

QoS Awareness and Adaptation in Service Composition

by

Silvana De Gyvés Avila

Submitted in accordance with the requirements for the degree of
Doctor of Philosophy

The University of Leeds
School of Computing

February 2014

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

All material in the following articles is the candidate's own work, under the supervision of the co-author Karim Djemame.

S. De Gyvés and K. Djemame, "A QoS Optimization Model for Service Composition", in Proceedings of the 4th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE'12), Nice, France, July 2012. Sections of this paper are included in chapters 3 and 6.

S. De Gyvés and K. Djemame, "Fuzzy Logic Based QoS Optimization Mechanism for Service Composition", in Proceedings of the 7th IEEE International Symposium on Service Oriented System Engineering (SOSE'13), San Francisco Bay, USA, March 2013. Sections of this paper are included in chapters 4 and 6.

S. De Gyvés and K. Djemame, "Proactive Adaptation in Service Composition Using a Fuzzy Logic Based Optimization Mechanism", in Proceedings of the 4th International Conference on Cloud Computing and Service Science (CLOSER'14), Barcelona, Spain, April 2014. Sections of this paper are included in chapters 5 and 6.

This copy has been supplied on the understanding that it is copyright material and that no quotation from the Thesis may be published without proper acknowledgement.

Acknowledgements

I would like to thank my supervisor Dr Karim Djemame for his guidance, support and advice since the beginning of my studies in the University of Leeds. This work would never have been possible without his help and I cannot adequately express my gratitude for that. Additionally, I would like to thank my examiners Dr. Vania Dimitrova and Dr. Radu Calinescu, for their excellent comments and feedback when examining this Thesis.

I would like to acknowledge the help and technical support that Django Armstrong gave me during the setting of the experimental testbed, and the advice that Richard Kavanagh gave me regarding the improvement of my writing. In the school I found not only nice mates, but very good friends. I would like to thank all those who have been a support to me during the last four years, specially Django, Richard and Asif, for their friendship, all those coffees and endless talks.

Moving abroad was a hard decision and involved big changes in many aspects of my life. I would like to express my heartiest gratitude to my husband Ismael for all his tireless support and patience. He had to deal with my ups and downs during the last four years, and it was definitely not an easy task. I would like to thank my Mom and Dad, Alicia and Gerardo, and my siblings, Pamela, Brenda and Gerardo, for everything they have done for me. My family has been there for me always; cheering me up and encouraging me in every quest I take, this time has been no exception.

Finally, I would like to thank the Mexican National Council on Science and Technology for the scholarship they granted me to perform my PhD studies in United Kingdom.

Abstract

The dynamic nature of a Web service execution environment generates frequent variations in the Quality of Service offered to the consumers, therefore, obtaining the expected results while running a composite service is not guaranteed. When combining this highly changing environment with the increasing emphasis on Quality of Service, management of composite services turns into a time consuming and complicated task. Different approaches and tools have been proposed to mitigate the impacts of unexpected events during the execution of composite services. Among them, self-adaptive proposals have stood out, since they aim to maintain functional and quality levels, by dynamically adapting composite services to the environment conditions, reducing human intervention.

The research presented in this Thesis is centred on self-adaptive properties in service composition, mainly focused on self-optimization. Three models have been proposed to target self-optimization, considering various QoS parameters, the benefit of performing adaptation, and looking at adaptation from two perspectives: reactive and proactive. They target situations where the QoS of the composition is decreasing. Also, they consider situations where a number of the accumulated QoS values, in certain point of the process, are better than expected, providing the possibility of improving other QoS parameters. These approaches have been implemented in service composition frameworks and evaluated through the execution of test cases.

Evaluation was performed by comparing the QoS values gathered from multiple executions of composite services, using the proposed optimization models and a non-adaptive approach. The benefit of adaptation was found a useful value during the decision making process, in order to determine if adaptation was needed or not.

Results show that using optimization mechanisms when executing composite services provide significant improvements in the global QoS values of the compositions. Nevertheless, in some cases there is a trade-off, where one of the measured parameters shows an increment, in order to improve the others.

List of Abbreviations

BPML	Business Process Modelling Language
BPMN	Business Process Model and Notation
DAML-S	DARPA Agent Markup Language for Services
DSS	Decision Support System
JSON	JavaScript Object Notation
LAN	Local Area Network
OCL	Object Constraint Language
OWL	Web Ontology Language
OWL-S	OWL for Services
PDDL	Planning Domain Definition Language
QoS	Quality of Service
REST	REpresentational State Transfer
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
UML	Unified Modelling Language
URL	Uniform Resource Locator
WS-Agreement	Web Services Agreement Specification
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WS-CoL	Web Service Constraint Language
WS-Discovery	Web Services Dynamic Discovery

WS-Policy4MASC	WS-Policy for the Manageable and Adaptive Service Compositions middleware
WAN	Wide Area Network
WSCI	Web Service Choreography Interface
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSLA	Web Service Level Agreement
WSRF	Web Services Resource Framework
XLANG	XML LANGuage
XML	Extensible Markup Language
XPath	XML Path language
XSLT	Extensible Stylesheet Language Transformations

Table of Contents

Acknowledgements	i
Abstract	ii
List of Abbreviations	iii
Table of Contents	v
List of Figures	ix
List of Tables	xii
Chapter 1. Introduction	1
1.1. Research Motivation.....	1
1.2. Aims and Objectives.....	3
1.3. Methodology.....	4
1.4. Research Contributions.....	6
1.5. Assumptions.....	7
1.6. Thesis Outline	8
Chapter 2. Background	10
2.1. Service Oriented Architecture	10
2.2. Web Services	11
2.2.1. Web Service Related Standards	13
2.2.2. Web Service Life Cycle	15
2.2.3. Benefits of Using Web Services	18
2.3. Web Service Composition	18
2.3.1. Composition Languages.....	21
2.3.2. Challenges in Service Composition.....	23
2.4. Quality of Service	25
2.4.1. Service Level Agreements	27
2.4.2. QoS in Service Composition	28
2.4.3. Estimation of QoS Parameters in Service Composition	29
2.4.4. Predictive Algorithms.....	30
2.5. Adaptive Service Composition	32

2.5.1. Autonomic Computing and Self-Adaptive Software.....	32
2.5.2. Adaptation in Service Composition	34
2.5.2.1. Self-Adaptive Properties.....	37
2.5.2.2. The Need for Adaptation in Service Composition	40
2.5.3. Reactive vs Proactive Adaptation.....	40
2.6. Other Adaptive Approaches for Service Composition	41
2.6.1. Late Binding	41
2.6.2. Fault Tolerance	43
2.7. Decision Support Systems	46
2.8. Summary.....	48

Chapter 3. A QoS Optimization Model for Service Composition..... 50

3.1. Motivation.....	50
3.2. Self-Adaptation in Service Composition	52
3.3. Proposed Solution	54
3.3.1. System Model	56
3.3.2. QoS Model	58
3.3.3. Service Selection Model.....	59
3.3.4. QoS Optimization Model	60
3.4. Implementation.....	63
3.4.1. Composition Engine	65
3.4.2. Service Binder	66
3.4.3. Service Repository and Service Selector	67
3.4.4. Predictor	67
3.4.5. Adaptation Manager	68
3.5. Evaluation	69
3.5.1. Test Case.....	69
3.5.2. Service Selection Based on Fixed Weights.....	71
3.5.3. First Stage of Evaluation	71
3.5.3.1. Experimental Environment	72
3.5.3.2. Experimental Results	72
3.5.4. Second Stage of Evaluation	74
3.5.4.1. Experimental Environment	75
3.5.4.2. Experimental Results	76

3.5.5. Discussion.....	78
3.6. Summary.....	80

Chapter 4. A Fuzzy Logic Based QoS Optimization Model for Service Composition..... 81

4.1. Motivation.....	81
4.2. Decision Support Systems in Service Selection and Service Composition	82
4.3. Proposed Solution.....	83
4.3.1. QoS Model and Service Selection Model.....	84
4.3.2. QoS Optimization Model	86
4.3.2.1. Fuzzy Logic Based Decision Support Systems	86
4.3.2.2. QoS Optimization Heuristic	90
4.4. Implementation.....	93
4.5. Evaluation	93
4.5.1. Service Selection Based on Fixed Weights.....	94
4.5.2. Dynamic QoS Parameters.....	94
4.5.2.1. Cost.....	95
4.5.2.2. Energy Consumption.....	96
4.5.3. QoS Parameters Configuration	97
4.5.4. First Stage of Evaluation	98
4.5.4.1. Experimental Results	98
4.5.5. Second Stage of Evaluation	102
4.5.5.1. Experimental Results	103
4.5.6. Discussion.....	107
4.6. Summary.....	108

Chapter 5. A Proactive Adaptation Mechanism for Service Composition 110

5.1. Motivation.....	110
5.2. Proactive Adaptation in Service Composition.....	111
5.3. Proposed Solution.....	113
5.3.1. System Model	113
5.3.2. QoS Model and Service Selection Model.....	115
5.3.3. QoS Optimization Model	117

5.4. Implementation.....	121
5.5. Evaluation	123
5.5.1. Test Cases	123
5.5.2. Service Selection Based on Fixed Weights.....	125
5.5.3. Experiment Description	125
5.5.4. Evaluation Results.....	126
5.5.5. Discussion.....	129
5.6. Summary.....	130
Chapter 6. Comparison, Discussion and Overall Assessment of the Evaluation.....	132
6.1. Overview	132
6.2. Adaptation in Service Composition - Comparison and Discussion.....	134
6.3. Assessment of the Evaluation	137
6.3.1. Overview of the Experiments	138
6.3.2. Analysis of Results	139
6.3.3. Limitations	145
6.4. Summary.....	145
Chapter 7. Conclusion and Future Work.....	147
7.1. Summary.....	147
7.2. Contributions	149
7.3. Future Work	151
Appendix A. Comparison of Predictive Algorithms to Support QoS Estimation	153
Appendix B. Self-Healing Features	156
Appendix C. Fuzzy Rules	158
References.....	162

List of Figures

Figure 2.1. SOA model used by Web services	12
Figure 2.2. Web service standards. (a) WSDL document. (b) SOAP message	14
Figure 2.3. Layered view of XML, SOAP, WSDL and UDDI	15
Figure 2.4. Web services roadmap	17
Figure 2.5. Centralized vs. decentralized service composition.....	19
Figure 2.6. Web service composition - dataflow models	20
Figure 2.7. Service composition languages and standards	22
Figure 2.8. QoS parameters	25
Figure 2.9. Self-Adaptive cycle for service composition	37
Figure 3.1. Composite service example	51
Figure 3.2. Events that can occur at runtime	52
Figure 3.3. Idea of solution.....	55
Figure 3.4. System model	56
Figure 3.5. QoS optimization heuristic.....	62
Figure 3.6. Packages diagram	63
Figure 3.7. Components interaction	64
Figure 3.8. BPEL code that defines the XML input for the service binder ...	66
Figure 3.9. Package diagram - adaptation manager	68
Figure 3.10. Travel planning process	70
Figure 3.11. Experimental environment - LAN	72
Figure 3.12. Response time comparison between variable and fixed weights approaches	73
Figure 3.13. Cost comparison between variable and fixed weights approaches	73
Figure 3.14. Score comparison between variable and fixed weights approaches	74
Figure 3.15. Experimental environment - WAN	75
Figure 3.16. Response time comparison between variable and fixed weights approaches	76
Figure 3.17. Response time evaluation - differences between executions.....	77

Figure 3.18. Cost comparison between variable and fixed weights approaches	77
Figure 3.19. Cost evaluation - differences between executions	78
Figure 3.20. Score comparison between variable and fixed weights approaches	78
Figure 3.21. Execution time for different number of available candidate services	79
Figure 4.1. Basic configuration of fuzzy systems with fuzzifier and defuzzifier	87
Figure 4.2. QoS optimization heuristic.....	92
Figure 4.3. Cost evaluation algorithm.....	95
Figure 4.4. Response time comparison between fuzzy based and fixed weights approaches	99
Figure 4.5. Response time evaluation - differences between executions....	99
Figure 4.6. Cost comparison between fuzzy based and fixed weights approaches	100
Figure 4.7. Cost evaluation - differences between executions	100
Figure 4.8. Energy consumption comparison between fuzzy based and fixed weights approaches.....	101
Figure 4.9. Energy consumption evaluation - differences between executions.....	101
Figure 4.10. Benefit of adaptation per each task in the travel planning process	102
Figure 4.11. Response time comparison between fuzzy based and fixed weights approaches.....	103
Figure 4.12. Response time evaluation - differences between executions.....	104
Figure 4.13. Cost comparison between fuzzy based and fixed weights approaches	104
Figure 4.14. Cost evaluation - differences between executions	105
Figure 4.15. Energy consumption comparison between fuzzy based and fixed weights approaches.....	105
Figure 4.16. Energy consumption evaluation - differences between executions.....	106
Figure 4.17. Benefit of adaptation per each task in the travel planning process	106
Figure 4.18. Execution time for different number of available candidate services	108
Figure 5.1. System model	114

Figure 5.2. QoS evaluation heuristic	120
Figure 5.3. Components interaction	122
Figure 5.4. Test cases. (a) Order booking process. (b) Travel planning process	124
Figure 5.5. Order booking process results. (a) Response time. (b) Energy consumption	127
Figure 5.6. Order booking process results. (a) Availability. (b) Cost	127
Figure 5.7. Summary of results - order booking process.....	128
Figure 5.8. Travel planning process results. (a) Response time. (b) Energy consumption	128
Figure 5.9. Travel planning process results. (a) Availability. (b) Cost.....	129
Figure 5.10. Execution time for different number of available candidate services.....	130
Figure 6.1. Experiment 1A- comparison between variable weights and fixed weights approaches. (a) Response time. (b) Cost.....	140
Figure 6.2. Experiment 1B- comparison between variable weights and fixed weights approaches. (a) Response time. (b) Cost.....	141
Figure 6.3. Experiment 2A- comparison between fuzzy based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption.....	142
Figure 6.4. Experiment 2B- comparison between fuzzy based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption.....	142
Figure 6.5. Experiment 3A- comparison between proactive fuzzy-based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption. (d) Availability	143
Figure 6.6. Experiment 3B- comparison between proactive fuzzy-based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption. (d) Availability	144
Figure A.1. Comparison of estimated values v_s . real (WS1).....	154
Figure A.2. Comparison of estimated values v_s . real (WS2).....	154
Figure B.1. Self-healing evaluation heuristic	157

List of Tables

Table 2.1. Relationship between self-* properties and events/action/goals in service composition	38
Table 2.2. SOA specific faults	43
Table 3.1. Composition tools	65
Table 3.2. QoS parameters configuration.....	70
Table 4.1. Fuzzy variables definition	88
Table 4.2. Benefit of adaptation related fuzzy rules	89
Table 4.3. Power consumption description per node	96
Table 4.4. QoS parameters configuration.....	97
Table 4.5. Results summary.....	103
Table 5.1. Fuzzy variable definition - availability	118
Table 5.2. QoS parameters configuration.....	124
Table 6.1. Adaptation in service composition - part 1.....	134
Table 6.2. Adaptation in service composition - part 2.....	135
Table 6.3. Proactive adaptation/monitoring in service-based systems	136
Table 6.4. QoS parameters configuration.....	139
Table 6.5. Summary of experiments configuration	139
Table C.1. Benefit of adaptation fuzzy rules - extended.....	158

Chapter 1

Introduction

1.1. Research Motivation

Web services are modular, self-contained and reusable software components that rely on XML-based and Web-related standards¹ to support machine-machine interactions over distributed environments [1]. One of the benefits offered by services is time/cost reduction during software development and maintenance. When a single service does not accomplish a consumer's requirement, different services can be used in conjunction to create a new value-added service, known as composite service, to fulfil this requirement.

A composite service provides a new software solution with specific functionalities and can be seen as an atomic component in other service compositions, or as a final solution to be used by a consumer [2]. The process of developing a composite Web service is called service composition. In service composition, it is necessary to have a set of available services that offer certain functionality and also fulfil Quality of Service (QoS) constraints [3]. QoS properties refer to non-functional aspects of Web services, such as performance, reliability, scalability, availability and security [4]. By evaluating the QoS aspects of a set of Web services that share the same goals, a consumer could identify which service meets his quality requirements.

The nature of service composition, dynamicity offered by the environments where services are executed, and growing number of available services (that may provide the same functionality), have brought the need of mechanisms focused not only in enabling automatic/dynamic composition, but also ensuring that the consumer will obtain the expected results when invoking a composite service. To achieve this goal, it is important to consider the QoS aspects of the services involved in the

¹ SOAP, JSON, REST, Thrift, Avro, among others.

composition, as their drawbacks will be inherited by the composite service. However, knowing the QoS of the components is not enough to warranty the behaviour of the composition, as unexpected events may occur at runtime.

In an ideal scenario, all the activities within a composite service (that involve invoking service operations) are executed without problems (i.e. delays, faults, etc.). When the composite service finishes its execution, it has performed all the scheduled tasks, and fulfilled the customer's requirements. However, in the real world the behaviour offered by services exhibits frequent variations, therefore, obtaining the expected results while running a composite service is not guaranteed. As a result, various approaches have been proposed in order to restore and maintain the functional and quality aspects of the composition. Among them, proposals of self-adaptive approaches have stood out, since they aim to provide composite services with capabilities that enable them to morph and function in spite of internal and external changes, searching to maximize the composition potential and reducing as much as possible human involvement.

Self-adaptive mechanisms provide software systems with capabilities to self-heal, self-configure, self-optimize, self-protect, etc., considering the objectives the system should achieve, the causes of adaptation, the system reaction towards change and the impact of adaptation upon the system [5]. Work in self-optimization for service composition has been mainly focused on the selection of services at runtime, in order to maintain the expected QoS of the entire composition. However, it only takes into account situations where QoS decays (e.g. cost increments, performance degradation, etc.), and some of the adaptation strategies apply in the next execution of the composition, or require human specifications.

When different QoS parameters are evaluated within a composite service, and one of them has been enhanced after executing a task (that involves invoking a component service), it is possible to use that leverage to improve other parameters. This can be achieved by applying weights during service selection, giving different priorities to the QoS parameters, which brings the following question:

Q.1. Is there any improvement in the global QoS of a composite service when using variable weights during service selection as part of a self-optimization mechanism?

However, performing adaptation everytime there is a significant variation in the service's behaviour does not ensure upgrading the overall QoS of the composition. Reason why, the benefit of performing adaptation can be considered, bringing the next question:

Q.2. How does the evaluation of the benefit of adaptation influence the adaptation process?

The use of reactive adaptation approaches may lead to increments in response time and cost of composite services. Self-optimization can also be targeted from a proactive perspective, which brings the following question:

Q.3. Does the use of a proactive adaptation approach based on self-optimization helps improving the global QoS of composite services?

To address these questions, the scope of this research is centered in the development of mechanisms that provide a service composition framework with capabilities that help providers in delivering services that satisfy a QoS optimization criteria. These mechanisms react when: the QoS levels of the composition can be improved, the QoS levels of the composition are degraded, a component service is unavailable, and a component service fails. Adaptation has been targeted primarily from a self-optimization perspective, looking at the QoS values of the composition during the different stages of its execution, aiming to improve/maintain the global QoS levels. Changes are applied at Web service level, using service selection strategies combined with dynamic binding.

1.2. Aims and Objectives

The aim of this work is to study QoS awareness and adaptation in the context of service composition, mainly focussing on self-optimization. This is because through self-optimization, composite services seek to restore and maintain their QoS levels. Work related to the provision of self-optimization is focused on the selection of services that provide the most appropriate QoS levels for the composition. The purpose of this research is to design and implement mechanisms that enable self-optimization, where adaptation is not limited to failure prevention and QoS degradation, but also considers

the possibility of improvement in the QoS levels of composite services. Taking into consideration this, the main objectives of this research are:

- *The design of self-optimization mechanisms for service composition.* Design of mechanisms that consider QoS degradation, but also explore situations where a number of the accumulated QoS values of the previous activity in the composite service are better than expected. This will help finding some slack that can be used while selecting the next service in the composition, providing the possibility of improving other QoS parameters.
- *The implementation of QoS aware and adaptive frameworks for service composition.* Through these environments, composite services will be aware of their QoS attributes, and in response to relevant changes on these values, evaluate the need for adaptation, and adapt when needed, in order to satisfy a QoS optimization criteria.

