation of SLA. In our work, we did not give any recommendation about generation of contracts between stakeholders. With service management methods, we determine the services most adapted to the execution according to user requirements and expectations. With our SLA management processes, we propose to adapt the SLA to quality variations and to use normative agents to control SLA executions. Upon basis of optimization results obtained with selection and composition methods, we could automatically generate SLAs between users and selected providers. Information about alternative services could also be used to adapt the SLA if the selected service fails its execution.

- Autonomic Service Composition. In our service composition method, the possible composition is given by a directed acyclic hypergraph. This representation is subject to a preliminary manual matching. The integrity of a service composition imposes a matching of its operations with those of its constituent component services. It imposes semantic constraints on the component services and guarantees that constraints on data that component services exchange are satisfied [163]. This matching could be automatized to improve the efficiency and the usability os services compositions.

- The definition of an appropriate language supporting the proposed work. Web services technologies are supported by several description languages enabling the communication between stakeholders.

In this work, we propose several methods and processes improving the management of web services. However, we did not integrate our proposals into any existing specification languages. The use of such languages is essential to allow the deployment of our proposals with existing technologies.

- The complete integration of proposed models and methods into existing web services technologies. To allow the utilization of methods and processes proposed in our work, these must be accessed with appropriate technologies. The methods we present could be integrated into existing standards (the Sun ONE application framework[1], Microsoft.NET[2], the Oracle BPEL process manager[3], or the IBM WebSphere Application Server[4]). Existing web services repositories could be used to monitor services executions and collect historical quality data of services executions. The cloud computing and its possibilities of services hosting offers large possibilities of services management.

- Defining web services communities according to user expectations. The management of web services could be improved with the use of web services communities. Web services communities offer services clusters assuming quality similarities. Service belonging to such clusters improve their reliability and self-healing capabilities. If one service of the cluster is down, it can be substituted by another member of the cluster.

[1] http://www.sun.com/software/products/application_framework/home_app_framework.xml
[2] http://www.microsoft.com/net/
[3] http://www.oracle.com/technology/products/ias/bpel/index.html,
[4] http://www-01.ibm.com/software/webservers/appserv/was/

Part V

Appendices

Appendix A

Service Selection Approaches

This appendix introduces the related work of the service selection approach proposed in Chapter 4. It is inspired by a service selection survey proposed by Yu and Reiff-Marganiec [226]. The fundamental issues of service selection are: (1) specifying requester's service requirements, (2) evaluating service proposals, and (3) aggregating the evaluation results into a comparable unit. Requester's requirements and the service offerings have both functional and complex non-functional aspects, which need to be expressed and matched against each other to be evaluated.

There exists numerous service selection approaches based on non-functional properties. These propositions present some overlap in functionalities but also have some divergences in their foundations. Such approaches can be differentiated according different criteria allowing to apply the most suited solutions to specific applications.

A.1 Service Selection Criteria

Yu and Reiff-Marganiec define 7 main criteria involved in the evaluation of selection approaches. These criteria are introduced here (further details can be found in [226]):

- **Model for non-functional properties:** service requesters need to objectively distinguish services based on their non-functional criteria to make the most appropriate choice amongst a number of services functionally equivalent. To this aim, a model for non-functional properties is required.
- **Hierarchical properties:** non-functional properties at a lower level are the most relevant to the selection issue. So, it is meaningful to place properties into a hierarchical structure.
- **User preferences:** service requesters usually have varying preferences for the non-functional criteria depending on their situation. The selection mechanism should allow them to express values for each property and represent the relationships between preferences.
- **Evaluation of properties:** it is difficult to predict how many non-functional properties will be available, and additionally the type of these properties. The evaluation framework must on one hand adapt to varying numbers of criteria, but also automatically identify the measurement methods that should be used to evaluate each criteria.
- **Dynamic aggregation:** when all desired non-functional criteria have been evaluated, these must be aggregated into an individual score to gain a final score for the service.
- **Automation:** the ultimate goal of service selection research is to provide fully automatic processes. A service designer would still specify data for the service when making it available, and a user would still be able to specify requirements, but the selection would be performed without human intervention.

- **Scalability and accuracy** Scalability means that the approach can consider large numbers of properties, but also that many ranking processes are taking place simultaneously. Of course there is also a question as to how accurate the result is. While one would aim for perfect accuracy (that is one has provably chosen the best service), it is often sufficient to choose a good enough service if the decision can be made quickly.

A.2 Categorization of Approaches

Yu and Reiff-Marganiec categorizes service selection approaches. Such categories are introduced here (further details can be found in [226]):

- **Policy vs reputation:** policy based service selection approaches allow to specify the non-functional requirements by coding these in a QoS policy model or policy language [92; 124]. In contrast to the policy based approaches, there are a number of approaches based on trust and reputation [59; 135].
- **UDDI extensions vs semantic web:** in order to address the problem of modeling and using services properties, some research projects have investigated the extensions to UDDI and Semantic Web service technologies [4; 188]. Such approaches do not offer extensible service quality model, meaning that the approaches are restricted for selections based on the few predefined, generic criteria. Understanding these disadvantages, some work has been conducted to define the nonfunctional models for web services using Semantic Web Service (SWS) technology. Wang et al. introduced the WSMO (Web service ModelingOntology) [213] based approach. Manikrao et al. [132] introduce the DAML-S based service selection approach. The matching algorithm uses the semantics of the vocabulary by introducing advanced concepts of matching.
- **Graphic preference modeling vs ontology based preference modeling:** It is important to consider how users can best express their needs. To that end, a graphical preference modeling and service selection approach has been discussed in [185], where the preferences are modeled as TCP network graph [23] or UCP network graph [20]. The selection algorithm that is presented is based on simple textual matching without making use of a model for non-functional properties and hence is less extensible and cannot deal with hierarchically structured properties. Maximilien et al. [133] and Lamparter [120] present two approaches which use ontology modeling techniques to model both the requesters requirements and service properties.

	NFP model	User preferences	Evaluation of properties	Hierarchical properties	Dynamic aggregation	Automation	Scalability
[4]	unknown	average	no	yes	low	average	high
[59]	yes	low	no	yes	low	low	high
[92]	no	average	no	no	low	low	low
[120]	yes	average	no	yes	average	high	low
[124]	yes	average	no	yes	low	low	average
[132]	yes	low	yes	yes	high	average	average
[133]	yes	average	no	yes	low	high	high
[135]	no	low	no	yes	average	low	high
[185]	yes	high	no	no	high	low	low
[188]	no	average	no	no	average	average	low
[213]	yes	low	no	yes	average	average	high

The previous table presents the comparison of investigated methods according criteria presented in Section A.1. The comparison illustrates that most of the approaches are lacking flexible methods for evaluating properties. Moreover, the level of expressing meaningful preferences is still low. However, the use of semantic web/ontology technologies has a huge advantage for addressing preference modeling and services non-functional properties. By analyzing the current service selection approaches, we found that most of them are designed by focusing on one or few aspects of the overall service selection problem.

Appendix B

Service Composition Approaches

This appendix presents the state-of-the-art of service composition approaches. It is inspired by the survey made by Rao and Su [174].

The service composition issue comes from different sources: (1) the growing number of services available over the Web; Web services can be created and updated on the fly. (2) The composition needs to consider the updating at runtime and must adapt its behavior. (3) Web services are developed by different organizations using different models to describe the services and their concepts. These does not exist a unique language to evaluate the Web services in an identical means.

Most existing web service composition approaches are inspired by cross enterprise workflows and AI planning research areas. Section B.1 prensents composition approaches using workflows while Section B.2 outlines composition approaches relying on AI planning.

B.1 Services Composition using Workflows

We briefly introduce here the service composition approaches relying on workflows, more details can be found in Rao and Su [174]. In the workflow-based composition methods, we should distinguish the static and dynamic workflow generations. The static one means that the requester should build an abstract process model before the composition planning starts. EFlow [32] is a platform for the specification, enactment and management of composite services. EFlow uses a static workflow generation method. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The authors further refine the service composition platform and propose a prototype of composite service definition language(CSDL) [33]. An interesting feature of CSDL is that it distinguishes between invocation of services and operations within a service. It provides the adaptive and dynamic features to cope with the rapidly evolving business and IT environment in which Web services are executed. Polymorphic Process Model (PPM) [186] uses a method that combines the static and dynamic service composition.

B.2 Services Composition using Artificial Intelligence Planning

Many research efforts tackling Web service composition problem via Artificial Iintelligence (AI) planning have been reported. Rao and Su classify the methods into five categories [174], namely, the situation calculus, the Planning Domain Definition Language (PDDL), rule-based planning, the theorem proving and others.

- **Situation calculus:** McIlraith et. al. [139; 140; 152] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints.

- **PDDL:** A strong interest to Web service composition from AI planning community could be explained roughly by similarities between DAML-S and PDDL representations. PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover, since DAML-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, DAML-S descriptions could be translated to PDDL format. Then different planners could be exploited for further service synthesis. In presenting the Web service composition method based on PDDL, McDermott [137] introduces a new type of knowledge, called value of an action, which persists and which is not treated as a truth literal. FromWeb service construction perspective, the feature enables us to distinguish the information transformation and the state change produced by the execution of the service.

- **Rule-based planning:** Medjahed [142] presents a technique to generate composite services from high level declarative description. The method uses composability rules to determine whether two services are composable. The composition approach consists of four phases. First, the specification phase enables high-level description of the desired compositions using a language called Composite Service Specification Language(CSSL). Second, the matchmaking phase uses composability rules to generate composition plans that conform to service requesters specifications. The third phase is selection phase. If more than one plan is generated, in the selection phase, the service requester selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). The final phase is the generation phase. SWORD [170] is another developer toolkit for building composite Web services using rule-based plan generation. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the Web services. In SWORD, a service is modeled by its preconditions and postconditions. They are specified in a world model that consists of entities and relationships among entities. A Web service is represented in the form of a Horn rule that denotes the postconditions are achieve if the preconditions are true. To create a composite service, the service requester only needs specify the initial and final states for the composite service, then the plan generation can be achieved using a rule-based expert system.

- **Other AI-planning methods:** Some other AI planning techniques are proposed for the automatic composition of Web services. The SHOP2 planner [221] is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is an Hierarchical Task Network(HTN) planner. The authors believe that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The authors also claim that the HTN planner is more efficient than other planning language, such as Golog. In their paper, the authors give a very detail description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way.

Appendix C

Service Level Agreement Management Approaches

This appendix outlines the related work of Service Level Agreements approaches. It is inspired by the survey made by Seidel et al. [187]. Currently, three main SLAs specification frameworks are established: the Web Service Level Agreement (WSLA) developed by IBM [109], the Web Service Agreement (WS-Agreement) developed by the Open Grid Forum (OGF) [128] and SLang [119].

C.1 WSLA

WSLA [109] is a framework developed by IBM for specifying and monitoring Service Level Agreements (SLA) for Web Services. The framework is able to measure and monitor the QoS parameters of a Web Service and reports violations to the contract specified in the SLA. In a Web Service environment, services are usually subscribed dynamically and on demand. In this environment automatic SLA monitoring and enforcement helps to fulfill the requirements of both service providers and requesters. WSLA provides a formal language based on XML Schema to express SLAs and a runtime architecture which is able to interpret this language. Components of the runtime architecture can be outsourced to third parties to ensure a maximum of objectivity. The WSLA language allows service requesters and providers to define SLAs and their parameters and specify how they are measured. The WSLA monitoring services are automatically configured to enforce an SLA upon receipt.

C.2 WS-Agreement

The Web Services Agreement Specification [128] from the Open Grid Forum (OGF) describes a protocol for establishing an agreement on the usage of Services between providers and requesters. It defines a language and a protocol to represent the services of providers, create agreements and monitor agreements compliance at runtime. An agreement defines a relationship between two parties that is dynamically established and managed. In the agreement each party agrees on its respective roles, rights and obligations. A provider offers a service according to conditions described in the SLA. The agreement specifies the availability and ther QoS characteristics offered by the provider. Agreements can be negotiated by entities acting on behalf the provider and/or the requester. An agreement creation process usually consists of three steps: (1) The initiator retrieves a template from the responder, which advertises the types of offers the responder is willing to accept. (2) The initiator makes an offer. (3) The offer is either accepted or rejected by the responder. An agreement consists of the agreement name, its context and the agreement terms. The context describes information about the involved parties and meta-data such as the duration of the agreement. Agreement terms define Service Level Objectives (SLOs), which describe the quality of service aspects of the service that have to be fulfilled by the provider. The Web Services Agreement

Specification allows the usage of any standard or domain specific condition expression language to define SLOs. The specification of domain-specific term languages is explicitly left open.