1.3. Methodology

Computational research can be developed and evaluated using three different approaches: mathematical modelling, prototyping and simulation. Mathematical modelling enables researchers to build a representation of a system using mathematic symbols and operations, and based on changes in its variables, estimate the system's behaviour. Prototyping refers to the development of incomplete versions of a product; it allows researchers to analyze and test functionality and design of solution ideas. Simulation is a tool used to imitate or emulate the behaviour of a system; it helps in the development of theories and hypotheses based on observed behaviours when the characteristics of the system have been altered.

The research methodology used in this Thesis was driven from a prototyping point of view, and is conformed by the elements described below.

- *A thorough literature review on self-* properties and adaptation in the context of service composition.* This review is to identify the different self-* properties used in service composition approaches, along with

their methods and objectives. It also helps in finding the relationships between these properties and the events that can occur when executing composite services, the actions the system should take in order to adapt and the goals of adaptation.

- *The identification of limitations within self-optimization approaches.* This is through a detailed analysis of different methods and mechanisms that perform self-optimization in service composition and service-based systems.
- *The design and development of QoS optimization mechanisms.* These mechanisms identify when adaptation is needed during the execution of composite services. This is achieved by analyzing the measured values of QoS parameters at runtime, and comparing them with the QoS objective goals obtained from historical data.
- *The design and development of prototypes.* This is accomplished through the analysis and extension performed on selected features of an open source composition engine. Prototyping helps performing experiments that provide sensible results.
- *An evaluation of the proposed solutions.* This is to assess the results obtained when using the proposed solutions, and compare them with the use of a service selection approach based on fixed weights (described in section 3.5.2) during the execution of composite services.

The use of prototyping in the context of this research enables the development of optimization mechanisms and prototypes from an evolutionary perspective. A prototype that enables the execution of composite services with QoS aware and adaptive capabilities is introduced in chapter 3 (addressing Q.1). It was extended with the model presented in chapter 4, to evaluate the need of performing adaptation (addressing Q.2). The resulted prototype, was modified and extended with the features described in chapter 5, in order to provide proactive adaptation (addressing Q.3).

1.4. Research Contributions

The main contributions of this Thesis are summarized in the following points:

- *QoS optimization mechanisms for service composition.* Three QoS optimization mechanisms are presented in this work. These mechanisms are developed to target QoS degradation and QoS improvement from a global perspective; considering when some of the measured QoS values at certain point of the composite service execution are better than expected, enabling the improvement of other QoS attributes. They use different QoS parameters and were implemented within composition frameworks that provide adaptation from reactive and proactive perspectives. Mechanism one (described in chapter 3) evaluates response time and cost as QoS parameters, and has been implemented in a reactive framework. Mechanism two (described in chapter 4) considers as QoS parameters: response time, cost and energy consumption. It was implemented within a reactive framework. Mechanism three (described in chapter 5) uses response time, cost, energy consumption and availability as QoS parameters, and has been implemented in a proactive framework. Mechanisms two and three use fuzzy logic as a decision making tool. They rely on the benefit of adaptation, value obtained by analyzing the measured QoS attributes of the composition, in order to determine whether adaptation is needed or not.
- *Conceptual frameworks that enable QoS aware and adaptive service composition.* Two abstract systems models are designed to provide a layered structure that enables adaptation from two perspectives: reactive and proactive (described in chapters 3 and 5, respectively). Their main components include: composition engine, adaptation manager, service binder, service selector, predictor and sensors.
- *Prototypes for reactive and proactive service composition.* Two prototypes are implemented as extensions of an open source composition engine. They provide support during the experimental stage, in order to assess the different QoS optimization mechanisms developed along this research. The first prototype enables adaptation from a reactive perspective (see chapter 3), while the second prototype from a proactive perspective (see chapter 5).

- *Discovery of benefits offered by the use of the QoS optimization mechanisms in service composition.* The experiments performed show that the mechanisms are effective and provide significant improvements in terms of global QoS when executing composite services. In some situations a trade-off can be found, where one of the QoS parameters decays in order to maintain/improve the values of the others. A summary of the experimental results can be found in chapter 6.

1.5. Assumptions

The following list contains the main assumptions considered during the development of this work.

- Services are atomic, stateless and their performance is not affected by the input values.
- Services contain only one operation.
- Services are registered correctly in the repository.
- Available services cover all the operations. Per each task of the composite services, there exist at least two component services to invoke.
- At the time of invoking a composite service, the system has available data from previous executions of the different components. If historical data is not available, services will not be selected using predictions, but only based on their functionality.
- Energy consumption is considered as the amount of energy consumed by a server during the time the service is being executed.
- WSDL files contain Web services' QoS information (cost and energy consumption). As this is not part of the standard, WSDL files have been extended to include quality values.
- In the scenarios used during the experimental stage, service malfunction is considered to last for short periods of time.

1.6. Thesis Outline

The remainder of this Thesis is structured as follows:

- *Chapter 2.* Presents a description of relevant topics related to the context of this research, which include: Service Oriented Architecture, Web services and service composition. The definitions of Quality of Service and service level agreements in the context of service environments are provided, followed by the definition of adaptation in service composition. Finally, a list of relevant decision support systems that can be applied during adaptation is presented.
- *Chapter 3.* Describes a QoS optimization model for service composition. It presents the motivation behind its development, along with a discussion on work related to the provision of self-adaptation in service composition. The proposed solution is given, followed by its implementation details. Finally, the experiments performed to evaluate the model and obtained results are discussed in detail.
- *Chapter 4.* Presents a QoS optimization model for service composition based on fuzzy logic. Motivation and a discussion on related approaches are provided. The proposed solution is described, along with its implementation details. Finally, evaluation is presented, covering the experimental setup and the results.
- *Chapter 5.* Introduces a proactive adaptation mechanism for service composition based on fuzzy logic. Motivation towards the development of the approach is given, followed by a review on work related to the provision of proactive adaptation in service composition. The proposed solution is described, along with information regarding implementation. Finally, the experiments performed to evaluate the proposed approach are provided.
- *Chapter 6.* Provides an overall assessment of the evaluation performed to establish the effectiveness of the adaptation approaches presented in chapters 3, 4 and 5. It includes a general overview of the research motivation, a comparison between related work and the research presented in this Thesis, and the assessment of the evaluation. The analysis of the gathered results and their limitations are then discussed.

- *Chapter 7.* Presents a summary of the Thesis on a chapter by chapter basis, major contributions, and a discussion on some directions that can be explored as part of future work.

Chapter 2 Background

This chapter comprises a description of relevant topics related to the context of this research, and provides the main concepts used in this Thesis. Service Oriented Architecture and Web services are defined, and services' background is explored in detail. The concept of service composition is then given, followed by the definitions of Quality of Service and service level agreements in the context of service environments. Adaptation in service composition is then described from the perspective of different mechanisms. Finally, this chapter presents some relevant decision support systems that can be applied during adaptation.

Description of approaches directly related to the mechanisms presented in chapters 3, 4 and 5, and the solution proposed to overcome their limitations is provided in the corresponding chapters. The contribution and novelty of the proposed solutions are discussed in chapter 6.

2.1. Service Oriented Architecture

Service Oriented Architecture (SOA) is a term that represents a model where the logic of an application is decomposed into small and distributed units of logic that exist autonomously, but not isolated from each other. As a group, these units, also known as services, represent a large piece of business automation logic [6]. SOA can be considered as “... a set of principles that define an architecture that is loosely coupled and comprised of service providers and service consumers that interact according to a negotiated contract or interface” [7].

Service Oriented Architecture is a paradigm for designing, developing, managing and organizing services inside a computing environment [7]. It enables applications written in different languages and running on different platforms, to communicate among them and be accessed by the same clients. In other words, SOA principles enable services to be used by other services or programs, as long as they are aware of each other [6].

Because of its flexibility, SOA has been proposed as a method to establish a relationship between information technologies and business requirements. From the IT perspective, some of the benefits that can be achieved by implementing SOA include time reduction, improvements during software development/maintenance, and enterprise application integration. For enterprises, it offers agility to collaborate, agility to adapt, better business operations, improved visibility across organizational data, and ease of introducing new technologies [7], [8].

2.2. Web Services

Web services are self-describing, self-contained, loosely coupled, platform-independent and reusable software components designed to support machine-machine interactions over a network. They can be used in a wide range of applications, from simple requests, to complete business solutions. Consumers can use a single service to accomplish a specific task, or if required, combine multiple services in order to solve a complex problem or conduct a business transaction [9].

Web services are described, published, discovered and invoked in distributed environments through a set of XML-based standards, including WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol) and UDDI (Universal Description Discovery & Integration). Services can also be developed as RESTful applications, without using SOAP and WSDL-based interfaces. RESTful services are considered as resources and identified by their URL's [10]. As a consequence of the use of standards, Web services enable interoperability between applications developed in different programming languages and executed on different platforms.

Some of the characteristics exhibited by a Web service include: functional and non-functional properties, granularity, complexity and synchronicity. Functional properties describe the operational behaviour of the service, while non-functional properties include quality attributes, such as cost, response time, scalability, etc. Granularity and complexity are relative measures of how a service must be in order to provide the required functionality (e.g. fine-grained services address small functionality, coarse-grained services solve complex tasks). Finally, synchronicity is related to the

programming styles used to develop and invoke Web services (synchronous and asynchronous) [9].

According to their capability to keep information from previous executions (state), services can be considered either stateless or stateful. Stateless services do not have the ability to hold state; plain SOAP-based services and RESTful services are stateless. In contrast, stateful services, which use WSRF (Web Services Resource Framework), maintain the state between different invocations through separate entities called resources [11]. Stateless services are used in traditional Web environments, Grid applications and Cloud applications, while stateful services are mostly used in Grid applications.

The Service Oriented Architecture used by Web services consists of three main components: provider, registry and consumer. Figure 2.1 presents an abstract model of this architecture and the relationships between its components.

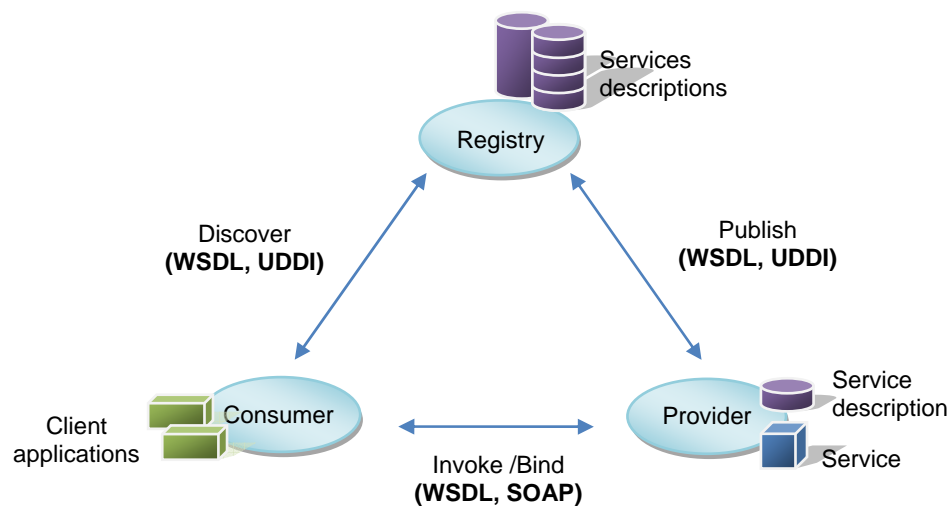


Figure 2.1. SOA model used by Web services.

The provider develops a Web service, generates its description (WSDL) and publishes it in the registry (UDDI), making it available for invocation. The registry contains information to identify the Web service, including an URL that indicates the location of the WSDL file. The consumer queries the registry, finds (discovers) the information of the service that fits its requirements, and uses the corresponding WSDL file to interact with the service through SOAP messages [12].

However, describe, publish, discover and invoke are not the only operations performed in the Web services field. Services can also be monitored and composed. Monitoring Web services involves the use of mechanisms that provide the consumer with information about the execution course and results [13]. While composition enables users/consumers to develop new value-added services by combining existing ones to achieve personalized tasks [14].

2.2.1. Web Service Related Standards

To achieve interoperable integration between heterogeneous applications, Web services are built on a set of widely adopted standards (specifications) proposed by different entities, which include the Organization for the Advancement of Structured Information Standards (OASIS) [15], the World Wide Web Consortium (W3C) [16], the Web Services Interoperability Organization (WS-I) [17] and the Internet Engineering Task Force (IETF) [18].

Most of the Web service related standards are based on XML (eXtensible Markup Language). XML is a simple and flexible text format used to describe data objects called XML documents [19], which play an important role within data-exchange between applications. XML is also a meta-language used to define other markup languages and protocols. Examples of XML-based standards are WSDL, SOAP, WS-BPEL and WS-CDL. Web Services Description Language (WSDL) is the representation language used to describe the public interface details and implementation characteristics of a Web service via WSDL documents (see Figure 2.2a). According to the W3C, a WSDL document “... *defines services as collections of network endpoints, or ports*” [20]. It provides information about the service such as what it does, where it is located and how it is invoked [21]. The elements used in a WSDL document to define a service are [20]:

- *Types*. Container for data type definitions.
- *Message*. Abstract definition of the data being communicated.
- *Operation*. Abstract description of an action supported by the service.
- *Port Type*. Abstract set of operations supported by one or more endpoints.

- *Binding*. Concrete protocol and data format specification for a particular port type.
- *Port*. Single endpoint defined as a combination of a binding and a network address.
- *Service*. Collection of related endpoints.

Interactions between customers and Web services rely on the Simple Object Access Protocol (SOAP). It is an XML-based communication protocol, developed to enable one-way message exchange between nodes (request/response). A SOAP message contains an envelope that includes two sections, header and body (see Figure 2.2b). The header is an optional element and describes complementary information about the message, while the body is mandatory and contains the main data (payload) [22]. SOAP is independent of programming language, operative system and platform [21], which enables interoperability between heterogeneous systems.

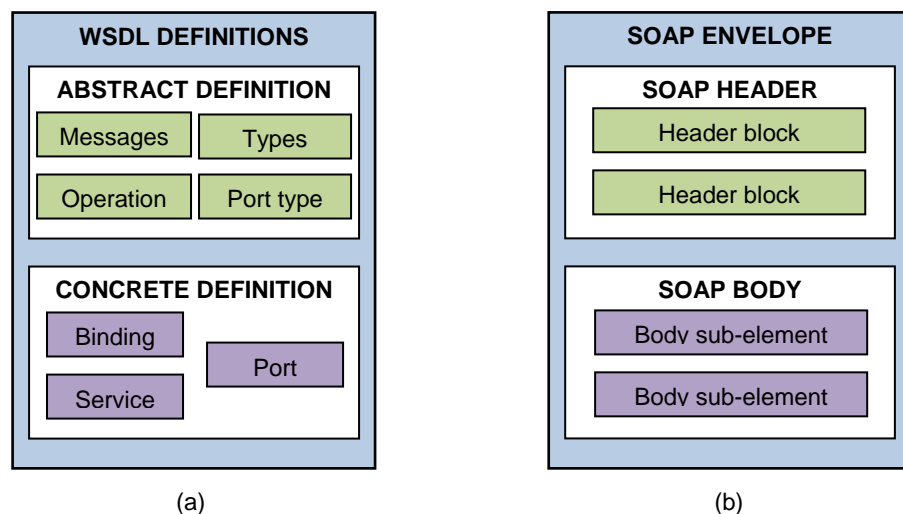


Figure 2.2. Web service standards. (a) WSDL document. (b) SOAP message.

In order to use Web services, customers must know sufficient information to execute them. The Universal Description Discovery & Integration (UDDI) is an initiative to define a set of services to describe and discover service providers, Web services, and the technical information used to access those services. Information in UDDI is represented through business entities, business services, binding templates and tmodels [23]. A UDDI business registry is itself a Web service. Information provided by this registry is classified in three main components [9]:

- *White pages.* Address and key points of contact.
- *Yellow pages.* Information according to industrial classifications.
- *Green pages.* Information of technical capabilities about services.

The layered relationship between XML, WSDL, SOAP and UDDI is shown in Figure 2.3. It can be noted that the UDDI layer works on top of SOAP and WSDL. Both, SOAP and WSDL are built on top of XML, and work using internet protocols (usually HTTP) to enable information exchanges across system boundaries [21]. Even though WSDL, SOAP and UDDI can be considered the core technologies within SOAP-based service environments, there is a large number of standards and specifications focused on diverse areas such as security, interoperability, management and business processes, among others, which enable the development and execution of complex service interactions [24].

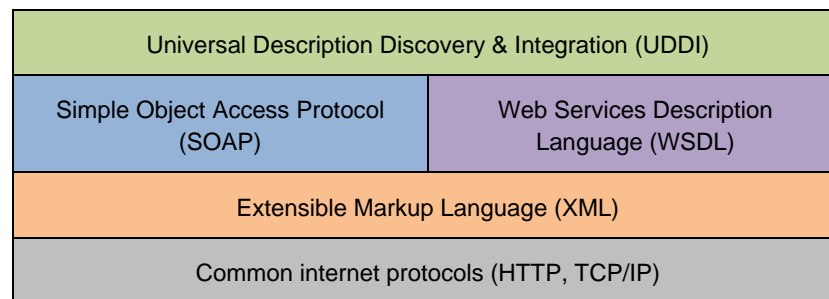


Figure 2.3. Layered view of XML, SOAP, WSDL and UDDI.

Web service development is not limited to the use of SOAP-based standards. Services can also be built using REST. REpresentational State Transfer (REST) is a design style with a stateless client-server architecture. It is not considered a standard; however is widely used due its lightweight infrastructure and presumed simplicity. A RESTful Web service is viewed as a stateless set of resources identified by their URLs [25].

2.2.2. Web Service Life Cycle

In Service Oriented Computing, the Web service life cycle is the foundation for engineering and management activities related to Web services. There are three main entities responsible of performing the different activities that take place during the stages of the service life cycle: service requesters (users, consumers, buyers, customers and their intelligent agents), service

brokers (intermediaries and their agents), and service providers (owners, sellers and their agents) [26].

Typical stages that can be found within a service life cycle are development, publishing, discovery, composing and monitoring [21],[26], [27]. Development comprises not only the creation of the service, but also activities like design, test and deploy. Publishing involves describing and registering in a service registry (UDDI) information about the business, service and its technical information. These two stages are directly related to the service provider. Discovery consists in finding within a service registry a service that provides the desired functionality. During this stage, the service requester interacts with the service broker. The stage of monitoring involves observing the service behaviour. It can be performed by service requesters and service providers. Finally, composition involves the use of different services, combined to provide a specific function. It can be performed by a service requester, but also by a service provider that will expose the resulting composition as a new service.

The growing number of developed services, complexity and time consumed during manual Web services discovery, monitoring and composition, have driven the development of different approaches and methods to perform these operations in an automatic or semi-automatic way. Automatic service discovery involves the implementation of algorithms to query the registries based not only on keywords. Some examples of these methods are described in [28] and [29]. In [28], a semantic-based algorithm is proposed, matching services on semantic relationships at conceptual level. In [29], a QoS-based model that applies QoS properties as constraints while searching services is described.

In the monitoring area, mechanisms are required to provide service users with knowledge about performance, execution and results of the Web services they invoke. Monitoring approaches can apply asynchronous, synchronous, functional and non-functional based techniques in order to obtain information about the service behaviour. For example, the work presented in [30] proposes a policy-based approach to detect exceptions, faults and QoS degradations in composite services during runtime, and uses policies specified in WS-Policy4MASC. An event-based mechanism for monitoring and logging interactions is proposed in [13]; it works with

semantic Web services supported on OWL-S. Composition is a key topic within this research; it will be reviewed in detail in section 2.3.

Information presented in this section is summarized in the roadmap illustrated in Figure 2.4. It is organized according to Web service standards, service classification, stages of the service life cycle and main service uses.