C.3 SLang

SLang is a reference model for inter-organizational service provision at storage, network, middleware and application level [119]. SLang meets multiple objectives: (1) it provides a format for the negotiation of QoS properties; (2) the means to capture these properties unambiguously for inclusion in contractual agreements; and, (3) a language appropriate as input for automated reasoning systems of QoS aware adaptive middleware. SLang associate QoS measures to identifiable architectural elements of Application Service Providers (ASP). The QoS semantic of SLang does not refer to a specific ASP model and are defined according to diverse domains of the performance perspectives. For example, the throughput of a database server and the throughput of a component container server are different concepts: the former is defined in terms of the query response time, the latter in terms of the roundtrip method invocations per second. Similarly, SLang allows to adapt the QoS syntax to the reference domain. SLang adopts classification of SLA according to vertical and horizontal SLAs. Horizontal SLAs govern the interaction between coordinated pairs whereas, vertical SLAs are dedicated to interactions between subordinated pairs, within the service provision architecture stack. The main requirements fulfilled by the SLang language are:

- Parametrization : each SLA includes a set of parameters, the values that quantitatively describe a service.
- Compositionality: a service can be the result of a cooperation between different domain entities. An SLA language has to enable such composition.
- Validation: Before initiating an SLA, contractors have to be able to validate it, checks its syntax and consistency, further verified as a result of a composition.
- Monitoring: parties should be able to automatically monitor the extent of which the service levels set forth in an agreement are actually provided by its providers.
- Enforcement: Once service levels are agreed, network routers, database management systems, middleware and web servers can be extended to enforce service levels in an automated manner by using techniques such as caching, replication, clustering and farming.

Bibliography

[1] Y. Achbany, I. J. Jureta, S. Faulkner, and F. Fouss. Continually learning optimal allocations of services to tasks. *IEEE Trans. Serv. Comput.*, 1(3):141–154, 2008. 58

[2] T. Akamatsu. Cyclic flows, markov process and stochastic traffic assignment. *Transportation Research B*, 30(5):369–386, 1996. 63

[3] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. G-qosm: Grid service discovery using qos properties. *Journal of Computing and Informatics (Special issue on Grid Computing)*, 21(4):363–382, 2002. 106

[4] E. Al-Masri and Q. H. Mahmoud. Discovering the best web service. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1257–1258, New York, NY, USA, 2007. ACM. 130

[5] E. Al-Masri and Q. H. Mahmoud. Investigating web services on the world wide web. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 795–804, New York, NY, USA, 2008. ACM. 123

[6] T. Amemiya. Tobit models: A survey. *Journal of Econometrics*, 24(1-2):3–61, 1984. 79

[7] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen. Wsdl-based automatic test case generation for web services testing. In *SOSE '05: Proceedings of the IEEE International Workshop*, pages 215–220, Washington, DC, USA, 2005. IEEE Computer Society. 16

[8] C. Bana e Costa and J. Vansnick. Macbeth - an interactive path towards the construction of cardinal value fonctions. *International transactions in operational Research*, 1:489–500, 1994. 54, 55

[9] A. C. Barbosa, J. Sauvé, W. Cirne, and M. Carelli. Evaluating architectures for independently auditing service level agreements. *Future Gener. Comput. Syst.*, 22(7):721–731, 2006. 106

[10] V. R. Basili and H. D. Rombach. The tame project: Towards improvement-oriented software environments. *IEEE Trans. Software Eng.*, 14(6):758–773, 1988. 39

[11] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet. Semantic web services framework (swsf). Technical report, http://www.daml.org/services/swsf/1.0/, 2005. 18

[12] BEA, IBM, Microsoft, and S. AG. Web services policy framework (ws-policy). Technical report, May 2003. 1

[13] A. Benharref, R. H. Glitho, and R. Dssouli. Mobile agents for testing web services in next generation networks. In *MATA*, pages 182–191, 2005. 16

[14] A. Benharref, M. A. Serhani, M. Salem, and R. Dssouli. *Managing Web Service Quality*, chapter Multi-Tier Framework for Management of Web Services Quality, pages 23 – 47. Khan, Khaled M., 2009. 15

[15] D. Berardi, D. Calvanese, G. D. Giacomo, and L. U. D. Bolzano/bozen. Automatic service composition based on behavioral descriptions. *International Journal of Cooperative Information Systems*, 14:2005, 2005. 66

[16] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001. 17

[17] D. Bianculli, R. Jurca, W. Binder, C. Ghezzi, and B. Faltings. Automated dynamic maintenance of composite services based on service reputation. In Springer-Verlag, editor, *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 449 – 455, Berlin, Heidelberg, 2007. 117

[18] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merrit. *Characteristics of Software Quality*. North-Holland, 1978. 17

[19] G. Boella, L. Torre, and H. Verhagen. Introduction to normative multiagent systems. *Comput. Math. Organ. Theory*, 12(2-3):71–79, 2006. 101

[20] C. Boutilier, F. Bacchus, and R. I. Brafman. *UCP-Networks: A Directed Graphical Representation of Conditional Utilities*, pages 56–64. 2001. 130

[21] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3c note, World Wide Web Consortium, May 2000. See `http://www.w3.org/TR/SOAP/`. 15

[22] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 28–33, New York, NY, USA, 2004. ACM. 115, 117

[23] R. I. Brafman, C. Domshlak, and S. E. Shimony. On graphical modeling of preference and importance. *J. Artif. Intell. Res. (JAIR)*, 25:389–424, 2006. 39, 130

[24] G. Brahnmath, R. R. Raje, A. Olson, M. Auguston, B. R. Bryant, and C. C. Burt. A quality of service catalogue for software components. In *Proceedings of the Southeastern Software Engineering Conference*, 2002. 26

[25] J. Brans and P. Vincke. A preference ranking organization method: The PROMETHEE method for multiple criteria decision-making. *Management Science*, 31(6):647–656, 1985. 51, 52, 55

[26] P. Brockhoff and I. Skovgaard. Modelling individual differences between assessors in sensory evaluations. *Food Quality and Preference*, 5:215–224, 1994. 77, 80, 82, 83, 86

[27] R. G. Brown. *Smoothing, forecasting and prediction of discrete time series*. Prentice-hall, 1962. 63

[28] Business objective driven REliable and Intelligent grids for real busiNess (BREIN project). Final brein architecture d4.1.3 v2 - wp 4.1 architectural design. Technical report, 2009. 107

[29] A. Campbell, G. Coulson, and D. Hutchison. A quality of service architecture. *SIGCOMM Comput. Commun. Rev.*, 24(2):6–27, 1994. 106

[30] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM. 66

[31] C. Cappiello, M. Comuzzi, and P. Plebani. On automated generation of web service level agreements. In *CAiSE*, pages 264–278, 2007. 99, 114

[32] F. Casati, S. Ilnicki, L.-j. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 13–31, London, UK, 2000. Springer-Verlag. 131

[33] F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing e-services. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 171–186, London, UK, 2001. Springer-Verlag. 131

[34] F. Casati, E. Shan, U. Dayal, and M.-C. Shan. Business-oriented management of web services. *Commun. ACM*, 46(10):55–60, 2003. 15

[35] C. Castelfranchi and Y.-H. Tan, editors. *Trust and deception in virtual societies*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. 100, 101

[36] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, 27(6):365–389, 2002. 39

[37] W.-N. Chen and J. Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(1):29–43, Jan. 2009. 66

[38] L. Cherkasova, Y. Fu, W. Tang, and A. Vahdat. Measuring and characterizing end-to-end internet service performance. *ACM Trans. Internet Technol.*, 3(4):347–391, 2003. 100

[39] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl). Technical report, W3C, 2001. 27

[40] M. A. Cibrán, B. Verheecke, W. Vanderperren, D. Suvée, and V. Jonckers. Aspect-oriented programming for dynamic web service selection, integration and management. *World Wide Web*, 10(3):211–242, 2007. 118

[41] A. D'Ambrogio. A model-driven wsdl extension for describing the qos of web services. In *Proceedings of the International Conference on Web Services (ICWS'06)*, 2006. 27, 45

[42] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *ACM Conference on Electronic Commerce*, pages 150–157, 2000. 72, 89

[43] C. Dellarocas. Reputation mechanism design in online trading environments with pure moral hazard. *Information Systems Research*, 16(2):209–230, 2005. 89

[44] C. Dellarocas. Strategic manipulation of internet opinion forums: Implications for consumers and firms. *Manage. Sci.*, 52(10):1577–1593, 2006. 84

[45] C. Dellarocas and C. A. Wood. The Sound of Silence in Online Feedback: Estimating Trading Risks in the Presence of Reporting Bias. *Management Sci.*, 2007. 89

[46] W. E. Deming. *Quality, productivity, and competitive position.* Massachusets Institute of Technology, Center for Advanced Engineering Study, 1982. 17

[47] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166, 2001. 112, 117

[48] Distributed Management Task Force, Inc. (DMTF). Web services for management (ws-management) specification. Technical report, 2008. 1

[49] W.-L. Dong, H. Yu, and Y.-B. Zhang. Testing bpel-based web service composition using high-level petri nets. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pages 441–444, Washington, DC, USA, 2006. IEEE Computer Society. 58

[50] D. M. F. Dignum. Towards socially sophisticated bdi agents. In *ICMAS '00: Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, page 111, Washington, DC, USA, 2000. IEEE Computer Society. 101

[51] J. A. Farrell and H. Kreger. Web servces management approaches. *IBM Systems Journal*, 41(2):212–227, 2002. 15

[52] A. V. Feigenbaum. *Quality control: Principles, practice, and administration.* McGraw-Hill, 1951. 17

[53] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* PWS Publishing Co., Boston, MA, USA, 1998. 16

[54] D. F. Ferguson and M. L. Stockton. Service-oriented architecture: Programming model and product architecture. *IBM Systems Journal*, 44(4):753–780, 2005. 17

[55] J. Figueira, S. Greco, and M. Ehrgott. *Multiple Criteria Decision Analysis: State of the Art Surveys.* Springer Verlag, Boston, Dordrecht, London, 2005. 51, 52, 53, 55, 61, 75

[56] J. Figueira and B. Roy. Determining the weights of criteria in the electre type methods with a revised simos' procedure. *European Journal of Operational Research*, 139:317–326(10), 1 June 2002. 54, 75

[57] F. Fouss, Y. Achbany, and M. Saerens. A probabilistic reputation model. *Information Sciences*, 2010. 78, 79

[58] S. Frolund and J. Koistinen. Qml: A language for quality of service specification. Technical report, HP Laboratories, Palo Alto, California, 1998. 26

[59] S. Galizia, A. Gugliotta, and J. Domingue. A trust based methodology for web service selection. In *ICSC '07: Proceedings of the International Conference on Semantic Computing*, 2007. 130

[60] D. Garvin. What does product quality really mean? *Sloan Management Review*, 26(1):25–43, 1984. 13

[61] D. Garvin. *Managing quality: The strategic and competitive edge.* Free Press, 1988. 17

[62] T. Gilb. *Principles of software engineering management.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988. 14

[63] W. M. Goldstein. Judgments of relative importance in decision making: Global vs local interpretations of subjective weight. *Organizational Behavior and Human Decision Processes*, 47(2):313–336, December 1990. 54

[64] R. E. Goodin. Structures of mutual obligations. *J. of Social Policy*, 31(4):579–596, 2002. 99, 100

[65] A. R. Gray and S. G. MacDonell. A comparison of techniques for developing predictive models of software metrics. *Information & Software Technology*, 39(6):425–437, 1997. 16

[66] P. Green. Bayesian reconstructions from emission tomography data using a modified em algorithm. *Medical Imaging, IEEE Transactions on*, 9(1):84–93, Mar 1990. 78, 79, 80

[67] P. J. Green. On use of the em for penalized likelihood estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 52(3):443–452, 1990. 79

[68] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993. 18

[69] X. Gu and K. Nahrstedt. A scalable qos-aware service aggregation model for peer-to-peer computing grids. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 73, Washington, DC, USA, 2002. IEEE Computer Society. 46, 66, 123