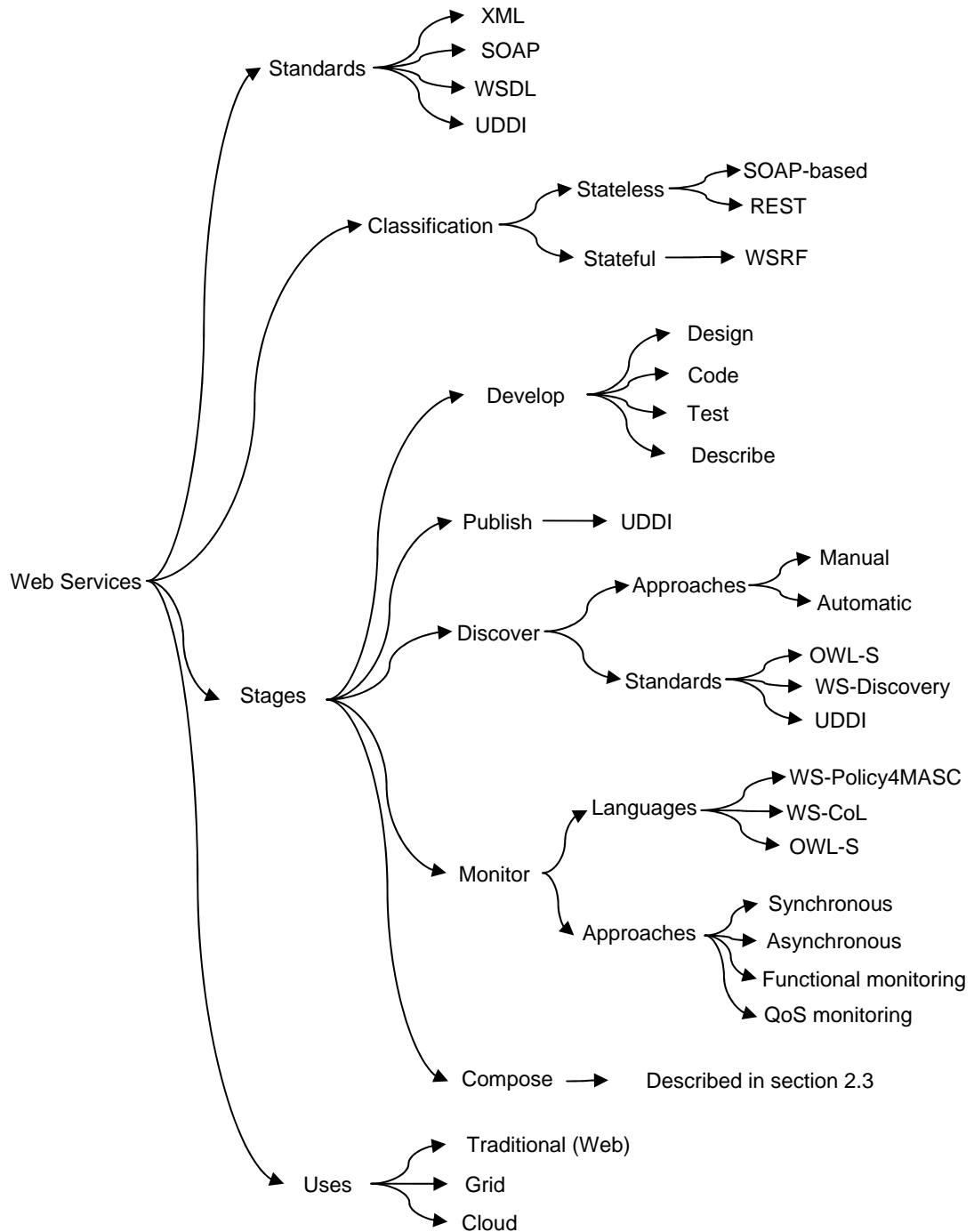


Figure 2.4. Web services roadmap.

2.2.3. Benefits of Using Web Services

During development and execution, Web services exhibit significant benefits when compared to traditional applications:

- *Interoperability.* Services can interact with other services and applications because of the use of standards. They are language and platform independent [21].
- *Ease and fast development.* Development of new services can be done by reusing or combining existing services [21].
- *Decoupling and just-in-time integration.* Services are based on the notion of building applications by discovering and orchestrating network-available services [21].
- *Reduced complexity by encapsulation.* Implementation is not relevant to service consumers, only the functionality provided by the service [21].
- *Self-description.* Services describe their functions, inputs and outputs. They can also describe their non-functional properties (e.g. cost, security, etc.) [9].
- *Ease in management.* Service behaviour can be monitored and managed at any time using external applications, even when the service is not running in an in-house system [9].
- *Brokering.* Services that perform the same tasks can be selected by a broker based on different attributes, such as cost, response time, security, etc. [9].
- *Development tool independence.* Development tools that support Web service standards should be able to invoke a service and access its data [31].

2.3. Web Service Composition

Service composition can be considered as a process that "... involves compiling value-added services from elementary or atomic services to

provide functionalities that were not available or defined at design time” [32]. Two key concepts associated to service composition are orchestration and choreography. Orchestration refers to “... an executable business process that can interact with internal and external services” [33], while choreography is related to coordination protocols and messages exchanged by multiple services, where no single party has full control of the conversation [34].

When a single service does not fulfil the consumer’s requirements, it can be used in conjunction with other services to provide that functionality. The obtained service (also known as composite service) can be used as a complete software solution by the consumer, or can be considered as an atomic service in other compositions.

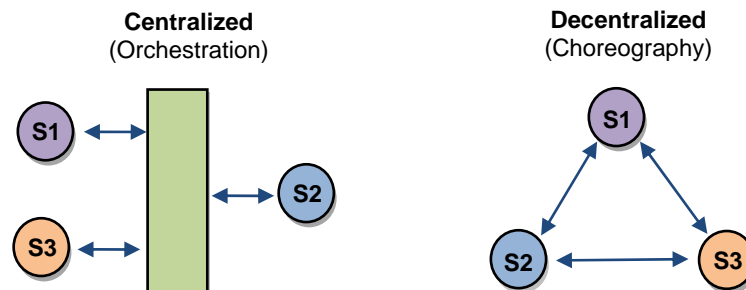


Figure 2.5. Centralized v.s. decentralized service composition.

Dataflow models used in service composition are centralized and decentralized, commonly achieved by orchestration and choreography, respectively (see Figure 2.5). In Web service orchestration, the composition process is always controlled by the perspective of one of the parties, which describes the services interaction at message level, including the business logic and tasks execution [33]. It can be specified by modelling languages like UML activity diagrams and BPMN (Business Process Modelling Notation), and implemented in XML-based languages such as WS-BPEL (Web Services Business Process Execution Language) and WSFL (Web Services Flow Language).

On the other hand, choreography is more collaborative, it allows each party involved to describe its role played in the composition. Choreography is associated with public message exchange and rules of interaction that occur between services [33]. It can be specified in languages like pi-calculus and UML, and implemented in WS-CDL (Web Services Choreography Description Language) and WSCI (Web Service Choreography Interface). Tools such as Taverna [35] and Kepler [36] were developed to implement

orchestration in e-Science projects; ActiveBPEL Designer [37], Oracle jDeveloper [38] and IBM WebSphere [39] to be used in the e-Business field; and pi4soa [40] to develop choreography processes. This information is summarized in Figure 2.6.

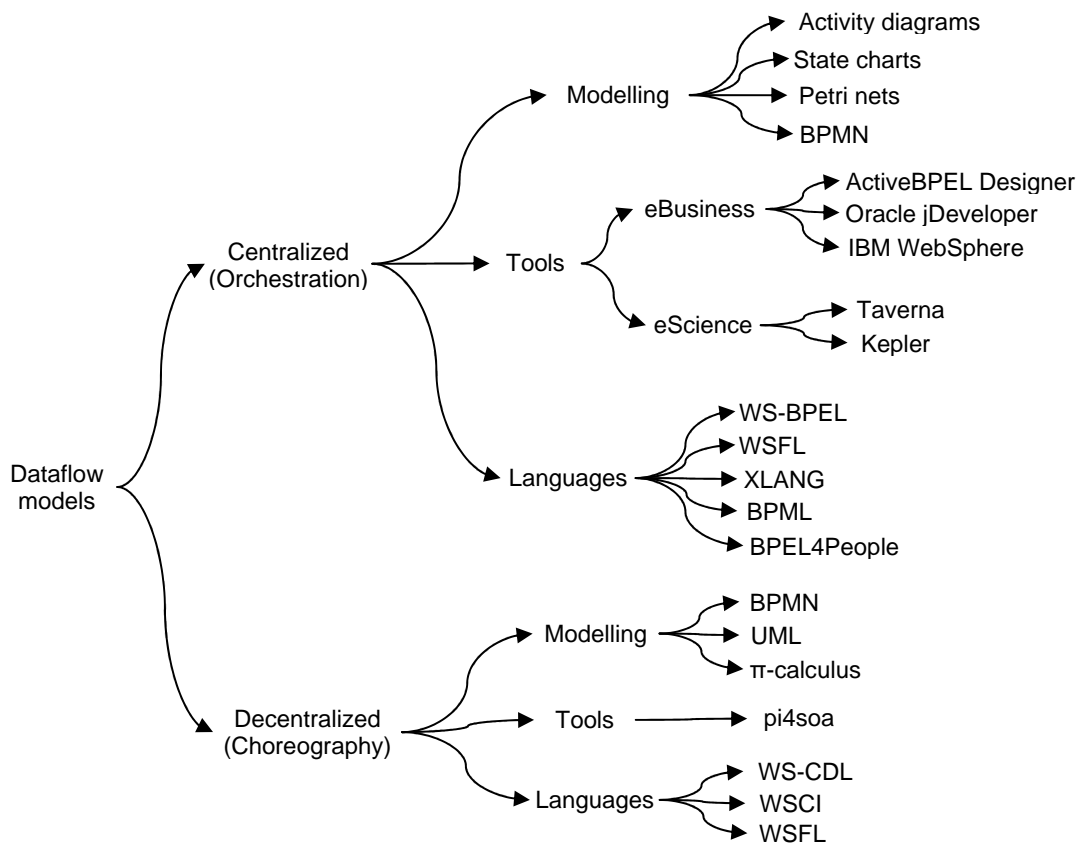


Figure 2.6. Web service composition - dataflow models.

For designing business processes that involve multiple Web services, probably executed in different containers, orchestration and choreography standards (languages and infrastructures) must achieve important technical requirements, such as asynchronous service invocation, concurrent service invocation, and management of exceptions and transaction integrity. The ability to invoke a service asynchronously is essential to accomplish the reliability and scalability required by IT environments. Concurrent service invocation can improve the process performance. Finally, the capability to manage exceptions and transaction integrity relies on how the system will respond if a service is unavailable or if there is an error, to ensure the completion of the process [33].

Beside dataflow models, different composition approaches have been proposed. These are commonly classified according to the time when

services are composed into static and dynamic, or according to user's intervention in manual and automatic mode. Static composition takes place during design time, when component services are chosen, linked, compiled and deployed [2]. In static composition, composite services are specified by models usually implemented through graphs and workflows. Dynamic composition is accomplished by defining abstract models that will be linked to services selected automatically during runtime [12]. Composite services can be developed dynamically using: model-driven [41], declarative [42], workflow-based [12] and ontology-driven [12] techniques. Model-driven composition can be specified via UML (Unified Modelling Language) and business rules written in OCL (Object Constraint Language); declarative composition, via mathematical models, PDDL (Planning Domain Definition Language) and state-charts; workflow-based composition, via abstract models; and ontology-driven composition via semantics descriptions. These specifications are analyzed and processed by different methods, matching constraints defined by the requester, and finally mapped to a composition language (e.g. WS-BPEL). SELF-SERV [42], FUSION [43], Argos [44], eFlow [45], SeGSec [46] and SHOP2 [47] are systems that implement dynamic service composition.

In manual composition, services are selected and assembled by the user. The behaviour of the composite service is usually implemented by workflows [2]. Manual composition is closely related to static composition. On the other hand, in automatic composition software agents and automated tools are used to select and assemble the composite service. Two important techniques within this approach are semantic [48] and AI-Planning [49] composition, where requests are defined by constraints and rules, and processed using ontologies.

2.3.1. Composition Languages

Web service composition is guided by languages and standards proposed by different entities to enable interoperable business processes. The language considered as a de-facto standard is WS-BPEL (Web Services Business Process Execution Language). It is an XML-based language that enables the specification of Web service interactions in business processes. It defines a model and a formal description of the behaviour and the message exchange between the process and its partners. Using WS-BPEL

it is possible to model abstract and executable processes. An abstract process is descriptive, partially specified and can be used to define a process template; while an executable process is fully specified and intended to be executed [50].

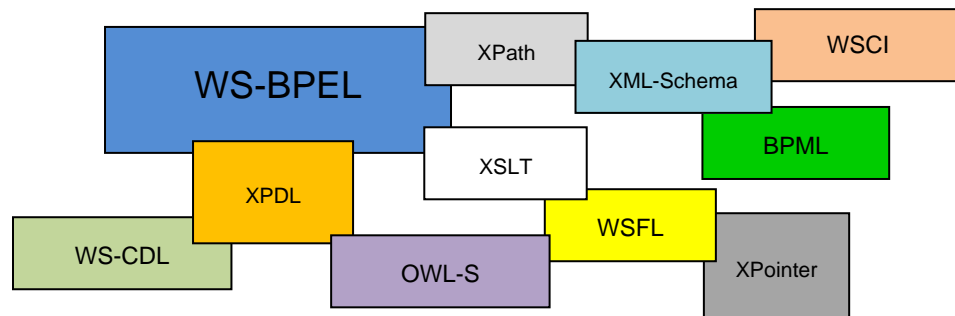


Figure 2.7. Service composition languages and standards.

Figure 2.7 illustrates some of the relevant languages and standards used within service composition. Each of these languages offers a set of different features, which are used by developers according to the specific needs of their applications.

Web Services Choreography Description Language (WS-CDL) and Web Service Choreography Interface (WSCI) are used to define decentralized service compositions. WS-CDL is an XML-based language used to describe (from a global point of view) collaborations between services, by defining their common observable behaviour. WS-CDL focuses on information exchanges and rules required during collaboration, in order to achieve a common business goal, without considering the supporting platform or programming languages used to implement the hosting environment. It is used to specify abstract business processes, in other words, is not an executable or implementation language [51]. WSCI is an XML-based language for describing dynamic interfaces of Web services participating in choreographed interaction, reusing the operations defined in a static interface (WSDL). It provides a description of the message exchange between the involved services with a global, message-oriented view of the interactions [52]. Web Services Flow Language (WSFL) is an XML-based language used to describe Web services composition. It uses flow and global models, which results into the description of business processes and partners interactions, respectively. WSFL provides support for recursive Web service composition, and enables orchestration and choreography [53].

From a semantic perspective, service composition can be specified using OWL-S. OWL for Services (OWL-S), formerly known as DAML-S, is a Web Services ontology based on OWL. In OWL-S the description of a Web service has three main classes, *ServiceProfile*, *ServiceGrounding* and *ServiceModel*, that describe what the service does, how it works, and how to access it, respectively. It provides users and software agents with a high degree of automation while discovering, invoking, monitoring and composing Web resources [54].

2.3.2. Challenges in Service Composition

Building composite services has driven the development of different proposals within academia and industry, given as a result a set of dataflow models, approaches and techniques that enable composition from various perspectives. However, some challenges are still open to solve. Some of these are closely related to automatic-dynamic service composition, and include the implementation of mechanisms that enable Quality of Service awareness, adaptive capabilities, risk awareness, conformance, security and interoperability.

- *QoS awareness*. To provide the expected results and behaviour, composite services should be aware of their QoS aspects and those of the different components involved, respecting and understanding each others policies, performance levels, security requirements and service level agreements [55].
- *Adaptive capabilities*. By implementing adaptive capabilities, Web services should be able to morph and function in spite of internal and external changes, searching to maximize the composition potential and reducing as much as possible human involvement. These adaptive capabilities include self-configuring, self-adapting, self-healing and self-optimizing. Where services are able to find new partners to interact with; function despite environmental changes; detect and react to components that do not satisfy the service requirements; and select partners that increase the benefits of the composition, respectively [55].
- *Risk awareness*. Service composition involves the use of external services, reason why users should be aware of the risks implicated,

since the QoS of the composite service can be affected as a result of problems with its components. If risk is significant, there must be a mechanism or an action to mitigate it, for example, negotiating Quality of Service with partners or invoking other services [56].

- *Conformance.* In order to ensure the integrity of a composite service, service conformance matches its operations with those of its component services. It includes mandatory semantic constraints on the components and ensures that constraints on data exchange between component services are fulfilled. It guarantees that operations do not lead to unexpected, erroneous results and preserve their meaning [55].
- *Security.* Web services enable users to interact with internal applications and databases through the Internet, which represents a security risk. Services should be concerned about security aspects including authentication, authorization, confidentiality, and integrity to protect sensitive information [34].
- *Interoperability.* During a composition process, component services should interoperate with each other to achieve a common goal. Interoperation occurs at two levels, syntactic and semantic. The former is concerned about syntactic features like the number of parameters and their data types; the latter, about semantic properties like the services domain and the functionality provided by an operation [34]. Composite services should achieve both levels of interoperability among their components to obtain the expected results.

These challenges can be addressed separately, however, some of them might complement each other (e.g. adaptive capabilities with QoS awareness, adaptive capabilities with risk awareness), bringing the possibility to combine them during the development process of composition approaches.

2.4. Quality of Service

Quality of Service (QoS) can be considered as a collection of characteristics that help evaluating and selecting resources. A detailed taxonomy of QoS parameters obtained from [57] is given in Figure 2.8. In the context of Web services, QoS properties refers to non-functional aspects (quality aspects) of Web services, such as performance, reliability, scalability, availability and security [4], which could be used as a differentiating point in the preference of consumers. By evaluating the QoS aspects of a set of Web services that share the same goals, a consumer could identify which service meets the quality requirements of the request.

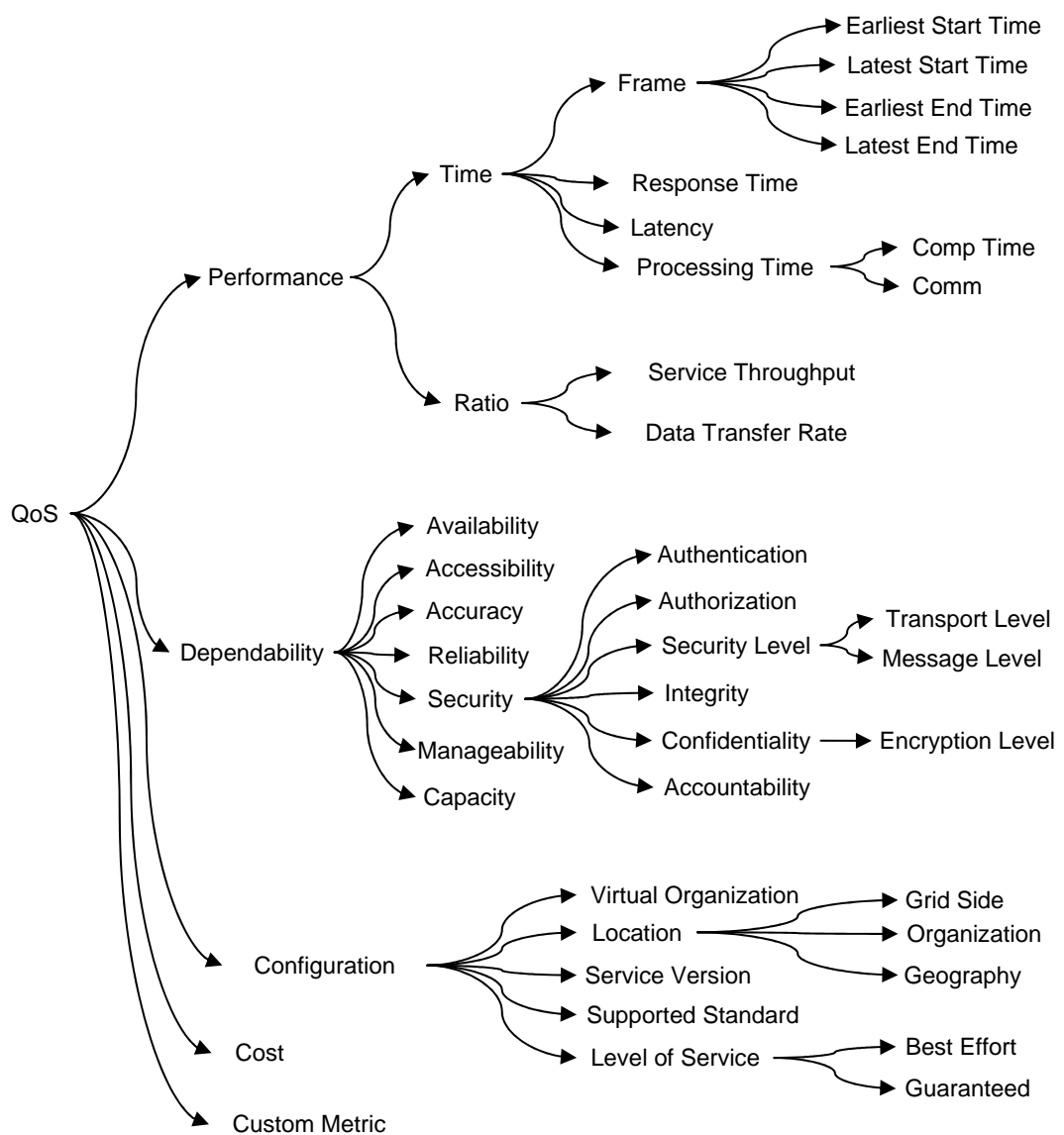


Figure 2.8. QoS parameters [57].

QoS for Web services can be classified in two subtypes: runtime quality and business quality. Runtime quality represents the measurement of properties related to a service operation (e.g. response time, reliability, availability and accessibility). On the other hand, business quality is focussed on the assessment of a service operation from a business perspective (e.g. cost, reputation and security) [34]. A list of QoS parameters for Web services and their definitions is presented as follows:

- *Performance*. Represents the speed in which a service request can be completed, measured in terms of throughput, response time, execution time, latency and transaction time [4], [58].
 - ◆ *Throughput*. Number of Web service requests served within a period of time [58].
 - ◆ *Response time*. Time consumed between invocation and completion of the requested service operation [4], [59].
 - ◆ *Processing time (execution time)*. Time taken by a Web service to process a request [4].
 - ◆ *Latency*. Time consumed between the service request arrives and the moment it is served [60].
 - ◆ *Transaction time*. Time used by the service to complete a transaction [4].
- *Reliability*. Probability that the request is correctly responded, maintaining the service quality [59]. A measure of reliability can be the number of failures per period of time (day, week, etc.) [4], [58].
- *Scalability*. Ability of increasing the computing capacity of a service provider's computer system to process more requests, operations or transactions in a given period of time [4].
- *Availability*. Probability that the system is ready to be used. The service should be available when it is invoked [4].
- *Accessibility*. Property of a service to serve a request from a consumer [58].

- *Cost.* Amount of money charged to the consumer when invoking a service operation [34].
- *Security.* Ability to ensure authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation [4].

Based on the application context and requirements, a sub-set of QoS parameters may be selected from those mentioned above, also new attributes/metrics can be defined.

2.4.1. Service Level Agreements

From a general point of view, a Service Level Agreement (SLA) can be considered as “...an explicit statement of the expectations and obligations that exist in a business relationship between two organizations: the service provider and the customer” [61]. In the context of Web services, an SLA is a document that defines the terms and conditions of quality that a service will deliver to its consumers. Its major component is the quality information, which consists of different criteria like response time and/or cost, and correspond to the service’s QoS [62]. The use of SLAs enables the negotiation process between service provider and consumer about the conditions of collaboration, and provides the consumer with confidence that the selected service will meet not only the functional but, also the quality requirements of the request.

The use of SLAs within service environments can be performed by using standards like Web Service Level Agreement (WSLA) and Web Services Agreement (WS-Agreement). WSLA is a framework for specifying and monitoring SLAs for Web services. It comprises a flexible and extensible language based on XML-Schema, and a runtime architecture that includes SLA monitoring services. The main sections comprised within the WSLA language are: parties, which identifies the contractual parties; service description, which specifies the characteristics of the service; and obligation, which defines guaranties and constraints to be imposed on SLA parameters [63]. WS-Agreement is a protocol that uses an extensible XML-based language for establishing an agreement between two entities, also enables the creation of agreement templates that help finding compatible agreement parties [64].

2.4.2. QoS in Service Composition

The relevance of QoS management in service environments has brought the need of QoS aware solutions for service composition. To experience the expected behaviour during execution of a composite service and guaranty the quality level of the composition, it may be important to consider the QoS aspects of the atomic services involved, as their drawbacks will be inherited by the composite service.

Different approaches have been presented to compute and evaluate QoS attributes of Web services within service composition and workflows' scopes. These attributes represent the non-functional characteristics required to accomplish the set of initial requirements of an application compose by different elements (tasks or services). Relevant work on this subject has been presented in [65] and [62]. The mathematical model proposed in [65] considers time, cost and reliability as the quality criteria to evaluate in workflows. This approach presents a set of metrics to obtain the quality values of individual tasks (Web services) and aggregation formulas to calculate the QoS of the entire workflow. The model used to compute these QoS metrics is based on an algorithm that reduces the workflow to an atomic task. It involves a set of inverse operations for constructing workflows including, sequential, parallel, conditional, loops, fault-tolerant and network (sub-workflows) structures. Per each structure, there are defined mathematical functions that obtain single values per quality metric. This work considers the specification of QoS attributes at design time (estimated) and a re-computation during execution. Estimation is based on collected data from previous executions (test executions), while re-computation is done using the estimate data and the workflow system log.

On the other hand, in the model presented in [62] the QoS attributes considered to evaluate single and composite services are: execution price, execution duration, reputation, successful execution rate and availability. These attributes are first obtained in the context of single services, and then computed to evaluate the QoS of the composite service. QoS attributes of single services are calculated using data from previous invocations. Composite services are considered as state charts and represented as directed acyclic graphs (DAG). If the original chart contains a cyclic structure, this is unfolded by obtaining the maximum number of possible executions, based on historical data, and determining a finite number of executions in the service structure. The aggregation formulas used to

compute QoS attributes of composite services work on execution plans obtained from their DAG representation.

QoS attributes can be considered according to the requirements of specific application domains. The models presented by Cardoso in [65], and by Zeng in [62], take in consideration quality criteria to evaluate elementary and composite services within workflow system and Web service domains, respectively. Due to their generic design, these models have been used in different works like those presented in [59], [66], [67] and [68], adjusting QoS attributes definitions and formulas based on specific needs. The aggregation formulas proposed by Cardoso are used in [59] as part of a self-healing mechanism for service composition. Cardoso's and Zeng's QoS attributes were combined in [66], where response time, cost, reliability and fidelity rating are measured for single services using a probability mass function on a finite scalar domain, and computed in workflows with specific formulas per each structure involved. The work described in [67] adopted and modified Cardoso's aggregation formulas to use them in dynamic service binding and replanning, and applied a Zeng-like method to compute loops. A Web service selection scheme that considers non-functional characteristics in two different contexts, single service discovery and optimization of service composition is presented in [68]. The QoS model includes response time, reliability, availability and price, obtained from different related work which includes Cardoso's model. Other methods and techniques proposed to evaluate QoS in service composition with the aim of fulfilling the user's quality requirements are described in [69], [70] and [71]. These works proposed the use of data mining techniques, service classification by domain and optimization algorithms, and service level agreements, respectively.

Research about QoS in service composition is not only about defining metrics to evaluate the attributes of a service, but also designing mechanisms to build services that meet both functional and quality requirements. Selecting a service that satisfies a QoS criteria for each task within a composite service can be considered a critical activity, reason why it is important to know or estimate its quality values.

2.4.3. Estimation of QoS Parameters in Service Composition

The QoS attributes of a service can be evaluated during design and execution time. At design time, these attributes help building a composite

service based on the QoS requirements of the user; while at execution time, they can be monitored to maintain the desired QoS level. Information about these attributes can be obtained from the service's profile [66], nevertheless, when this information is not available, it can be obtained by analyzing data collected from previous invocations [65].

Different approaches have been proposed to estimate the value of QoS attributes for single and composite services using historical data; some of these works are described in [59], [62], [72], [73], [74], [75] and [76]. A Semi-Markov model is presented in [59]; it is used to predict performance of single services during the execution of composite services, considering that performance may vary based on data transmission speed. The work described in [62] uses data from past observations to compute the QoS attributes of single services; for composite services, QoS values are calculated per each of the execution paths of the workflow. A comparison of different prediction methods applied to service QoS is described in [72]. Results show that last current value can provide meaningful results when predicting variability. The use of layering query networks with UML models to predict the performance of composite services is proposed in [73]. The approach presented in [74] uses decision trees for performance prediction with the aim reducing the number of service's reselections in service composition. An algorithm based on graph reduction is presented in [75]; its objective is to predict response time of composite services. Different forecast techniques are combined in [76] to provide an adaptive QoS prediction approach, which aims to improve the overall accuracy of the predictions by combining the advantages of the individual techniques.

By accurately estimating the QoS values of a Web service, considered part of a composite service, and also estimating the QoS values of the composition itself, it may be possible to minimize performance problems during its execution and maintain its quality levels.

2.4.4. Predictive Algorithms

Computer systems that keep information from previous executions can use it in order to learn and predict future events. A prediction can be considered as "*... an estimation of the value of a variable V_i occurring at time T_i on the future, conditioned on historical information*" [77]. Predictive algorithms are tools that can be used to analyze data collected from a sequence of

observations of an event in long-term and short-term predictions. They have been extensively used in different areas, like performance prediction, systems and networks management.

In the context of Web services, predictive algorithms have been used to estimate QoS values. Running average was applied in [62] as part of a composition approach, while single last observation and running average were used in [72] as part of a comparison study. Single last observation, running average and low pass filter are examples of predictive algorithms [78] that are simple and require little processing time, which makes them a good alternative in software solutions where time is a key constraint.

- *Single last observation.* The prediction is the most recent observation. It considers that the last value will reflect the behaviour of the next run.

$$P = V \quad (2.1)$$

- *Running average.* The prediction is the mean average of all the previous observations. It can be limited, defining a window of “ n ” most recent observations.

$$P = \frac{\sum_1^n V}{n} \quad (2.2)$$

- *Low pass filter.* The prediction is the average recent behaviour of an indexer. It uses a degrading function that affects the values of older observations.

$$P = w \cdot P_{n-1} + ((1 - w) \cdot V) \quad (2.3)$$

Where:

- P is the prediction of the new value,
- P_{n-1} is the previous pass filter value,
- V is the last observation value,
- w is a weight value between 0 and 1.

Other prediction methods include Auto Regressive Integrated Moving Average (ARIMA) [79], linear regression [72] and exponential smoothing [80]. They can be also used to predict QoS values of Web services, based on the available information and type of prediction required by the application. However, it is important to consider that because of their complexity, the use of these methods involves high processing time.

2.5. Adaptive Service Composition

Different factors, like the involvement of third-party resources (components/infrastructure) and the use of wide area networks, can influence the behaviour of distributed systems. In the field of Web services and service composition, adaptive techniques have been proposed to deal with the consequences of external and internal factors, and ensure that services maintain their functional and quality levels, by adapting in automatic to unexpected events and environmental conditions. These techniques are closely related to autonomic computing and self-* properties.

2.5.1. Autonomic Computing and Self-Adaptive Software

The growing complexity of Web service platforms, increase emphasis on QoS, and variable workloads, make the management of Web services' performance a time-consuming and complicated task. Autonomic computing has appeared as a solution to deal with this complexity and ensure SLA compliance. It aims to transfer software management responsibilities from administrators to the software it self. Systems with self-managing capabilities, also known as autonomic systems, make possible to deal with their complexity by managing themselves according to objectives specified by humans [81].

Because of the broad context of autonomic computing (coverage at hardware, operative system, network, middleware and application levels), more limited self-managing models fall under its umbrella, that is the case of self-adaptive software. Self-adaptive software evaluates and changes its own behaviour when it is not achieving its goals, usually focusing on the application and middleware layers [82]. To accomplish these tasks, self-adaptive systems should embrace certain capabilities, also known as self-* properties, which include: self-healing, self-configuring, self-optimizing and self-protecting.

- *Self-healing.* Focus on discover, diagnose, and react to disruptions. It can also predict potential problems and take suitable actions to prevent the system from failing. Some of the sub-properties related to self-healing are self-diagnosing and self-repairing [82]. "*Self-healing components can detect system malfunctions and initiate policy-based corrective actions without disrupting the IT environment. Corrective*

actions could involve a product altering its own state or effecting changes in other components in the environment” [83]. Self-healing systems should consider a fault model to define the faults they are going to be able to heal. Some of the characteristics to include in this model are the duration, manifestation, source and granularity [84]. The analysis and classification of these faults allow the definition and implementation of the recovery strategies. Self-healing is related to availability, survivability, maintainability and reliability [82].

- *Self-configuring*. Reconfigures automatically and dynamically by installing, updating, integrating, and composing/decomposing software entities [82]. “*Self-configuring components adapt dynamically to changes in the environment. Such changes could include the deployment of new components or the removal of existing ones, or dramatic changes in the system characteristics*” [83]. Dynamic adaptation helps software systems to ensure their functionality, provide flexibility and reduce human involvement to minimum. Self-configuring is related to maintainability, functionality, portability, and usability [82].
- *Self-optimizing*. Management of performance and resource allocation in order to fulfil user’s requirements [82]. “*Self-optimizing components can tune themselves to meet end-user or business needs. The tuning actions could mean reallocating resources to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion*” [83]. By using self-optimizing methods, users get high service levels, as systems continuously try to improve their own behaviour. Self-optimizing is related with efficiency and functionality [82].
- *Self-protecting*. Detects, identifies and protects the system from security breaches, and recovers from their effects [82]. “*Self-protecting components can detect hostile behaviours as they occur and take corrective actions to make themselves less vulnerable. Hostile behaviours can include unauthorized access and use, virus infection and proliferation, and denial-of-service attacks*” [83]. The use of self-protecting methods enable systems to defend against large-scale, correlated problems arising from malicious attacks or

cascading failures that remain uncorrected by self-healing measures [85]. Self-protecting is related to reliability and functionality [82].

2.5.2. Adaptation in Service Composition

Adaptation in service composition aims to mitigate the impact of unexpected events that take place during the execution of composite services, maintaining functional and Quality of Service levels. Some of the main aspects that can be considered as part of adaptation solutions in service composition include, but are not limited to:

- *Adaptation goal.* Purpose of adaptation. Adaptation goals can be defined based on functional and/or non-functional (Quality of Service) needs. Some approaches deal with single Quality of Service requirements, while others focus on maintaining multiple requirements [86].
- *Adaptation level.* Defines those elements that will change in order to achieve the adaptation goal. From a Web service level perspective, adaptation is tackled per each activity that involves service binding. In workflow level approaches, the logic of the composition can be modified, by adding or removing elements, or reorganizing certain sections of the process [86]. At a higher level, adaptation can also involve the allocation of physical resources (e.g. CPU, memory, and bandwidth) to specific services in order to improve/maintain their QoS (this is limited to in-house services) [87].
- *Adaptation actions.* Actions used to solve the adaptation problem. The actions taken are based on the adaptation levels discussed above [86]. For adaptation performed at Web service and workflow levels, actions applied can involve:
 - ◆ *Service re-call (retry).* Invokes the same faulty service [88].
 - ◆ *Service tuning.* Changes the behaviour/properties of the invoked service operation [86].
 - ◆ *Service selection (service replacement).* Selects from a set of candidates, a new service with equivalent functional/non-

functional characteristics and dynamically binds this service to the failed task [86], [88].

- ◆ *Redundancy*. Executes equivalent operations from multiple services by using coordination patterns or replication. With a coordination pattern, each activity is bound to a set of equivalent operations applying a redundancy pattern [86]. On the other hand, replication techniques use similar services as redundant replicas [89].
- ◆ *Workflow redesign*. When there is no alternative service to rebind to, the activity is replaced by an equivalent sequence/parallel branch that executes two or more services [88]. Other redesign approach involves adding/removing certain functions from the workflow.
- *Adaptive mechanisms*. Approaches applied to implement the adaptation actions.
 - ◆ *Agent-based*. Involves the use of agents to manage the adaptive properties of the service composition [90].
 - ◆ *Policy-based*. Uses the definition of policies that manage different stages of adaptive service composition, e.g. service discovery/selection, monitoring and/or recovery actions [91].
 - ◆ *Rule-based*. Applies rules to describe constraints that lead the adaptation process.
 - ◆ *Feedback-based*. Collects feedback reports about service executions, and uses this information to decide whether to use or not certain services [92].
- *Stage of adaptation*. Time when adaptation is performed. During service composition there can be identified different adaptation stages: development time, compile/link time, load time and runtime. At runtime, adaptation can be triggered from two perspectives, proactive and reactive. The former is activated in advance, using predictions of future states, while the latter is activated after a change has been detected [86].

- *Awareness levels*. Describes the scope of information that will be available in order to perform adaptation.
 - ♦ *Event awareness*. Based on simple events, which trigger basic event-condition-action rules [93].
 - ♦ *Situation awareness*. Considers relevant events, understanding their implication in a wider context [93].
 - ♦ *Adaptability awareness*. Focuses on the adaptation capabilities of an entity in its environment; it enables cooperative adaptation [93].
 - ♦ *Goal awareness*. Involves understanding the goals of the different entities implicated, as well as the goal of the entire composition. Goals can be functional and non-functional properties [93].
 - ♦ *Future awareness*. Looks at the life-cycle of the system. It requires information on probable future states based on future events [93].
 - ♦ *Context awareness*. The system is aware of its context, which is its working environment [82].

Software systems must become more flexible, customizable, configurable, recoverable and dependable, by adapting to environmental, context and systems changes [5]. Distributed software as Web services, must be capable to adapt in response to their perception of the environment and their own behaviour, without compromising their efficiency. Composite services should be able to adapt, also based on their components performance, in order to provide the user with the expected behaviour and result on the request.

The adaptation life cycle for service composition used in different approaches, like those presented in [86] and [87], derives from the generic MAPE-K loop in autonomic computing [83]. It is a closed loop that comprises four main functions: monitoring, analyzing, planning and executing, as depicted in Figure 2.9.

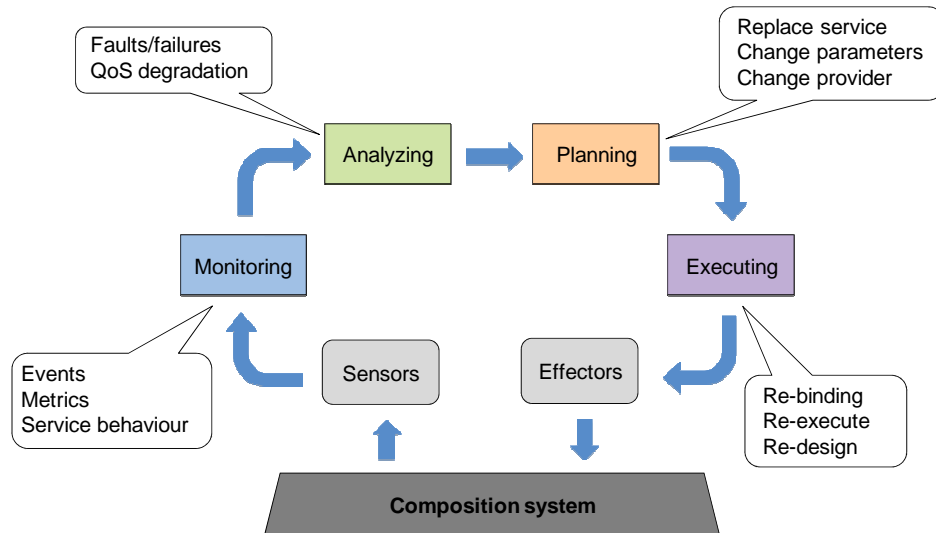


Figure 2.9. Self-Adaptive cycle for service composition.

These functions (steps) are closely related to the adaptation aspects previously described. The monitoring function collects data from sensors and obtains behavioural patterns and symptoms (relates to adaptation level, goal and awareness level), which are then computed by the analyzing function in order to detect when adaptation is needed (relates to adaptation level and stage). Next, within the planning function, it is determined what needs to be changed and how (relates to adaptation level and actions). Finally, the executing function provides mechanisms to perform the changes and applies those actions determined by the planning function (relates to adaptation actions and mechanisms).

2.5.2.1. Self-Adaptive Properties

Besides the adaptation aspects described earlier in this section, other important characteristics to look at within adaptive service composition, are the self-* (self-adaptive) properties and their benefits. The use of self-* properties enable composite services to deal with the dynamicity of the Web service execution environment. These properties allow services to function despite environmental changes, detect and react to components that do not satisfy the service requirements, and select partners that increase the benefits of the composition.

Based on the objectives of the composition, and the causes and impact of adaptation, different properties can be selected and implemented. Some self-* properties applied in service composition approaches are: self-

healing, self-optimizing and self-configuring (discussed in section 2.5.1). Self-healing services can monitor themselves, predict/detect the causes of failure and make the adjustments to restore their states to normal. Failures can be either functional or non-functional [94]. Self-optimizing systems aim to select the best available services for executing the composition, and define the most appropriate Quality of Service levels in order to maximize benefits and reduce cost [95]. Self-configuring services can leverage services and resources to compose an optimal configuration based on user requirements and the characteristics of the system [96].

Composite services can also be self-aware. Self-awareness is defined by the combination of three properties: self-reflective, self-predictive and self-adaptive, which enable services to be aware of the system architecture and execution platform; predict the effect of changes on their behaviour and effects of adaptation actions; and proactively adapt to ensure that their QoS requirements are satisfied [97].