[70] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4):293–333, 1996. 58

[71] M. He, N. R. Jennings, and H.-F. Leung. On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985–1003, 2003. 107

[72] C. Herssens, S. Faulkner, F. Fouss, and I. J. Jureta. A framework for qos driven selection of services. In *SCC '08: Proceedings of the IEEE Service Computing Conference*, volume 2, pages 537–538, Los Alamitos, CA, USA, 2008. IEEE Computer Society. 43, 45

[73] C. Herssens, S. Faulkner, and I. J. Jureta. Context-driven autonomic adaptation of sla. In *ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing*, Lecture Notes in Computer Science, pages 362–377, Berlin, Heidelberg, 2008. Springer-Verlag. 95, 109

[74] C. Herssens, I. J. Jureta, and S. Faulkner. Capturing and using qos relationships to improve service selection. In *CAiSE 08: Proceedings of the Conference on Advanced Information System Engineering*, pages 312–327, 2008. 37, 38, 39, 43, 101

[75] C. Herssens, I. J. Jureta, and S. Faulkner. Dealing with quality tradeoffs during service selection. In *ICAC '08: Proceedings of the IEEE International Conference on Autonomic Computing*, volume 0, pages 77–86, Los Alamitos, CA, USA, 2008. IEEE Computer Society. 37, 38, 39, 43, 116

[76] Hewlett Packard (HP). Hp software & solutions. Technical report, 2007. 15

[77] J. Hokkanen and P. Salminen. Choosing a solid waste management system using multicriteria decision analysis. *European Journal of Operational Research*, 98(1):19–36, April 1997. 54

[78] J. Hokkanen, P. Salminen, E. Rossi, and M. Ettala. The choice of a solid waste management system using the electre ii decision-aid method. *Waste Management and Research*, 13:175–193, 1995. 61, 75

[79] I. Horrocks. DAML+OIL: a description logic for the semantic web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002. 18

[80] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. Technical report, W3C, 2003. 25

[81] C. Hwang and K. Yoon. *Multiple attribute decision making : Methods and applications.* Springer-Verlag, 1981. 55, 56, 61, 75

[82] IEEE. *Software Engineering Standards.* IEEE, 1989. 17

[83] H. P. In, C. Kim, and S. S. Yau. Q-mar: An adaptive qos management model for situation-aware middleware. In L. T. Yang, M. Guo, G. R. Gao, and N. K. Jha, editors, *EUC*, volume 3207 of *Lecture Notes in Computer Science*, pages 972–981. Springer, 2004. 106, 117, 118

[84] International Business Machines (IBM). Web service level agreement (wsla) language specification. Technical report, 2003. 99, 101

[85] International Organization for Standardization (ISO). *ISO 8402 Quality management and quality assurance - Vocabulary.* 1986. 17

[86] International Organization for Standardization (ISO). Cd15935 information technology: Open distributed processing—reference model—quality of service. Technical report, 1998. 27

[87] International Organization for Standardization (ISO). Quality management systems - fundamentals and vocabulary, iso 9000-2000. Technical report, 2000. 14

[88] International Organization for Standardization (ISO). Quality management systems - guidelines for performance improvements, iso 9004-2000. Technical report, 2000. 14

[89] International Organization for Standardization (ISO). Quality management systems - requirements, iso 9001-2000. Technical report, 2000. 14

[90] K. Ishikawa. *What is total quality control? The Japanese way.* Prentice Hall, 1985. 17

[91] M. C. Jaeger, G. Mühl, and S. Golze. Qos-aware composition of web services: An evaluation of selection algorithms. In *OTM Conferences (1)*, pages 646–661, 2005. 66

[92] H. Janicke and M. Solanki. Policy driven service discovery. In *Workshop on Service oriented Computing*, 2007. 130

[93] W. S. Jevons. *The Theory of Political Economy*, chapter Theory of Utility. 1965. 105

[94] Y. Jiang, C.-K. Tham, and C.-C. Ko. Challenges and approaches in providing qos monitoring. *Int. J. Netw. Manag.*, 10(6):323–334, 2000. 16

[95] C. Julien. Adaptive preference specifications for application sessions. In *ICSOC 2006, 4th International Conference on Service-Oriented Computing*, pages 78–89, 2006. 112

[96] H.-W. Jung and B. Choi. Optimization models for quality and cost of modular software systems. *European Journal of Operational Research*, 112(3):613–619, February 1999. 55

[97] J. M. Juran. *Quality control handbook.* McGraw-Hill, 1951. 17

[98] R. Jurca and B. Faltings. Obtaining reliable feedback for sanctioning reputation mechanisms. *J. Artif. Intell. Res. (JAIR)*, 29:391–419, 2007. 72, 89

[99] I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *RE '08: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, 2008. 38

[100] I. J. Jureta, S. Faulkner, Y. Achbany, and M. Saerens. Dynamic task allocation wihin an open service-oriented mas architecture. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'07)*, 2007. 18, 38, 40

[101] I. J. Jureta, S. Faulkner, Y. Achbany, and M. Saerens. Dynamic web service composition within a service-oriented architecture. In *Proceedings of the International Conference on Web Services (ICWS'07)*, 2007. 18, 38, 40, 67

[102] I. J. Jureta, S. Faulkner, and P. Thiran. Dynamic requirements specification for adaptable and open service-oriented systems. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'07)*, 2007. 39

[103] I. J. Jureta, C. Herssens, and S. Faulkner. A comprehensive quality model for service-oriented systems. *Software Quality Journal*, 17(1):65–98, 2009. 11, 17, 60

[104] S. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003. 64

[105] S. Kalepu, S. Krishnaswamy, and S. W. Loke. Verity: a qos metric for selecting web services and providers. In *Proceedings of Web Information Systems Engineering Workshops*, pages 131–139, 2003. 16

[106] H. Kaminski and M. Perry. Sla automated negotiation manager for computing services. In *CEC/EEE '06: IEEE Int. Conf. E-Commerce Tech*, 2006. 109

[107] H. Kaminski and M. Perry. *Emerging Web Services Technology*, chapter Employing Intelligent Agents to Automate SLA Creation, pages 33–46. Springer-Verlag, 2007. 100