Table 2.1 presents a list of self-* properties and their relationships to the events that can occur when executing composite services, the actions (response) the system should take in order to adapt and the goals of adaptation.

Table 2.1. Relationship between self-* properties and events/action/goals in service composition.

Self-* property	Event	Action	Goal
Self-healing	Server not available	Select a new service	Prevent composition from failing (time out)
	Service not available	Select a new service	Prevent composition from failing
	Wrong service found	Select a new service	Prevent composition from failing (wrong functionality)
	Wrong parameter type	Perform a cast to send the right parameter type	Prevent composition from failing (component crash)
		Select a new service that matches the parameters type	Prevent composition from failing (component crash)

	Service crashed	Select a new service	Prevent composition from failing
	Service QoS constraint violation	Recall or select a new service	Prevent composition from failing (global QoS violation)
Self-configuration	No service provides the required functionality	Workflow redesign (add new branch that provides the functionality)	Enable composition to provide the required functionality
	Parameters mismatch	Adjust input parameters	Avoid obtaining incorrect results
	Workflow inconsistency	Workflow redesign	Prevent composition from failing
Self-optimization	Global QoS degrading	Select a new service	Maintain the global QoS
	Value of a QoS property is degrading	Select a new service	Maintain the global QoS
	Value of a QoS property is upgrading	Select a new service	Upgrade the global QoS
Self-awareness*	Service failure (E)	Determine the type of failure and trigger its corresponding adaptation mechanism	Prevent composition from failing
	Service QoS upgrading (E)	Trigger the corresponding adaptation mechanism	Upgrade the global QoS
	Service QoS degrading (E)	Trigger the corresponding adaptation mechanism	Maintain the global QoS
	Global QoS upgrading (E)	Trigger the corresponding adaptation mechanism	Upgrade the global QoS
	Global QoS degrading (E)	Trigger the corresponding adaptation mechanism	Maintain the global QoS

	Adaptation could be triggered (A)	Analyze whether adaptation is necessary or not	Maintain the composition functionality and QoS
	Different self-* properties can be triggered (A)	Find the most suitable self-* property and trigger the adaptation mechanism	Maintain the composition functionality and QoS

*(E) Event-awareness

*(A) Adaptability awareness

2.5.2.2. The Need for Adaptation in Service Composition

As stated in section 2.3.2, one of the challenges in service composition is the implementation of adaptive capabilities that enable services to work despite of unexpected situations that may affect their behaviour. These capabilities do not only focus on preventing composite services from failing, but also maintaining their Quality of Service levels (discussed in section 2.5.2.1), in order to ensure that the service consumer obtains the expected results.

Factors like: the nature of service composition, dynamicity offered by the environments where services are executed, and increasing number of services (that may provide the same functionality), have turned the management of composite services into a highly complex task. Besides, when customers invoke a composite service, they may have different QoS constraints, but expect the same functional requirement to be fulfilled. Therefore the need for adaptation, as the use of self-adaptive capabilities enable composite services to modify their behaviour according to changes in the environment, their internal components' behaviour and pre-established constraints, increasing the flexibility of the service itself and reducing as much as possible human involvement.

2.5.3. Reactive vs Proactive Adaptation

Based on the moment when adjustments take place, adaptation approaches are classified as either reactive or proactive. In service-based applications, reactive adaptation is triggered after problems have occurred, when situations like the use of faulty services or services that present undesirable

QoS have already affected the application [98]. The use of reactive mechanisms may cause increases in the execution time and financial loss, which can lead to user and business dissatisfaction [99]. Proactive approaches aim to deal with some of these drawbacks by detecting the need for a change in advance.

Situations that can be predicted in proactive adaptation approaches for service composition include: the impact of a new requirement, misbehaviour of a service and the existence of new services [99]. Techniques like data mining, online testing, statistical analysis, runtime verification and simulation, are applied during the prediction stage of the process, with the aim of accurately predict the behaviour of services and service oriented systems [100].

2.6. Other Adaptive Approaches for Service Composition

Adaptation in service composition is not limited to the use of self-* properties. The abilities to bind services dynamically at run time and offer a set of fault tolerant techniques to support service composition can also be considered as part of adaptive mechanisms.

2.6.1. Late Binding

Late binding, also known as dynamic binding, is a concept related to the time when entities are bound to their implementations (e.g. procedures, libraries) [101]. In service oriented systems, late binding mechanisms provide the ability to bind services dynamically at runtime, after selecting them based on specific required functionality and/or quality criteria (e.g. response time, cost, etc). When using simple clients, services can be invoked using:

- *Dynamic proxy*. Invokes a Web service without stubs code generation. It is obtained at runtime and requires a service endpoint interface to be instantiated [102].
- *Dynamic invocation interface*. Invokes a Web service at run time without using a WSDL file. The client needs to provide operation name, parameter names, types, modes, and port type [102].

- *Broker*. Manages the binding establishing a bridge between consumer and provider. It selects a set of candidate services, ranks them and then selects the top service to bind to [103].

When late binding is going to be performed from a service composition perspective, some of the approaches that can be used include:

- *BPEL features*. Include limited dynamic binding characteristics that enable reassigning end points at runtime [104].
- *Agents*. Executed on top of a composition engine. They perform activities like discovering, matching and binding services [90].
- *Proxy service*. Works with abstract processes (e.g. abstract BPEL), binding abstract tasks to proxy services that will point to the actual component services at runtime [105].
- *Semantic-based middleware*. Uses semantic technology to find the most suitable services from a set of candidates, and then performs dynamic invocation [106]. It is used as a bridge between abstract processes and services.

Different approaches have been proposed to enable late binding in service composition, like those described in [101], [105], [106], [107] and [108], where functional and non-functional characteristics on candidate services are considered with the aim of optimizing the overall QoS of the applications. The work presented in [101] describes a late-binding mechanism for adaptive business processes. It introduces a pre-processing stage to avoid delays at call time. The user's QoS preferences are modelled using a linguistic conditional preference networks (LCP-nets) model and specified using WS-agreements. The implementation language for business processes is an extended version of WS-BPEL, which includes three new activities: *lateBindingConfigure*, *monitoring*, and *lateBindingInvoke*. A framework for enabling late binding in service compositions is presented in [105]. It supports pre-execution binding, run-time binding and run-time re-binding. Pre-execution binding is performed before the actual execution. Run-time binding permits the selection of a service bind at run time, just before its abstract service is invoked. Finally, re-binding is a strategy to support recovery actions in runtime, when the QoS values deviate from estimates or a constraints violation occurs. The use of a semantic approach

is presented in [106]. In this project, BPEL processes are bind to a semantic-based middleware, instead of performing static binding to a specific service. The middleware uses semantic technology to find the most suitable services from a set of candidates, and perform a dynamic invocation. In the work described in [107], if there is any deviation from the estimated QoS of the composite service or a service becomes unavailable, a re-binding mechanism is triggered. The framework presented in [108] uses policy-based mechanisms for service composition. It combines late binding with runtime service discovery. This approach proposes two binding types, QoS-based and content-based dynamic binding.

A common characteristic among these approaches is the use of abstraction into each task or function of the composite service. These abstract elements are bind dynamically to concrete services to provide an agile execution of the composition.

2.6.2. Fault Tolerance

A fault is an abnormal condition in a component, which can lead to failure. In Service Oriented Architectures, like in other distributed systems, failures can occur at hardware, software, network (communication), and operator level [109], [110]. However, there are specific faults that can take place during different steps of the SOA process.

The following table contains a fault taxonomy for SOA developed combining the approaches proposed in [110] and [111].

Table 2.2. SOA specific faults.

Stage	General fault	Specific fault	Causes
Publish	Service description fault	<ul style="list-style-type: none"> Faulty description Service/description mismatch 	<ul style="list-style-type: none"> Development fault
	Service deployment fault	<ul style="list-style-type: none"> Required resource missing Service/server incompatible 	<ul style="list-style-type: none"> Development fault
Discovery	No service found	<ul style="list-style-type: none"> Required service not existing Not listed in lookup service 	---

	Wrong service found	<ul style="list-style-type: none"> • Incorrect search criteria • Faulty lookup service 	<ul style="list-style-type: none"> • Service description fault
	Timed out	---	<ul style="list-style-type: none"> • Service down • Server crashed • Communication faults
Composition	No valid composition	<ul style="list-style-type: none"> • Incompatible components • Parts of composition missing 	<ul style="list-style-type: none"> • Development fault
	Faulty composition	<ul style="list-style-type: none"> • Criteria not met • Contract not met • Misunderstood behaviour • Workflow inconsistency • Composition engine fault 	<ul style="list-style-type: none"> • Service description fault • Wrong service found • Development fault
	Timed out	<ul style="list-style-type: none"> • Unavailable service 	<ul style="list-style-type: none"> • Service down • Server crashed • Communication faults
Binding	Binding denied	<ul style="list-style-type: none"> • Authorization denied • Authentication failed • Accounting problems • Insufficient security 	<ul style="list-style-type: none"> • Unprivileged users
	Bound to wrong service	---	<ul style="list-style-type: none"> • Service description fault
	Timed out	<ul style="list-style-type: none"> • Unavailable service 	<ul style="list-style-type: none"> • Service down • Server crashed • Communication faults
Execution	Service crashed	---	<ul style="list-style-type: none"> • Development fault
	Incorrect result	<ul style="list-style-type: none"> • Incorrect input • Faulty service 	<ul style="list-style-type: none"> • Service description fault • Development fault • Faulty composition
	Timed out	<ul style="list-style-type: none"> • Unavailable service 	<ul style="list-style-type: none"> • Service down • Server crashed • Communication faults

Some of the causes of these faults are also faults, and some of them introduced in previous SOA stages, e.g. a wrong service description may

cause a wrong search result. Development faults, which can be introduced by human developers, are one of the common fault causes. From the provider side, these can include changes in the service interface and changes in the logic of the service; from the client side (e.g. composite service), wrong bindings and parameters incompatibility.

The ability of a system to deliver its expected service, despite the presence of fault-caused error, it is called fault tolerance [112]. A fault tolerant service is capable to detect errors and recover from them without external interventions, by using fault tolerance mechanisms. By distinguish and classify the different faults that can affect a specific system, it is possible to develop the proper fault tolerance mechanism. In the Web service context, fault tolerance mechanisms can be applied at atomic service and composite service levels. For atomic services, some of the mechanisms used are replication, check point, retry and the use of alternate resources. On the other hand, for workflows or composite services, fault tolerant mechanisms include the use of exception handlers (defined by the user), alternate task, redundancy and rescue workflow [113].

A fault tolerant framework for Web services is presented in [114]. It uses active, warm passive and cold passive replication techniques to create service groups, where only one service is designated as primary member. When the primary member fails, it is removed from the service group, a backup member is set as the new primary and a new backup member is deployed. The work in described in [115] aims to provide Web services with higher resilience to failure. Fault tolerance is implemented by using a passive replication scheme, where a service group is created and each service has a warm replica to call in case of failure. In the composition context, a fault tolerant model for service orchestration, which uses passive and active replication techniques, is presented in [116]. The model supports fault of crash by replicating services; per each service replica, there is a standby replica. When a replica call fails, the BPEL fault handler redirects the call to a standby replica. Results are given to the client when at least one replica had no faults. The approach presented in [117] proposes a mechanism to develop fault tolerant composite services using alternative resources, allowing developers to include different services per each task. This mechanism also evaluates the behaviour of the components at run time, considering response time, availability and correctness.

These works rely on the implementation of redundant replicas or the use of multiple services to satisfy a single task, which creates a dependency on other servers and generates high costs in processing power. The use of fault-handlers within BPEL code can turn the development of models into a highly complex activity when many tasks are involved.

2.7. Decision Support Systems

Adaptive mechanisms require tools to rely on during the decision making process. These mechanisms are known as decision support systems. Decision Support Systems (DSS) are interactive components that help during judgment and choice tasks. In order to support framing, modelling, and problem solving, DSS enhance the use of information with models and model-based reasoning. Decision making models consider three main components: a measure of preferences over decision objectives, available decision options, and a measure of uncertainty over variables influencing the decision and the outcomes [118]. A list of DSS and their definitions are presented as follows:

- *Bayesian networks*. Probabilistic graphical models used to represent knowledge about uncertain domains. Graphs have two main elements, nodes and edges. Each node corresponds to the representation of a random variable, while the edges between nodes represent probabilistic dependencies among the corresponding random variables. Dependencies can be estimated using statistical and computational methods [119].
- *Decision trees*. Method for approximating discrete-valued target functions, where functions are represented by diagrams of decision. Decision trees are tree-like diagrams, which have leaf nodes and branches. Leaf nodes represent attributes, while branches correspond to possible values for the attribute. Each path from the root to a leaf matches a conjunction of attribute tests, and the tree itself, to a disjunction of these conjunctions [120].
- *Decision tree ensembles*. Ensemble methods are generic techniques used to improve a learning algorithm by using several models and then aggregating their predictions. Some of these methods proposed

for decision trees include: bagging, random forest, extra-trees and boosting [121].

- *Neural networks*. Technique used for learning real-valued, discrete-valued, and vector-valued target functions from examples [120]. They can be used to predict the behaviour of a system based on different inputs, and build a model by using training samples. The performance of the network is based on its structure and the quality of the training data. Neural networks are useful when building models for control purposes [122].
- *Genetic algorithms*. Stochastic-based techniques based on simulated evolution [120]. They comprise a population of individuals, where each individual encodes a candidate solution in a chromosome. During each step of evolution, hypothesis of every individual are recalculated, and parts of the best hypothesis are combined and/or mutated to form the next generation [123].
- *Reinforcement learning*. Paradigm focused on learning how to control a system and maximize its long-term objective. Its goal is to develop learning algorithms, along with the understanding of their metrics and limitations. Reinforcement learning algorithms use powerful function approximation methods to compactly represent value functions [124].
- *Fuzzy logic*. Method based on multi-valued logic which aims to formalize approximate reasoning. It is used to deal with different types of uncertainty in knowledge-based systems. Some of the main components of a fuzzy system are: fuzzy sets, linguistic variables and fuzzy rules; where a set is a collections of objects characterized by a function, linguistic variables represent their values with words, and fuzzy rules correspond to human knowledge [125].
- *Case based reasoning*. Method for problem solving and learning based on previous experiences. Old situations and their solutions are encapsulated into a case structure and stored in a case-base, which is queried when a new problem is encountered. The most similar cases are retrieved, solutions of these cases are modified to conform to the new situation and then, stored in the case-base [126].

Based on the information available, along with the nature and specific needs of the application, different decision support systems can be selected. It is important to consider that due to their complexity levels, some of these mechanisms may exhibit significant overheads at runtime.

2.8. Summary

This chapter has provided a definition on Service Oriented Architectures to help introducing Web services and service composition. The concept of Web services was explored in detail, presenting an overview on service related standards, the service life cycle, and listing some of the benefits of using services when developing software solutions.

An outline of service composition is then presented, along with the descriptions of relevant composition languages and main challenges in the field. Quality of Service in the context of service oriented environments is then defined, alongside a discussion of related work in QoS management and QoS estimation in service composition.

Adaptive service composition is then described from the perspective of autonomic computing. Different self-* properties are defined and related to the events that can occur during the execution of composite services, followed by a discussion about the need for adaptation in service composition. Other adaptive approaches used in service environments are then discussed. Finally, different decision support systems that can be used within adaptive mechanisms are defined.

This chapter provided a background on relevant topics and related approaches to help the understanding of the work described in this Thesis, which performs service composition using a centralized model (defined in section 2.3), considering different non-functional attributes (defined in section 2.4) as adaptation goal. Adaptation is carried out at Web service level, using a rule-based approach for service selection at runtime, taking into account self-optimization and self-healing capabilities (defined in section 2.5). Fuzzy logic is used as a decision support system (defined in section 2.7) to help during the decision making process of mechanisms two and three, described in chapters 4 and 5, respectively.

The following chapter will describe a self-optimization model for service composition. It will present a discussion on related approaches and the solution proposed to overcome their limitations.

Chapter 3

A QoS Optimization Model for Service Composition

This chapter introduces a QoS optimization model for service composition. The motivation behind the development of the optimization model is described through a service composition scenario. A discussion on work related to the provision of self-adaptation in service composition is then provided. The proposed solution is described, followed by its implementation details. Finally, the experiments performed to evaluate the model, alongside their configuration and results, are discussed in detail.

3.1. Motivation

Service Oriented Architectures have encouraged the development of applications based on reusable and distributed components, and the design of flexible business processes. These business processes, also known as composite services, enable the structured interaction of services developed and hosted by different entities. The scenario described in this section provides a representative example of a composite service and its interaction with other services and a service consumer. The actors involved in this scenario are a customer, a travel agent Web site (service consumer), a travel agent service, service providers and a credit card company [127].

- *Customer.* Aims to obtain a vacation package with the best services and prices available.
- *Travel agent Web site.* Offers the ability to book vacation packages (hotel, airplane tickets, ground transportation, etc.) and tries to provide customer satisfaction.
- *Travel agent service.* Interacts and coordinates the execution of Web services.
- *Service providers.* Aim to sell products and expose Web services to query information and perform reservations on them (hotels, airlines, etc.).

- *Credit card company.* Provides services to guaranty and process payments.

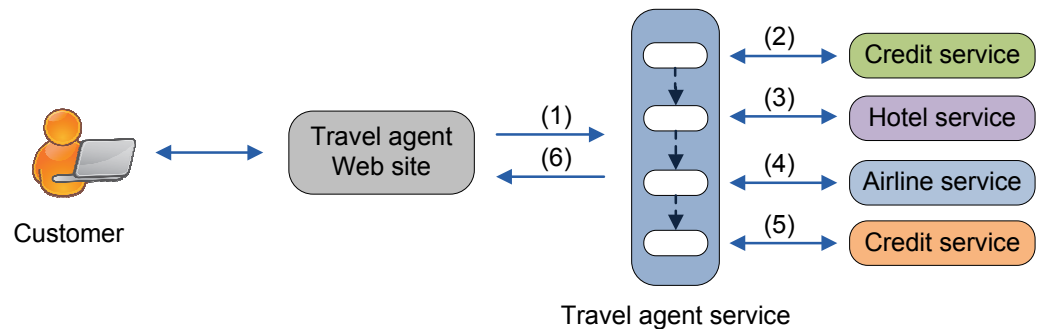


Figure 3.1. Composite service example.

The process starts when the customer fills and submits a form with the holidays package requirements, through the agent's Web site. The travel agent site finds a list of hotels and airlines, and presents the list of results to the customer, letting him choose the best options according to his needs. To book the customer's choice, (1) the travel agent Web site invokes a composite Web service (travel agent service) to coordinate the interaction of credit, airline and hotel services (see Figure 3.1). (2) The travel agent service contacts the credit service to confirm payment, if the response indicates success with an authorization identifier (signed by the payment authority), proceeds to book the hotel room. (3) The travel agent service requests a description of how to book a room to the hotel service, sends the request accordingly and a payment authorization identifier from the credit service. (4) To confirm the flight reservation, the service requests a description of how to buy a ticket to the airline service, sends the request accordingly and a payment authorization identifier from the credit service. (5) The travel agent service charges a fee to the customer, using the authorization identifier signed by the credit service. (6) Finally, the service sends to the customer, through the Web site, the confirmation identifiers of the vacation package.

This is an ideal scenario where all the activities (in the composite service) that involve invoking service operations are executed without problems (delays, faults, etc.). At the end of the process, the service consumer (travel agent Web site) gets the expected results and fulfils the customer's requirements. However, in the real world the behaviour offered by services exhibits frequent variations (see Figure 3.2), therefore, obtaining the expected results while running a service is not guaranteed. This has

caused the need of mechanisms and tools focused on helping providers to ensure the provision of services with certain quality levels.

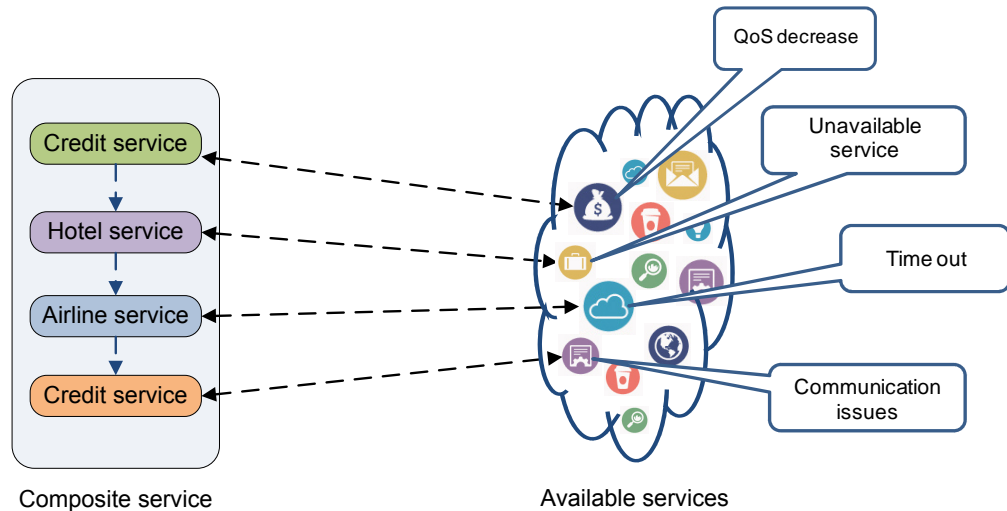


Figure 3.2. Events that can occur at runtime.

As described in section 2.5, adaptive mechanisms provide software systems with capabilities to self-heal, self-configure, self-optimize, self-protect, etc., in order to deal and mitigate the impact of unexpected events that can occur during service executions. The scope of the work described in this chapter is primarily concerned on the development of a model that helps maintaining and, if possible, improving the QoS levels of composite services.

3.2. Self-Adaptation in Service Composition

Research on self-adaptation in service composition is primarily associated with the design and implementation of self-healing and self-optimizing capabilities. Self-healing methods have been extensively studied in the last years. Work in this area can be found in approaches like those presented in [59], [90], [107], [108], [128] and [129], where new services are selected and invoked after a functional failure or a QoS constraint violation. These works are mainly focused on targeting events like:

- *QoS degradation.* The quality values of the composition have decayed and are far from expected.
- *Unavailable service.* The service is down or has no network connection.

- *Service time out.* The server where the service was running crashed or there is a network fault.
- *Communication issues.* The network is not working correctly.

On the other hand, mechanisms that implement self-optimization are closely related to the selection of services at runtime, in order to maintain the expected QoS of the entire composition. Examples of works that use these mechanisms are described in [86],[87],[95],[107] and [130], and summarized as follows:

- The methodology and framework proposed in [86] are focused on QoS driven adaptation for service composition. Adaptation is performed using service selection and coordination patterns. When using redundancy schemes, QoS levels of a single service operation are improved by increasing its cost. The framework uses an optimization engine to determine the adaptation policy and ensure the composition meets the QoS goals.
- The framework presented in [87] facilitates the development of adaptive service-based systems by implementing service selection, runtime reconfiguration and resource assignment. Based on the behaviour of previous executions and adaptation requirements, concrete workflows are re-deployed, replacing older versions. When adaptation is targeted by resource allocation, applies only for in-house services and takes place at runtime.
- The framework described in [95] enables designers to develop BPEL workflows, in which they can define at design time the information required to adapt at runtime, including a set of candidate services and constraints. The framework aims to select the best services to invoke from the process along with the most appropriate QoS levels. A disadvantage of this work, is the level of human involvement at design time.
- The solution presented in [107] proposes a QoS aware binding mechanism based on genetic algorithms. It searches for the best possible set of services to invoke. At runtime, the bindings can be reconsidered and sections of the composition can change. This action is triggered when estimations of the workflow's QoS indicate a

possible deviation of the initial QoS and risk of SLA violation. Then, the composition stops, the remaining part of the workflow is re-planned and re-bound, and finally, the workflow execution finishes.

- The framework presented in [130] applies mixed programming to relate abstract services with executable services. It proposes the use of an adaptive QoS negotiation mechanism between users and the service broker. This enables users to decrease their requirements at runtime and reduce the number of QoS violations.

These approaches are mainly focused on the selection of services that offer high quality values and the use of utility functions while selecting the set of services to bind to. However, they only consider situations where quality levels are degraded. Besides, some of the adaptation strategies apply in the next execution of the composition, or require human specifications. Self-optimization can be also targeted when one of the QoS values of the entire composition has being enhanced at certain point of the execution.

The work presented in this chapter includes this information as part of its adaptation mechanism, considering that this behaviour provides some slack that can be used while selecting the next service in the process, enabling the improvement of other QoS attributes. Also, adaptation is considered to take place at runtime, without stopping the execution of the composite service. Further discussion between related approaches and the model presented in this chapter will be provided in chapter 6.

3.3. Proposed Solution

The use of a QoS aware and adaptive environment on the service provider side can help fulfilling customer requirements from both perspectives, functional and qualitative. From a higher level, this environment works as a middle point between the final consumer (user/application) and those services involved in a composition process, as depicted in Figure 3.3.

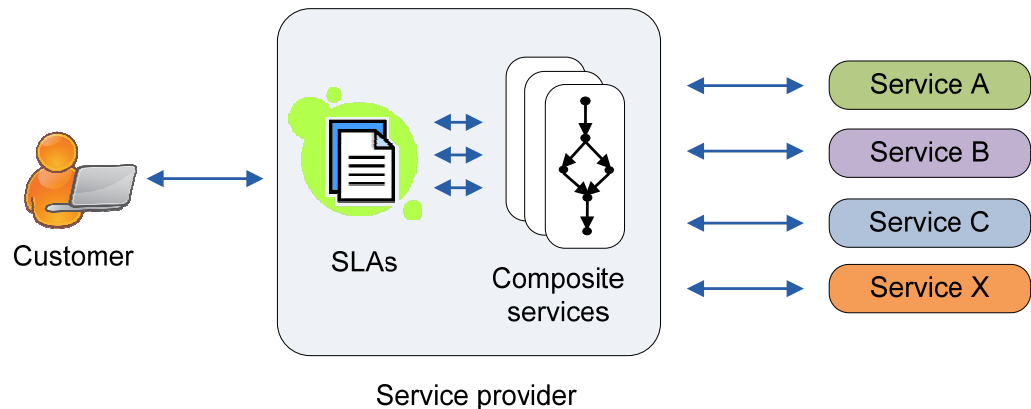


Figure 3.3. Idea of solution.

The consumer selects a composite service from a repository, based on its QoS specification. It is assumed that a functional search has already been performed. Then, based on the service's contract (SLA), the consumer decides whether to accept or not the usage conditions. If possible, a negotiation process between consumer and provider takes place, in order to adjust the contracts clauses. To avoid exposing QoS attributes' raw data, that may not be relevant to the consumer, some of them can be expressed using linguistic terms (e.g. low, medium, high) or considered as Business Level Objectives (BLOs) [131].

When the consumer accepts the contract offered by the provider and invokes the composite service, the provider has to ensure that the composition behaves as specified in the contract, avoiding violations and payment of compensation fees. This research is focussed in developing a model (mechanisms) to provide such environment, helping the provider to deliver the expected service, by maintaining/improving the QoS levels of the composition. The aim of these mechanisms is to react to situations where:

- *QoS levels can be improved.* Some of the QoS values of the composite service have been enhanced, providing the possibility of improving the global QoS.
- *QoS degradation.* The quality values of the composition have decayed and are far from expected.
- *Unavailable service.* The service is down or has no network connection.

- *Service fails.* The service does not finish its execution or sends an error message.

The use of service level agreements enables the establishment of contracts between consumers and providers, ensuring that both entities get the most of their interaction. However, this work is focused on adaptation mechanisms and the use of SLAs is out of the scope.

3.3.1. System Model

An overview of the system model proposed in this chapter is illustrated in Figure 3.4. It shows the system's core components: composition engine, adaptation manager, service binder, service selector, predictor, effectors and sensors; and their interactions. This model was implemented with the aim of creating an environment in which QoS aware and adaptive composition can be executed. Descriptions of the system's components are provided below.

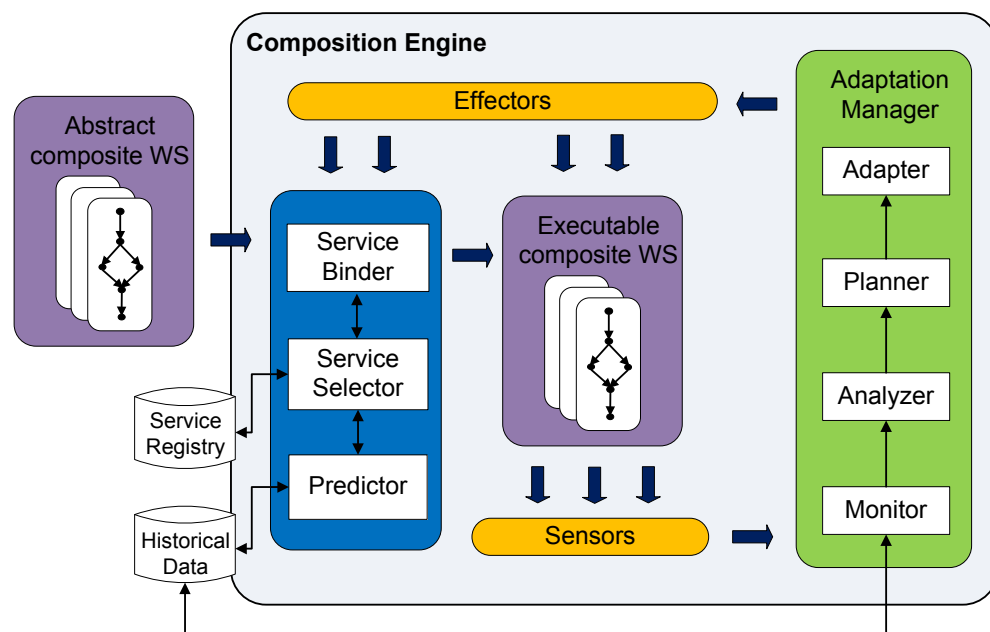


Figure 3.4. System model.

- *Service binder.* Binds dynamically each of the composition's tasks to executable services. These services are selected using functional and QoS criteria.
- *Service selector.* Searches in the registry those elements that fulfil the task's requirements.

- *Predictor*. Obtains estimates for the QoS attributes of the selected services by using predictive algorithms and a collection of historical QoS data.
- *Sensors*. Collect information about different events at run time and send it to the adaptation manager. Events are related to functional and quality aspects of the compositions' elements.
- *Adaptation manager*. Monitors and analyzes the behaviour of composite services at runtime. According to its analysis, determines when is necessary to perform changes, in order to improve/maintain the offered QoS of the compositions. Its components are based in the self-adaptive cycle for service composition described in section 2.5.2.
 - ◆ *Monitor*. Gathers data (collected by sensors) related to the behaviour of the services.
 - ◆ *Analyzer*. Analyzes and detects when is necessary to perform a change in the composite service.
 - ◆ *Planner*. Decides how to perform adaptation.
 - ◆ *Adapter*. Coordinates the changes to be performed on the composite services.
- *Effectors*. Apply the actions provided by the adaptation manager, enabling composite services to adapt at runtime.
- *Composition engine*. Executes the composite services (processes' definitions).

Composite services are considered to consist of a series of abstract tasks that will be linked to executable services at runtime. To obtain these services, for each task in the composite service, the service binder invokes the service selector with the desired characteristics that the component service should provide. The service selector performs a search into the service registry based on the provided functional requirements. For each of the pre-selected services (candidates), the service selector invokes the predictor to obtain its estimated QoS. A sub-set of candidates is then sent to the binder, along with their estimated QoS. The binder ranks these services and selects one to be invoked. During the execution of the composite

service, sensors capture information about the behaviour of the service and its components. Sensors send this information to the adaptation manager, which determines if adaptation is needed and the appropriate adaptation strategy. At the same time, QoS data is stored in the historical database. Finally, the adaptation manager sends the actions to be performed to the corresponding effectors, in order to maintain/improve the QoS of the composition.

3.3.2. QoS Model

Services that offer the same functionality may be associated with several QoS attributes [62],[65], providing different QoS levels. By evaluating these attributes within a set of services that share the same goals, consumers can search/select components to be used in their applications. In the first stage of this work, the quality attributes considered for each service are response time and cost. The use of other QoS parameters, energy consumption and availability, has been considered during this research. These parameters are included in the QoS models of the approaches described in further chapters.

- *Response time (Rt)*. Time consumed between the invocation and completion of the service operation [59].
- *Cost (C)*. Fee charged to the consumer when invoking a service [86].

Considering response time and cost enables the selection of faster and cheaper services, providing a competitive advantage [65]. Both parameters have been used in other approaches, like those presented in [3], [59], [86] and [132]. Assuming that a service (s) only contains one operation, its QoS (Q) can be defined using Eq. 3.1.

$$Q(s) = (Rt(s), C(s)) \quad (3.1)$$

To compute the values of these parameters at execution time, three situations have been considered within the composite service structure: single, sequential and concurrent service invocations. When computing the QoS parameters of a single service invocation, the QoS values of the activity that performs the invocation corresponds to the QoS of the invoked service, as defined in Eq. 3.2 and 3.3.

$$Rt(t_i) = Rt(s) \quad (3.2)$$

$$C(t_i) = C(s) \quad (3.3)$$

For activities in a sequential structure, the values of response time (Rt) and cost (C) are summed for the different activities with service invocations, as shown in Eq. 3.4 and Eq. 3.5, respectively.

$$Rt(P) = \sum_{i=1}^n Rt(t_i) \quad (3.4)$$

$$C(P) = \sum_{i=1}^n C(t_i) \quad (3.5)$$

For activities in a concurrent/parallel structure, the value of response time (Rt) is considered as the maximum response time of the completed activities; while value of cost (C) is the sum of the cost of the activities involved, as defined in Eq. 3.6 and Eq. 3.7, respectively.

$$Rt(P) = \max_{i=1, \dots, n} Rt(t_i) \quad (3.6)$$

$$C(P) = \sum_{i=1}^n C(t_i) \quad (3.7)$$

In this set of equations, the value of t_i corresponds to an activity (task) with a service invocation within the composite service P .

3.3.3. Service Selection Model

Estimation of QoS values is a key step during the service selection process. Estimated values are calculated using historical QoS data recorded from previous executions. This data is filtered, discarding values considered as outliers, and the average of the last N executions of the remaining subset is obtained. Concrete services are searched in the registry by name, assuming that this parameter includes/describes the service's functionality. The resulting set of candidate services is sorted according to the relationship between their estimated QoS values. Due to these attributes having different units of measurement, raw values are first normalized with natural logarithms. The overall quality score (W) for each service is then computed using the following formula:

$$W_i = w_1 \text{ est}T_i + w_2 \text{ est}C_i \quad (3.8)$$

Where:

$\text{est}T_i$ corresponds to the service estimated response time,

$\text{est}C_i$ corresponds to the service estimated cost,

w_1 and w_2 correspond to weights, where $0 \leq w_1, w_2 \leq 1$ and

$w_1 + w_2 = 1$.

Values for w_1 and w_2 are provided by the QoS evaluation heuristic described in the following section. The set of candidate services is ranked based on the values of W_i , and the service with the lowest value is selected.

3.3.4. QoS Optimization Model

Monitoring the execution of services is a critical task in the adaptation process. By monitoring and collecting data from services executions, based on their behaviour it is possible to take decisions about future actions [30]. As part of this work, at runtime QoS information is collected from service, task and process perspectives, where:

- *Service*. Corresponds to a concrete Web service.
- *Task*. Refers to an element within the composite service that invokes a service operation.
- *Process*. Corresponds to the entire composition (service workflow).

Response time is measured during each stage of the process, while cost is obtained from the WSDL¹ files of the services. The QoS values of a task are registered as an individual invocation and as the accumulated QoS of the composition at the time of executing the task. The optimization approach is based on the service selection model previously described. It uses variable weights and performs service selection on the obtained set of candidates. When the accumulated response time (or cost) of the previous activity in the process is less than expected, it provides some slack that can be used while selecting the next service in the process. The use of weights gives priorities to certain QoS parameter during the service selection phase,

¹ The WSDL standard was extended to include the service's QoS information.

which can help to enhance its values (e.g. a large weight assigned to cost enables the selection of a candidate service with low cost).

The heuristic presented in Figure 3.5 describes the QoS evaluation method applied during optimization. The notation used is shown as follows. Let,

- $T = \{t_1, t_2, \dots, t_n\}$ be the set of n tasks in process P .
- j be the task number, where $t_j \in T$.
- $T' = \{t'_1, t'_2, \dots, t'_k\}$ be the set of k ancestors of t_j , where $T' \in T$. When $j = 1$, then $T' = \{\emptyset\}$.
- $aRT, aRC, aestT, aestC$ be the accumulated values corresponding to real response time, real cost, estimated response time and estimated cost for a task.
- $S_j = \{s_1, s_2, \dots, s_m\}$ be the set of m services that can be used to implement t_j .
- i be the service number, where $s_i \in S_j$.
- $estT, estC$ be estimated QoS values corresponding to response time and cost for a service.
- w_1, w_2 be weights used to obtain the score of a service (see Eq. 3.8).
- w_1^0, w_2^0 be default values used to establish the weights, where $0 \leq w_1^0 < w_2^0 \leq 1$ and $w_1^0 + w_2^0 = 1$.
- $limitT, limitC$ be default values set as maximum difference between $aestT$ and aRT , and $aestC$ and aRC , respectively.

Before invoking a Web service operation for t_j , the ancestors T' for t_j are obtained (step 3). w_1 and w_2 are set initially to 0.5, enabling a service ranking with no preference (step 4). This is used in case there are no meaningful differences between the QoS values of P before t_j . If t_j is not the first task in P , this task has ancestors and QoS evaluation takes place (steps 6 to 20). Values within T' are sorted based on aRT (step 6). The task with the highest aRT value is selected and the differences between its estimated and real QoS values are obtained (steps 7 to 9). These values are compared to the maximum desired percentage of difference between real and estimated values. If the accumulated time is smaller than expected or the accumulated cost is higher than expected (step 10), weights are assigned giving priority to w_2 (steps 11 and 12). This enables the selection of a service with a smaller cost for t_j .

If there is no adaptation required based on time, the values within T' are sorted based on aRC (step 14). The task with the highest aRC value is selected and the differences between its estimated and real QoS values are obtained (steps 15 to 17). These values are compared to the maximum desired percentage of difference between real and estimated values. If the accumulated cost is smaller than expected or the accumulated time is higher than expected (step 18), weights are assigned giving priority to w_1 (steps 19 and 20), enabling the selection of a service with a smaller response time for t_j . Scores are obtained per each of the services within S_j (steps 21 and 22). Finally, S_j is sorted and the heuristic returns the service with the smaller score (steps 23 and 24).

```

SelectService( $t_j, T, S_j$ )
1   let  $t'$  be a task
2   let  $T'$  be an empty list
3    $T' = \text{ObtainAncestors}(t_j, T, T')$ 
4    $w_1 = w_2 = 0.5$ 
   // weights selection phase
5   if  $T'.length \neq 0$ 
6       sort  $T'$  by  $aRT$  descendent
7        $t' = T'[0]$ 
8        $difT = t'.aestT - t'.aRT$ 
9        $difC = t'.aestC - t'.aRC$ 
10      if  $difT \geq limitT \parallel - difC \geq limitC$ 
11           $w_1 = w_1^0$ 
12           $w_2 = w_2^0$ 
13      else
14          sort  $T'$  by  $aRC$  descendent
15           $t' = T'[0]$ 
16           $difT = t'.aestT - t'.aRT$ 
17           $difC = t'.aestC - t'.aRC$ 
18          if  $difC \geq limitC \parallel - difT \geq limitT$ 
19               $w_1 = w_2^0$ 
20               $w_2 = w_1^0$ 
   //score computation and service ranking phase
21  for  $i = 0$  to ( $S_j.length - 1$ )
22       $S_j[i].W = w_1 S_j[i].estT + w_2 S_j[i].estC$ 
23  sort  $S_j$  by  $W$  ascendent
24  return  $S_j[0]$ 

ObtainAncestors( $t, T, T'$ )
1   for  $j = 0$  to ( $T.length - 1$ )
2       if  $T[j]$  is ancestor of  $t$ 
3           insert  $T[j]$  into  $T'$ 
4   return  $T'$ 

```

Figure 3.5. QoS optimization heuristic.

After invoking the operation of the selected service, the obtained QoS values for service and task are stored in the historical database. Accumulated QoS per task are calculated using the formulas presented in equations 3.2 to 3.7.

3.4. Implementation

Implementation of the solution described in the previous section, was carried out extending the functionality provided by a java-based composition engine. It includes modifications to existing files and packages, and development of new components (service binder, service selector, predictor and adaptation manager). As a result, the engine provides the features required to execute QoS aware service compositions, according to the proposed QoS optimization model.