[108] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, 1993. 55

[109] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network Systems Management*, 11(1), 2003. 2, 28, 97, 99, 102, 106, 133

[110] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003. 17

[111] C. Keum, S. Kang, I.-Y. Ko, J. Baik, and Y.-I. Choi. Generating test cases for web services using extended finite state machine. In *TestCom 2006: Proceedings of 18th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems,*, volume 3964, pages 103–117. Springer, 2006. 16

[112] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *Conference on Advanced Information Systems Engineering*, pages 431–445, 2000. 58

[113] B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. *IEEE Softw.*, 13(1):12–21, 1996. 14, 73

[114] J. M. Ko, C. O. Kim, and I.-H. Kwon. Quality-of-service oriented web service composition algorithm and planning architecture. *J. Syst. Softw.*, 81(11):2079–2090, 2008. 66

[115] M. J. Kollingbaum. *Norm-governed Practical Reasoning Agents*. PhD thesis, University of Aberdeen, 2005. 101, 102

[116] M. J. Kollingbaum and T. J. Norman. Supervised interaction - a form of contract management to create trust between agents. In *Trust, Reputation, and Security*, pages 108–122, 2002. 102

[117] T. Korkmaz and M. Krunz. A randomized algorithm for finding a path subject to multiple qos constraints. *Computer Networks*, 36:1694–1698, 1999. 66

[118] V. S. Lai, B. K. Wong, and W. Cheung. Group decision making in a multiple criteria environment: A case using the ahp in software selection. *European Journal of Operational Research*, 137(1):134–144, February 2002. 55

[119] D. D. Lamanna, J. Skene, and W. Emmerich. Slang: A language for defining service level agreements. volume 0, page 100, Los Alamitos, CA, USA, 2003. IEEE Computer Society. 133, 134

[120] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm. Preference-based selection of highly configurable web services. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1013–1022, New York, NY, USA, 2007. ACM. 123, 130

[121] P. Laureti, L. Moret, Y. C. Zhang, and Y. K. Yu. Information filtering via iterative refinement. *Europhysics Letters*, 75:1006, 2006. 80, 83, 84

[122] F. Leymann. Web services flow language (wsfl 1.0). Technical report, IBM, May 2001. 1

[123] J. Liu, N. Gu, Y. Zong, Z. Ding, S. Zhang, and Q. Zhang. Web services automatic composition based on qos. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 607–610, Washington, DC, USA, 2005. IEEE Computer Society. 66

[124] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004. ACM. 56, 130

[125] P. Lockemann, J. Nimis, L. Braubach, A. Pokahr, and W. Lamersdorf. *Multiagent Engineering*, chapter Architectural Design, pages 405–429. Springer-Verlag, 2006. 99

[126] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proceedings of the International Symposium on Object-Oriented Real-Time Distributed Computing*, 1998. 26

[127] H. Ludwig, R. Dan, A. Franck, A. Keller, and R. P. King. Web service level agreement (wsla) language specification. Technical report, IBM Corporation, 2003. 109, 114

[128] H. Ludwig, T. Nakata, O. Wldrich, P. Wieder, and W. Ziegler. Reliable orchestration of resources using ws-agreement. In *Proceedings of the 2006 International Conference on High Performance Computing and Communications , (HPCC06)*, volume 4208 of *LNCS*, pages 753 – 762, Munich, September 2006. Springer. 133

[129] S. A. Lundesgaard, K. Lund, and E. F. Utilising alternative application configurations in context- and qos- aware mobile middleware. In *DAIS '06: Distributed Applications and Interoperable Syst.*, 2006. 118

[130] Z. Maamar, S. Mostefaoui, and Y. H. Toward an agent-based and context-oriented approach for web services composition. *IEEE Trans. Knowl. and Data Eng.*, 17:686 – 697, 2005. 112

[131] A. Mani and A. Nagarajan. Understanding quality of service for web services. Technical report, IBM, Developerworks web site, 2002. 14

[132] U. S. Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, page 117, Washington, DC, USA, 2005. IEEE Computer Society. 130

[133] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004. 130

[134] E. M. Maximilien and M. P. Singh. Toward autonomic services trust and selection. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'04)*, 2004. 29, 46, 115, 116

[135] E. M. Maximilien and M. P. Singh. Multiagent system for dynamic web services selection. In *Proceedings of the AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE)*, 2005. 130

[136] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. volume i. concepts and definitions of software quality. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA, 1977. 14

[137] D. V. McDermott. Estimated-regression planning for interactions with web services. In *AIPS*, pages 204–211, 2002. 132

[138] S. A. McIlraith and D. L. Martin. Bringing semantics to web services. *IEEE Intelligent Systems*, 18(1):90–93, 2003. 17

[139] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, 2002. 131

[140] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001. 18, 131

[141] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley, 1997. 78, 80

[142] B. Medjahed, A. Bougettaya, and A. K. Elmagarmid. Composing web services on the semantic web. *VLDB Journal*, 12:333–351, 2003. 132

[143] D. A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002. 2, 45, 97, 109

[144] G. Morgan, S. Parkin, C. Molina-Jimenez, and J. Skene. *Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government*, chapter Monitoring Middleware for Service Level Agreements in Heterogeneous Environments, pages 79–93. Springer-Verlag, 2005. 100

[145] V. Mousseau. Eliciting information concerning the relative importance of criteria. In P. Pardalos, Y. Siskos, and C. Zopounidis, editors, *Advances in Multicriteria Analysis*, pages 17–43. Kluwer Academic, Dordrecht, 1995. 54, 61, 75

[146] L. Mui, M. Mohtashemi, and A. Halberstadt. Notions of reputation in multi-agents systems: a review. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 280–287, New York, NY, USA, 2002. ACM. 100

[147] C. Muldoon, G. OHare, D. Phelan, R. Strahan, and C. R. Access: An agent architecture for ubiquitous service delivery. In *CIA '03: Int. Worksh. Cooperative Info. Agents*, 2003. 111

[148] J. Myerson. Use slas in a web services context, part 1: Guarantee your web service with a sla. Technical report, IBM, 2004. 109, 111, 112

[149] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web services security: Soap message security 1.1 (ws-security 2004), February 2006. 1