The diagram depicted in Figure 3.6 illustrates the main packages of the engine and their dependencies. It is derived from the system model described in section 3.3.1.

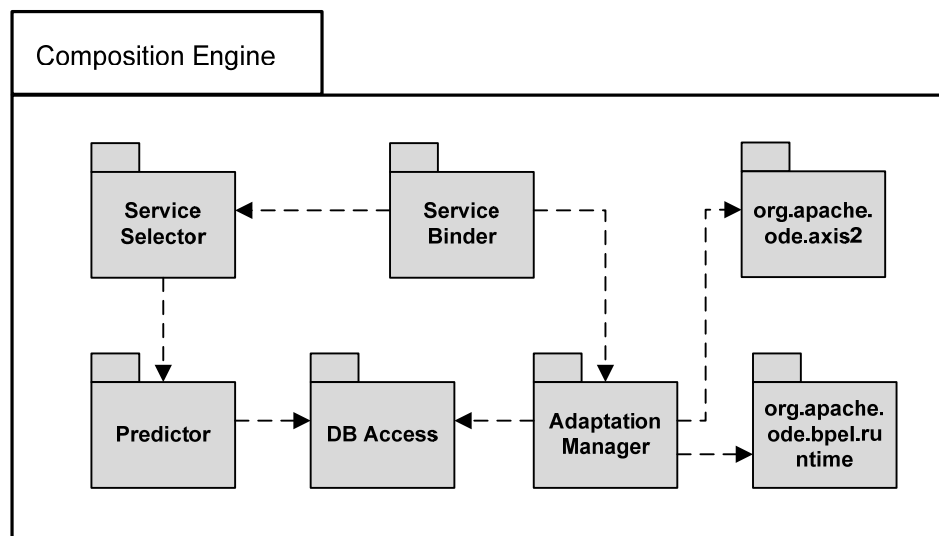


Figure 3.6. Packages diagram.

Interaction between packages is illustrated in Figure 3.7. When the composite service is being executed, before selecting a new service to bind to a task, candidate services are searched in the registry based on functional requirements. This activity is performed by classes within the *Service Selector*. For each of the services found in the registry, a prediction of QoS

values takes place (based on historical data). In this step, the *Predictor* has access to the database via *DBAccess*. The obtained predictions and service data (service name, WSDL's URL) are then sent to the *Service Binder*.

In the next step, the *Service Binder* interacts with the *Adaptation Manager*, in order to obtain the weights to be used during the service ranking process. To analyze and evaluate the behaviour of the composition, the *Adaptation Manager* needs information from the composite service behaviour. It accesses historical information via *DBAccess*, determines if adaptation is needed, and the weights to be used during service selection. Weights are then sent to the *Service Binder*, which applies them to rank the pre-selected services, and finally invoke the service situated in the higher position of the ranking.

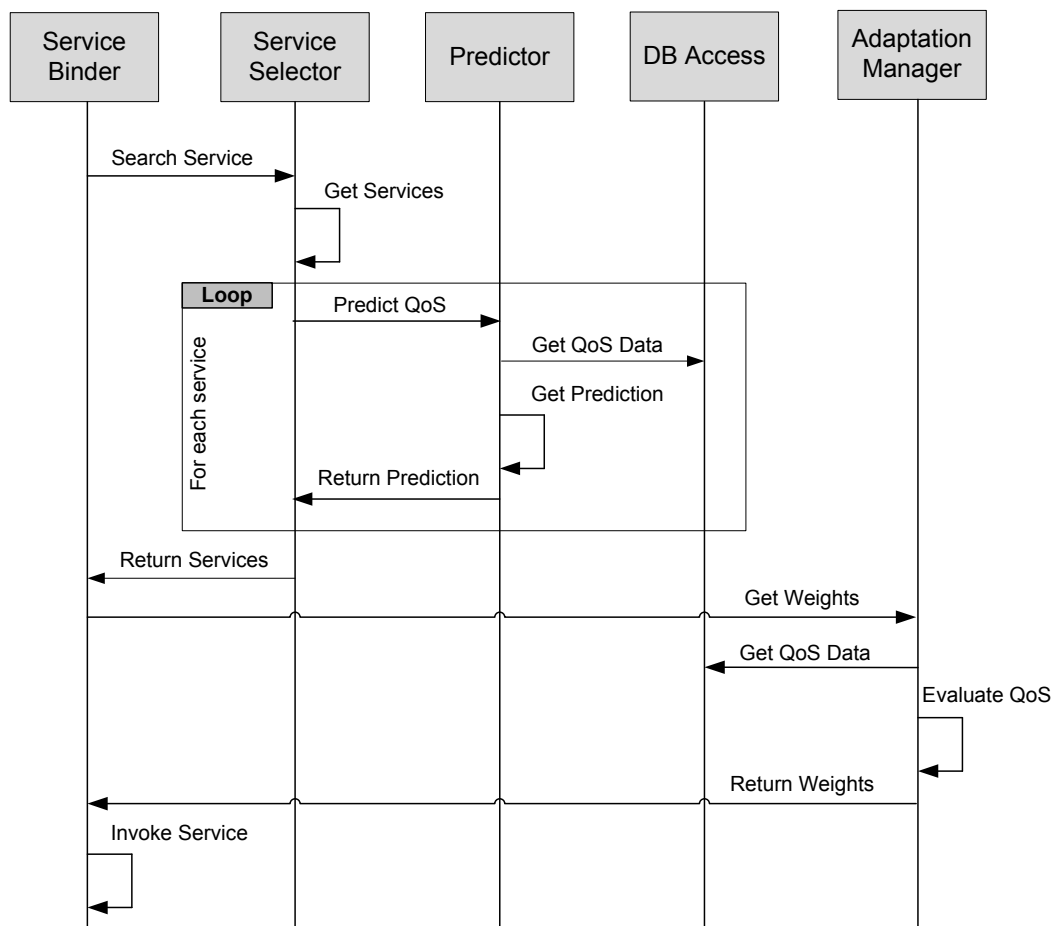


Figure 3.7. Components interaction.

Packages *org.apache.ode.axis2* and *org.apache.ode.bpel.runtime* correspond to original components of the composition engine. Even though they do not interact with other components during the service selection process, both have an important role in the monitoring process.

3.4.1. Composition Engine

In order to select the engine to use in this work, different tools that enable service composition were installed and tested (See Table 3.1). Based on the results of this exercise, it was necessary to have not only a composition engine, but also a designer. The parameters used to compare these tools include: ease of obtaining source codes, licensing and compatibility.

The selected tools were Apache ODE [133] and BPEL designer for eclipse [134]. Apache ODE is a BPEL engine that runs on standard servlet containers, like Apache Tomcat. It is an open source project that exposes its source codes online, enabling the extension of its functionality. BPEL designer is a plug-in that brings support for WS-BPEL on eclipse. Composition projects created on the designer are compatible with the structure of ODE. The use of these tools combined provides a design/execution environment for service compositions.

Table 3.1. Composition tools.

Tools	Installation requirements	Available source codes	Composition language	Licensing
ActiveBPEL engine [135]	Apache Tomcat	Java	BPEL4WS 1.1	GPL license
BPEL designer for eclipse [134]	Eclipse	---	WS-BPEL 2.0	Open source
Pi4soa designer [40]	Eclipse	---	WS-CDL	Open source
JBoss AS [136]	JBossESB Overlord CDL	Java	WS-CDL	LGPL license
JOpera [137]	Eclipse	---	---	Free with non commercial purposes
Apache ODE [133]	Apache Tomcat	Java	WS-BPEL 2.0	Apache license

Modifications and extensions to the original ODE sources were performed in order to provide the execution environment with monitoring, dynamic binding and adaptive capabilities. Monitoring features enable the collection of information from process and activity (task) perspectives. Processes' response time can be monitored within the class *PROCESS* (package *org.apache.ode.bpel.runtime*). When the execution of the BPEL

process ends, information is collected and registered in the historical database. From the task perspective, information can be obtained from the class *SoapExternalService* (package *org.apache.ode.axis2*). This class is in charge of performing service invocations. Dynamic binding and adaptive features are described in sections 3.4.2 and 3.4.5, respectively.

3.4.2. Service Binder

Dynamic binding can be performed using different techniques, as described in section 2.6.1. In order to develop the service binder component of the model, it was selected the use of a proxy service [102], as it can be linked to tasks defined in BPEL, and enables the invocation of executable services at runtime. The service binder is deployed on top of the composition engine as a Web service.

```

<bpel:copy>
  <bpel:from>
    <bpel:literal>
      <impl:fnCallService xmlns:impl="http://dynamicBinding"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <impl:values>impl:values</impl:values>
        <impl:values>impl:values</impl:values>
      </impl:fnCallService>
    </bpel:literal>
  </bpel:from>
  <bpel:to variable="DynamicProxyRequest" part="parameters">
  </bpel:to>
</bpel:copy>
<bpel:copy>
  <bpel:from><![CDATA[string('CreditCardChecking')]]>
  </bpel:from>
  <bpel:to>
    <![CDATA[$DynamicProxyRequest.parameters/ns:values[1]]>
  </bpel:to>
</bpel:copy>
<bpel:copy>
  <bpel:from part="payload" variable="input">
    <bpel:query
      queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      <![CDATA[tns:CardNumber]]>
    </bpel:query>
  </bpel:from>
  <bpel:to>
    <![CDATA[$DynamicProxyRequest.parameters/ns:values[2]]>
  </bpel:to>
</bpel:copy>

```

Figure 3.8. BPEL code that defines the XML input for the service binder.

At runtime, the composite service sends to the service binder an XML fragment with the value of the desired functionality and execution parameters, as shown in Figure 3.8. The value of the variable that holds the functionality details is obtained from the XML and used as input parameter during the service search process. Values of execution parameters are also obtained and sent as inputs during service invocation.

With the aim of selecting a specific service to bind to a task (based on the results of the service selector), QoS values of the pre-selected services are processed, enabling the ranking process (based on the weights values obtained from the adaptation manager). When the invoked service has finished its execution, information about its QoS levels is obtained and stored in the historical database. This information is used in the prediction stage of further compositions related to the same functionality.

3.4.3. Service Repository and Service Selector

In order to store information from different services, a repository was implemented configuring a UDDI registry using jUDDI v.3 [138]. jUDDI is a java-based implementation of UDDI that was created to integrate effectively with java application servers, like Tomcat. Because the selected composition engine was already deployed on top of Tomcat, the use of jUDDI was considered as a suitable technical solution.

In the literature there are different ways of searching services within a repository, some of them using semantics, where services are linked to their functionality using ontologies. However, developing a complex search engine is out of the scope of this research, and search is limited to a basic mechanism where services are evaluated based on keywords that describe their functionalities. The purpose of the service selector is finding services that match a functional requirement, and obtaining their QoS values (by interacting with the predictor), along with statistical details that are used by the service binder when computing the service's information.

3.4.4. Predictor

Estimations of QoS values are carried out by the predictor using the running average for the last 10 executions, after removing values considered as

outliers. The use of this predictive algorithm was decided after performing a set of experiments where different algorithms were evaluated and compared when obtaining estimated QoS of Web services. The evaluated algorithms include: single last observation (SLO), running average (RA), running average for the last 10 executions (RA-10) and low pass filter (LPF). Details about this experiment are discussed in appendix A.

The predictor provides the system with the capability of performing QoS estimations at service and task (activity) level, enabling the adaptation manager to evaluate the overall behaviour of the composition at different stages, and make decisions regarding adaptation actions.

3.4.5. Adaptation Manager

The adaptation manager consists of four packages that combined, evaluate the behaviour of composite services at runtime and enable QoS awareness and adaptation. Its core components and their dependencies are illustrated in Figure 3.9.

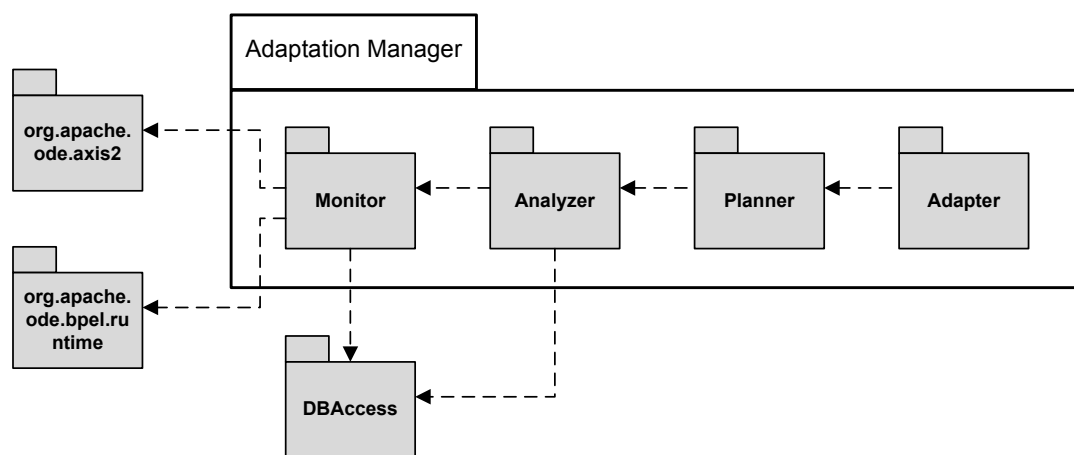


Figure 3.9. Package diagram - adaptation manager.

As mentioned earlier in this section, *org.apache.ode.bpel.runtime* and *org.apache.ode.axis2* correspond to elements within the composition engine and have been extended to work with the adaptation manager, as part of the monitoring mechanism. Information collected by these packages is collected by the *Monitor* and stored in the historical database via *DBAccess*. Implementation of the QoS evaluation heuristic (described in section 3.3.4) and support classes for the QoS optimization model, are distributed along the *Analyzer*, *Planner* and *Adapter*. The *Analyzer* interacts with the database

in order to obtain historical data, and combines this information with the new information collected by the *Monitor*, in order to evaluate the performance of the composition. The *Planner* and *Adapter* are in charge of deciding the adaptation strategy, obtain the weights to be used in the service selection process, and send the information to the corresponding modules. Adaptation strategies are not limited to QoS optimization, also include self-healing features that enable the composition to react to service unavailability and service failures. Details regarding the self-healing functionality are described in appendix B.

3.5. Evaluation

In order to assess the effectiveness of the proposed optimization mechanism, two sets of experiments were designed, executing a test case in two different environments. The first environment is setup within a local area network, while the second environment is setup on a wide area network. Experiments were carried out to address the following question:

- Is there any improvement in the global QoS when using variable weights during service selection as part of a self-optimization mechanism?

The work performed to provide an answer to this question is listed as follows:

- Assessment and comparison of the behaviour of the proposed optimization model vs. a baseline approach that does not use optimization.
- Assessment of the behaviour of the optimization approach when executing component services in local and remote environments.

3.5.1. Test Case

In the literature, there have been proposed several test cases (scenarios) to model service compositions, such as travel planning [62], order management [106], order fulfilments [66], DNA sequencing [65], etc. The test case used in this work, is a BPEL process that implements a travel

planning process. It validates a credit card, performs flight and hotel reservations in parallel, and finally invokes a car rental operation, as illustrated in Figure 3.10. For simplicity, the diagram only depicts activities that involve service invocations.

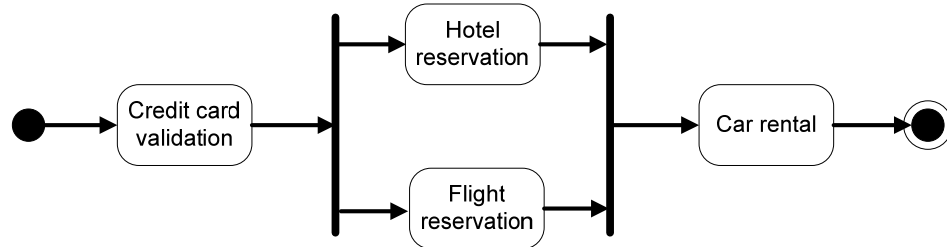


Figure 3.10. Travel planning process.

Per each of the tasks in the process, there are 9 candidate services that fulfil the required functionality and offer different QoS, giving a total of 36 services, distributed in 9 sets among the servers (nodes). These services were previously registered into the service registry (UDDI), and executed several times to populate the historical database and enable the estimation of their QoS attributes.

Table 3.2. QoS parameters configuration.

Experiment 1	Experiment 2	Set	Time delays (ms)	Cost
Node 1	Node 2	S1	0	120
		S2	350	80
		S3	200	100
Node 2	Node 3	S1	0	150
		S2	350	100
		S3	200	120
Node 3	Node 4	S1	0	100
		S2	350	60
		S3	200	80

The initial values of QoS parameters for the candidate services used in both experiments, are established based on the node where the service is running and the corresponding set. Delays are inserted on some of the service sets to obtain different response times, not only based on the network latency, but the Web services performance. This information is shown in Table 3.2.

The amount of information available per each service, before starting the execution of the composite services, corresponds to 1,000 records in the database. After performing each set of executions, information above 1,000 records is stored in external files and deleted from the database, in order to have the same information available at the beginning of each experiment.

3.5.2. Service Selection Based on Fixed Weights

A service selection mechanism based on fixed weights was implemented to be compared with the proposed optimization approach. It uses equation 3.8 (presented in section 3.3.3) and does not consider QoS optimization. For this approach the values for w_1 and w_2 are set equally to 0.5. The steps used to select a service are as follows:

1. Service's QoS data from previous executions is filtered in order to remove outliers.
2. Average of the last 10 executions is obtained per each QoS parameter.
3. Raw QoS values are normalized with natural logarithms.
4. Service's score is obtained using Eq. 3.8.
5. Services are ranked and the one with the smaller score is selected.

Results that correspond to the execution of the service selection mechanism based on fixed weights are labelled in further sections as "fixed weights approach".

3.5.3. First Stage of Evaluation

During the first stage of evaluation, the travel planning process was executed 50 times to analyze the behaviour of the optimization approach and evaluate its overall benefit. The maximum difference between estimated/real response time and cost was established in 10%. Weights provide priorities to the QoS attributes at the time of performing service selection, values for w_1^0 and w_2^0 (corresponding to the heuristic described in section 3.3.4) were set to 0.3 and 0.7, respectively. The process was also executed using the service selection mechanism based on fixed weights described in section 3.5.2.

3.5.3.1. Experimental Environment

The experimental environment, illustrated in Figure 3.11, consists of three nodes configured on a local area network. One computer with Windows Vista, 4GB RAM and one Intel core2 duo 2.1GHz processor (node 1); and two virtual machines with lubuntu 11.10, 512 Mb RAM and one processor (nodes 2 and 3). Node 1 hosts the BPEL engine (Apache ODE 1.3.4), service registry (jUDDI 3.0.4), historical database (MySQL 5.1.51) and one application server (Tomcat 6.0.26). Nodes 2 and 3 host one application server each (Tomcat 6.0.35). Web services are allocated in the application servers. Every node contains 3 sets of Web services. The travel planning process is hosted and invoked from Node 1.

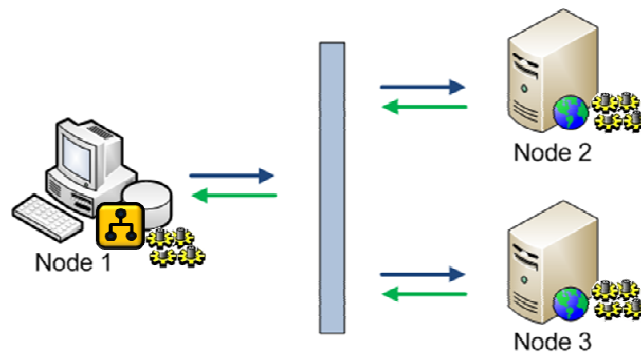


Figure 3.11. Experimental environment - LAN.

Based on the analysis of the behaviour of Web services found on the Internet, response time of the candidate services was modified by adding random delays generated with a log-normal distribution. The distribution and its input values were determined after executing 5 services 1,000 times, collect their response times and analyze the difference between each execution.

3.5.3.2. Experimental Results

Results show that the proposed approach provides a meaningful improvement on the global QoS of the compositions, when comparing with the fixed weights approach. Global QoS refers to the final values of the different QoS properties of the composite service.