[150] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. Qos-aware middleware for ubiquitous and heterogeneous environments. *Comm. Mag.*, 19(11):140–148, 2001. 117, 118

[151] S. Naik and P. Tripathy. *Software Testing and Quality Assurance: Theory and Practice*. Wiley-Spektrum, 2008. 13, 14

[152] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the International Conference on the World Wide Web (WWW 2002)*, 2002. 131

[153] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven integration of heterogenous information systems. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 447–458, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. 56, 66

[154] M. A. Netto, K. Bubendorfer, and R. Buyya. Sla-based advance reservations with flexible and adaptive time qos parameters. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 119–131, Berlin, Heidelberg, 2007. Springer-Verlag. 117

[155] OASIS (Advanced open standards for the information society. Uddi version 3.0.2. Technical report, 2004. 1

[156] Object Management Group (OMG). The corba trading services. Technical report, 1997. 26

[157] Object Management Group (OMG). Uml profile for modeling qos and fault tolerance characteristics and mechanisms specification, v1.0. Technical report, May 2005. 4, 18, 27, 30, 40, 48

[158] L. Osterweil. Strategic directions in software quality. *ACM Comput. Surv.*, 28(4):738–750, 1996. 17

[159] J. O'Sullivan, D. Edmond, and A. ter Hofstede. What's in a service? *Distributed and Parallel Databases*, 12(2):117–133, 2002. 45

[160] G. Oulsnam. Unravelling unstructured programs. *Comput. J.*, 25(3):379–387, 1982. 58

[161] M. Ouzzani and A. Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004. 66

[162] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. *Commun. ACM*, 46(10):24–28, 2003. 1, 17, 45, 57, 109

[163] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007. 1, 121, 122, 124

[164] Parasoft. Soatest. Technical report, 2006. 15

[165] A. Parasuraman and V. A. Zeithaml. Understanding and improving service quality: A literature review and research agenda. In B. Weitz and R. Wensley, editors, *Handbook of Marketing*. Sage Publications, 2006. 73

[166] A. Parasuraman, V. A. Zeithaml, and L. L. Berry. A Conceptual Model of Service Quality and Its Implications for Future Research. *Journal of Marketing*, 49(4):41–50, 1985. 73

[167] S. Park, L. Liu, C. Pu, M. Srivatsa, and J. Zhang. Resilient trust management for web service integration. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 499–506, Washington, DC, USA, 2005. IEEE Computer Society. 90

[168] A. Paschke, J. Dietrich, and K. Kuhla. A logic based sla management framework. In *SWPC '05: Semantic Web Policy Workshop at ISWC '05*, 2005. 102, 106

[169] J. Pitt, P. Venkataram, and A. Mamdani. Qos management in manets using norm-governed agent societies. In *ESAW*, pages 221–240, 2005. 107

[170] S. Ponnekanti and A. Fox. Sword: A developer toolkit for building composite web services. In *Proc. Alternate Tracks of the 11th World Wide Web Conf.*, 2002. 132

[171] R. S. Poston. Using and fixing biased rating schemes. *Commun. ACM*, 51(9):105–109, 2008. 89

[172] L. Qiu, L. Chang, and Z. Lin, F. andShi. Context optimization of ai planning for semantic web services composition. *Service Oriented Comput. and Applications*, 1(2):117–128, 2007. 111, 112

[173] S. Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003. 16, 20

[174] J. Rao and X. Su. A survey of automated web service composition methods. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, pages 43–54, 2004. 131

[175] C. A. Reeves and D. A. Bednar. Defining quality: Alternatives and implications. *The Academy of Management Review, Special Issue: Total Quality*, 19(3):419–445, July 1994. 17, 73

[176] B. Roy and V. Mousseau. A theoretical framework for analysing the notion of relative importance of criteria. *Journal of Multi-Criteria Decision Analysis*, 5:145–159, 1996. 54

[177] T. Saaty. *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New york, 1980. 51, 54, 61, 75

[178] T. L. Saaty. Axiomatic foundation of the analytic hierarchy process. *Manage. Sci.*, 32(7):841–855, 1986. 54, 55

[179] M. Saerens, Y. Achbany, F. Fouss, and L. Yen. Randomized shortest-path problems: Two related models. *Neural Computation*, 21(8):2363–2404, 2009. 62, 63, 67

[180] A. Sahai. Automated sla monitoring for web services. In *DSOM '02: IFIP/IEEE Int. Worksh. Distrib. Syst.: Operations and Management*, 2002. 97, 99, 109, 114

[181] A. Sahai, A. Durante, and V. Machiraju. Towards automated sla management for web services. Technical report, Hewlett-Packard Laboratories, 2002. 114

[182] W. J. Salamon and D. R. Wallace. Quality characteristics and metrics for reusable software. Technical report, National Institute of Standards and Technology, 1994. 16

[183] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM. 86, 87

[184] A. Schmietendorf, R. Dumke, and D. Reitz. Sla management - challenges in the context of web-service-based infrastructures. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 606, Washington, DC, USA, 2004. IEEE Computer Society. 16

[185] C. Schröpfer, M. Binshtok, S. E. Shimony, A. Dayan, R. Brafman, P. Offermann, and O. Holschke. Introducing preferences over nfps into service selection in soa. In *Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers*, pages 68–79, Berlin, Heidelberg, 2007. Springer-Verlag. 130

[186] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based nd reference process-based multi-enterprise processes. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 247–263, London, UK, 2000. Springer-Verlag. 131

[187] J. Seidel, O. Wldrich, P. Wieder, R. Yahyapour, and W. Ziegler. Using sla for resource management and scheduling - a survey. In D. Talia, R. Yahyapour, and W. Ziegler, editors, *Grid Middleware and Services - Challenges and Solutions*, CoreGRID Series. Springer, 2008. Also published as CoreGRID Technical Report TR-0096. 133

[188] Y.-J. Seo, H.-Y. Jeong, and Y.-J. Song. A study on web services selection method based on the negotiation through quality broker: A maut-based approach. In *ICESS*, pages 65–73, 2004. 130

[189] M. A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui. A qos broker based architecture for efficient web services selection. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 113–120, Washington, DC, USA, 2005. IEEE Computer Society. 16

[190] N. Shadbolt, T. Berners-Lee, and W. Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006. 17

[191] S. E. Shaikh and N. Mehandjiev. Multi-attribute negotiation in e-business process composition. In *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 141–146, Washington, DC, USA, 2004. IEEE Computer Society. 55

[192] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. *Applications and the Internet Workshops, IEEE/IPSJ International Symposium on*, 0:85, 2003. 16

[193] C. Sierra and F. Dignum. Agent-mediated electronic commerce: Scientific and technological roadmap. In *Agent Mediated Electronic Commerce*, pages 1–18, 2001. 107

[194] J. Simos. *Gestion des Déchets Solides Urbains Genevois : Les Faits, le Traitement, l'Analyse*. Presses Polytechniques et Universitaires Romandes, Lausanne, 1990. 54, 61, 75

[195] J. Skene, D. D. Lamanna, and W. Emmerich. Precise service level agreements. In *Proceedings of the International Conference on Software Engineering (ICSE'04)*, 2004. 26

[196] L. Skorin-Kapov, , and M. Matijasevic. Dynamic qos negotiation and adaptation for networked virtual reality services. In *WOWMOM '05: IEEE Int. Symp. World of Wireless Mobile and Multimedia Networks*, 2005. 117

[197] J. C. Spall. *Introduction to stochastic search and optimization*. Wiley, 2003. 63, 64

[198] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004. 18

[199] R. Staehli, F. Eliassen, J. O. Aagedal, and G. Blair. Quality of service semantics for component-based systems. In *Proceedings of the International Conference on Reflective and Adaptive Middleware Systems*, 2003. 26

[200] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. 62, 63, 64

[201] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck. Coping with inaccurate reputation sources: experimental analysis of a probabilistic trust model. In *AAMAS*, pages 997–1004, 2005. 91

[202] D. Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000. 17

[203] H. Tong and S. Zhang. A fuzzy multi-attribute decision making algorithm for web services selection based on qos. In *APSCC '06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages 51–57, Washington, DC, USA, 2006. IEEE Computer Society. 56, 123

[204] V. Tosic. *Service offerings for xml web services and their management applications*. PhD thesis, Carleton University, 2004. 118

[205] V. Tosic, B. Esfandiari, B. Pagurek, and K. Patel. On requirements for ontologies in management of web services. In *Proceedings of the International Workshop on Web Services, e-Business, and the Semantic Web (WES'02)*, 2002. 26, 102

[206] V. Tosic, B. Pagurek, and K. Patel. Wsol - a language for the formal specification of classes of service for web services. In *ICWS' 03, IEEE Int. Conf. Web Serv.*, 2003. 118

[207] K. Trzec and D. Huljenic. Intelligent agents for qos management. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1405–1412, New York, NY, USA, 2002. ACM. 107

[208] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao. Verification of web services using an enhanced uddi server. *Object-Oriented Real-Time Dependable Systems, IEEE International Workshop on*, 0:131, 2003. 16

[209] V. M. R. Tummala, K. S. Chin, and S. H. Ho. Assessing success factors for implementing ce a case study in hong kong electronics industry by ahp. *International Journal of Production Economics*, 49(3):265–283, May 1997. 55

[210] J. F. Vilas, J. J. P. Arias, and A. F. Vilas. High availability with clusters of web services. In *APWeb*, pages 644–653, 2004. 115

[211] A. E. Walsh. *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference, 2002. 109

[212] C. Wang, G. Wang, H. Wang, A. Chen, and R. Santiago. Quality of service (qos) contract specification, establishment, and monitoring for service level management. In *EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, page 49, Washington, DC, USA, 2006. IEEE Computer Society. 112

[213] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A qos-aware selection model for semantic web services. In *ICSOC*, pages 390–401, 2006. 130

[214] W. C. Wedley, E. U. Choo, and B. Schoner. Magnitude adjustment for ahp benefit/cost ratios. *European Journal of Operational Research*, 133(2):342–351, January 2001. 55

[215] E. W. V. Winterfeldt. *Decision analysis and behavioral research*. Cambridge University Press, 1986. 55

[216] World Wide Web Consortium (W3C). Hypertext transfer protocol – http/1.1. Technical report, 1999. 1

[217] World Wide Web Consortium (W3C). Web services description language (wsdl) 1.1. Technical report, 2001. 1

[218] World Wide Web Consortium (W3C). Web services glossary. Technical report, 2004. 1

[219] World Wide Web Consortium (W3C). Soap version 1.2. Technical report, 2007. 1

[220] World Wide Web Consortium (W3C). Extensible markup language (xml) 1.0 (fifth edition). Technical report, 2008. 1

[221] D. Wu, E. Sirin, J. A. Hendler, D. S. Nau, and B. Parsia. Automatic web services composition using shop2. In *WWW (Posters)*, 2003. 132

[222] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang. Autonomous service level agreement negotiation for service composition provision. *Future Gener. Comput. Syst.*, 23(6):748–759, 2007. 99, 101, 107, 114, 116

[223] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002. 117

[224] B. Yu and M. P. Singh. An evidential model of distributed reputation management. In *AAMAS*, pages 294–301, 2002. 90, 91

[225] B. Yu and M. P. Singh. Detecting deception in reputation management. In *AAMAS*, pages 73–80, 2003. 90, 91

[226] H. Q. Yu and S. Reiff-Marganiec. Non-functional property based service selection: A survey and classification of approaches. In *2nd Workshop on Non Functional Properties and Service Level Agreements in Service Oriented Computing*, 2008. 129, 130

[227] T. Yu and K.-J. Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *ICSOC*, pages 130–143, 2005. 66

[228] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM. 29

[229] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004. 46, 58, 66, 116, 123

[230] L. Zeng, H. Lei, and H. Chang. Monitoring the qos for web services. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, 2007. 40

[231] C. Zhang, R. N. Chang, C.-S. Perng, E. So, C. Tang, and T. Tao. Qos-aware optimization of composite-service fulfillment policy. *Services Computing, IEEE International Conference on*, 0:11–19, 2007. 66

[232] Z. Zheng and M. R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pages 145–152, Washington, DC, USA, 2008. IEEE Computer Society. 123

[233] C. Zhou, L.-T. Chia, and B.-S. Lee. Daml-qos ontology for web services. In *Proceedings of the International Conference on Web Services (ICWS'04)*, 2004. 28