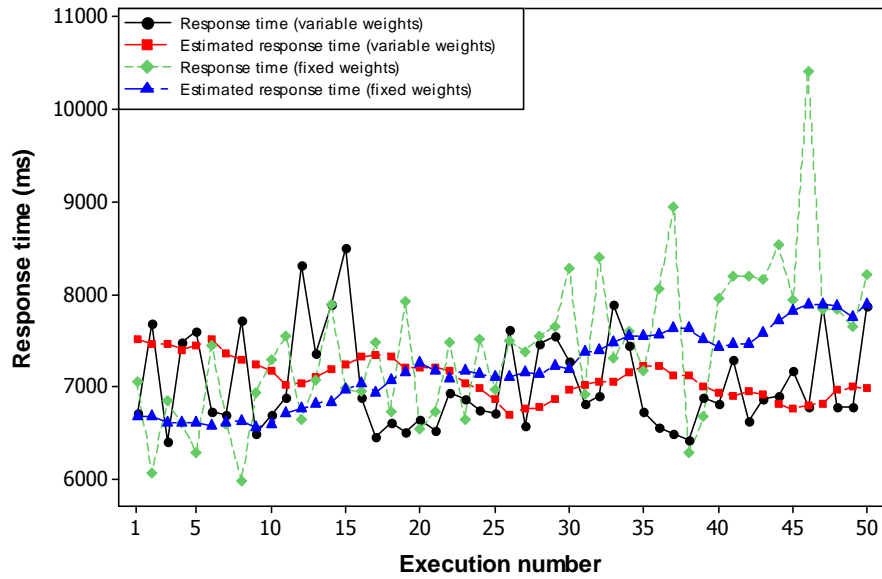


Figure 3.12. Response time comparison between variable and fixed weights approaches.

The plot depicted in Figure 3.12 shows that the measured response time of the composite service executed using the optimization approach (variable weights) is closer to the corresponding estimated values, as compared to the behaviour of the fixed weights mechanism, where most of the values are above the estimations. Measured average response time values correspond to 7,049ms and 7,416ms, where the proposed approach provides a mean reduction of 5%, a highest reduction of 14%, and standard deviation of 7.45%.

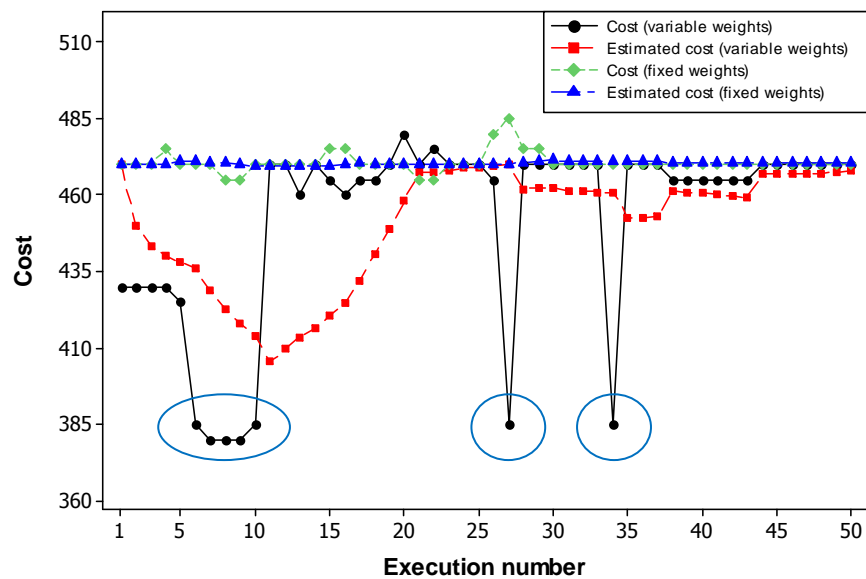


Figure 3.13. Cost comparison between variable and fixed weights approaches.

In contrast to the behaviour of response time, in the majority of the cases, cost estimations for the proposed approach are not close to the real measurements. As illustrated in Figure 3.13, most values are above estimations; nevertheless, there can be found some significant cost reductions, the highest being of 16%. Average cost value was 452, with a standard deviation of 6.8%. Some executions show large discrepancies (marked with circles), where the obtained composition cost is not close to the average. These situations are caused by low response time in some of the stages of the composite service, giving priority to cost (based on the QoS optimization heuristic), which will encourage the search of cheaper services in the next stage.

To obtain an overview of the compositions' behaviour, response time and cost values were normalized and related using simple additive weighting. For both QoS attributes, weights were established at 0.5. From a global perspective (illustrated in Figure 3.14), it can be noted that in most of the service executions, using the proposed optimization model provides smaller scores, which represents improvement in their QoS levels.

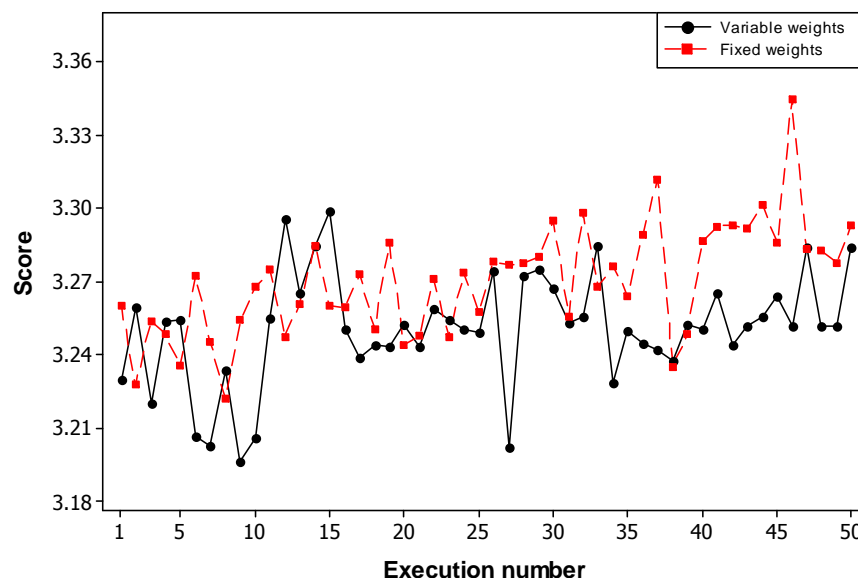


Figure 3.14. Score comparison between variable and fixed weights approaches.

3.5.4. Second Stage of Evaluation

During the second stage of evaluation, the travel planning process was executed 50 times to analyze the behaviour of the optimization approach and evaluate its overall benefit. Configuration values used for the

optimization heuristic were the same used in the first stage of evaluation (described in section 3.5.3). The maximum difference between estimated/real response time and cost was established in 10%, and the values for w_1^0 and w_2^0 were set to 0.3 and 0.7, respectively. The process was also executed using the service selection mechanism based on fixed weights described in section 3.5.2.

This stage of evaluation was performed using services deployed in a remote environment. Due to the randomness added by the use of a wide area network, the experiment (set of 50 runs) was performed 3 times, to evaluate the consistency of the results based on statistical analysis.

3.5.4.1. Experimental Environment

The experimental environment, depicted in Figure 3.15, consists of 4 nodes configured on a wide area network, distributed between United Kingdom and Germany, with estimated values for bandwidth and latency around 32 Mbit/s and 29ms, respectively.

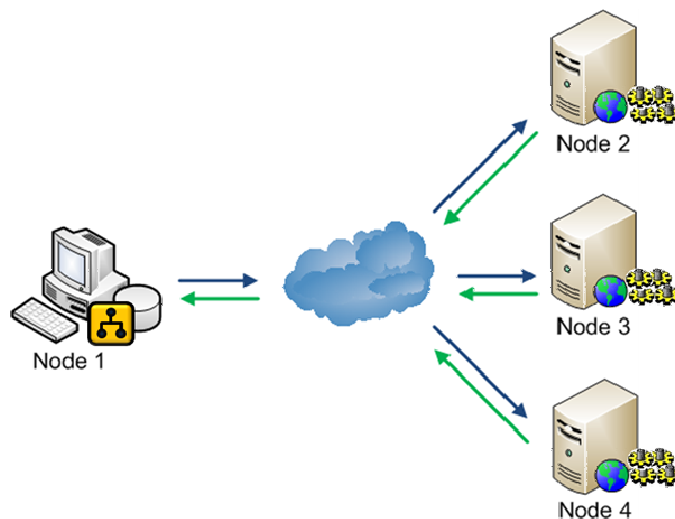


Figure 3.15. Experimental environment - WAN.

Node 1 is a computer with Windows Vista, 4GB RAM and one Intel core2 duo 2.1GHz processor (located in United Kingdom). This node hosts the BPEL engine (Apache ODE 1.3.4), service registry (jUDDI 3.0.4) and historical database (MySQL 5.1.51). It is in charge of coordinate the execution of the compositions and record all the gathered information. Nodes 2 to 4 are virtual machines setup on remote servers (located in Germany), each of the VM's uses Debian Squeeze x86 and 1GB RAM.

These nodes host one application server (Tomcat 6.0.35.0) each, which contains 3 sets of Web services.

3.5.4.2. Experimental Results

A similar behaviour was obtained when collecting the results of the second set of experiments. The proposed optimization model provides a significant improvement on the global QoS over the fixed weights approach.

When analyzing the collected results, measured average response time values correspond to 3,277ms and 3,422ms. The proposed approach (variable weights) provides a mean reduction of 4.5% with a mean standard deviation of 17%, and a 95% confidence interval between 3,178.5ms and 3,375.6ms. It presents a more stable behaviour, without showing high peaks, as compared to the fixed weights approach, as shown in Figure 3.16.

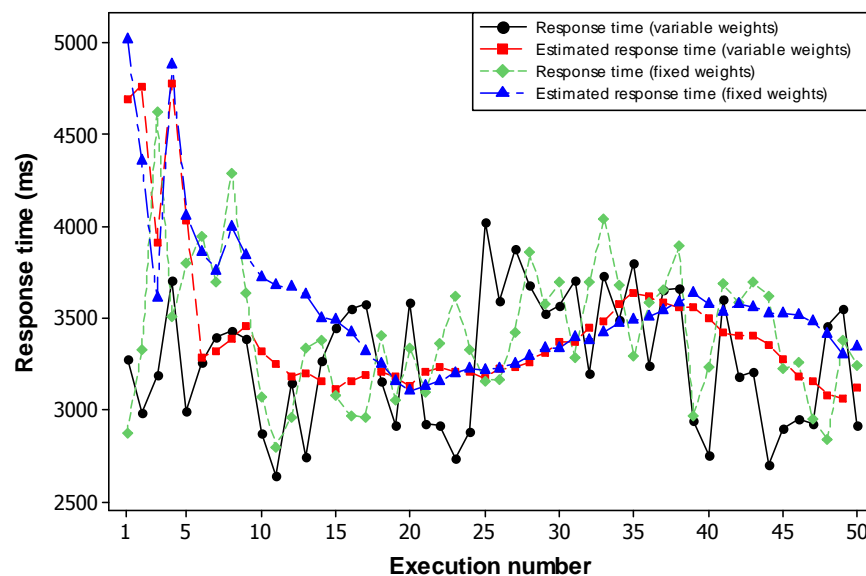


Figure 3.16. Response time comparison between variable and fixed weights approaches.

Differences among the 3 sets of executions are illustrated in Figure 3.17. The variation between response time values collected in experiments 1 and 2 (4,000ms approximately), was caused by the execution of virtual machines in the computer where experiment 1 was performed, which decreased the performance of the experimental environment.

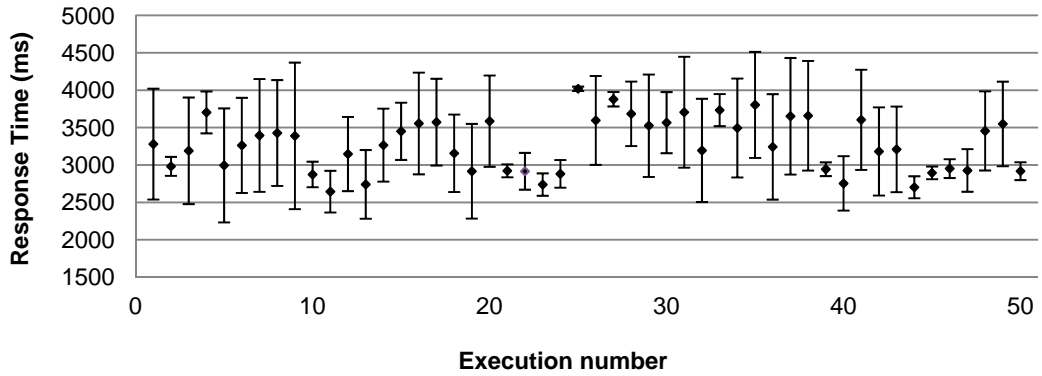


Figure 3.17. Response time evaluation - differences between executions.

In terms of cost, there is a significant mean reduction of 11.7% with a standard deviation of 14.14%, and a 95% confidence interval between 420.64 and 445.50. Behaviour of the composition cost when using the fixed weights approach is closer to its estimated values as compared with the optimization approach. This mechanism can encourage the invocation of the same set of services at runtime.

The following plots depict the behaviour of the composition’s cost. Figure 3.18 illustrates a comparison between the two approaches; while Figure 3.19 summarizes the standard deviations values of the 3 different sets of 50 executions using error bars.

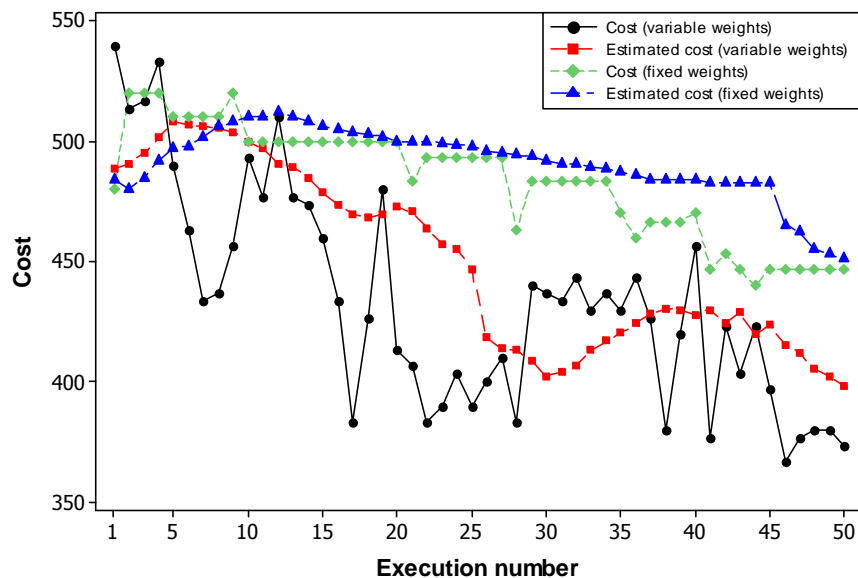


Figure 3.18. Cost comparison between variable and fixed weights approaches.

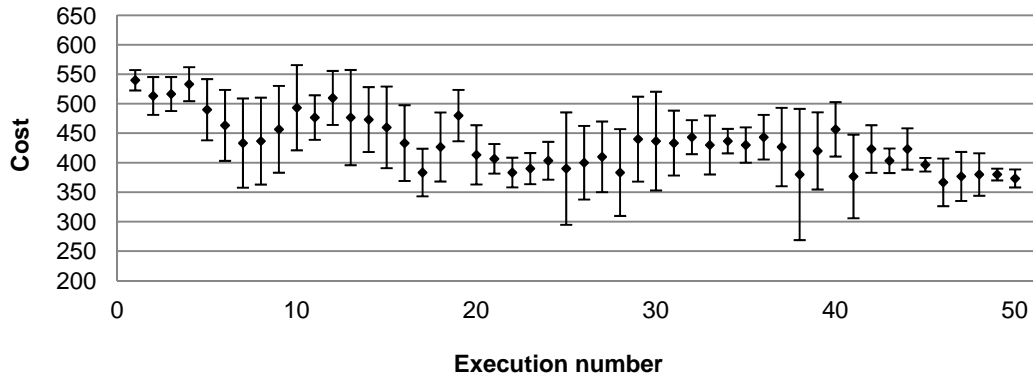


Figure 3.19. Cost evaluation - differences between executions.

Similar to the results presented in section 3.5.3.2, the score values for both approaches were computed using simple additive weighting. The plot in Figure 3.20 contrasts their behaviour. It can be noted that using optimization during the execution of composite services improves the QoS values for most of the service executions.

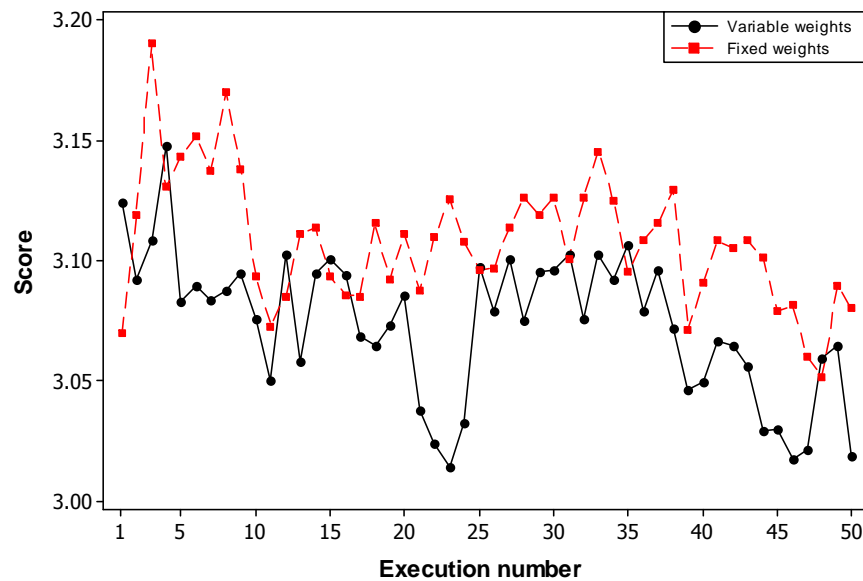


Figure 3.20. Score comparison between variable and fixed weights approaches.

3.5.5. Discussion

The use of weights with different values (small and large) establishes priorities during the service selection phase. A large weight value provides a high priority to its related QoS parameter. On the other hand, a small weight value provides a low priority to its related QoS parameter. For example, when the composition has achieved a response time smaller than expected

and its cost is higher than expected (according to historical values), a large weight is assigned to cost and a small weight to response time, encouraging the selection of a service with a high estimated response time and small estimated cost. This incurs in a trade-off that decreases the achieved reductions obtained in terms of response time, but enhances the global QoS of the composition by reducing the overall cost.

Results collected during the experimental stage indicate that the use of the proposed optimization model helps to obtain meaningful improvements regarding the global QoS of the test case scenario, with reductions up to 14% in response time and 16% in cost. In terms of performance, the use of the proposed model causes an average increment of 480ms in the invocation time per task (information obtained using a database with 10 candidate services and 100 records per service). Overheads increase following a quadratic model. This behaviour was determined after performing various sets of executions increasing the number of candidate services and analyzing the measured execution time (see Figure 3.21).

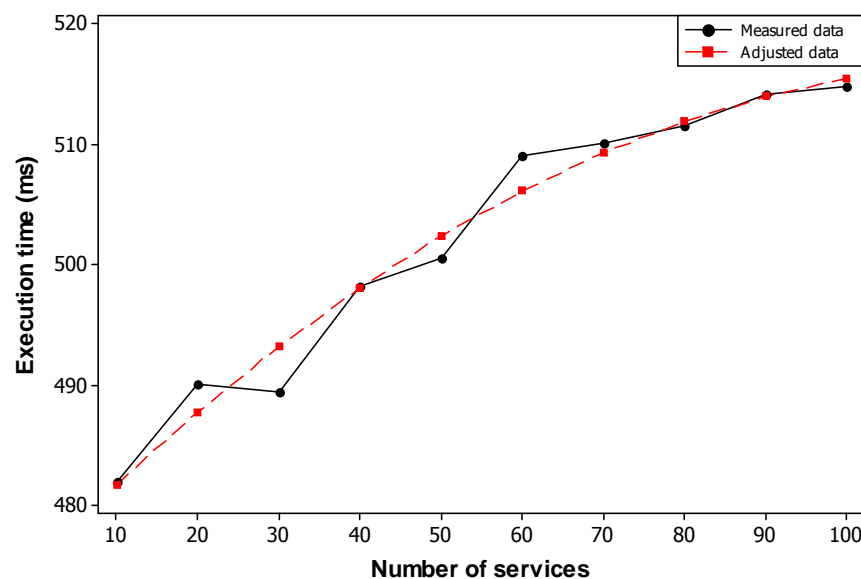


Figure 3.21. Execution time for different number of available candidate services.

In the scenarios used during the experimental stage, component services may fail during short periods of time. This information can be filtered by removing outliers before performing QoS prediction (as described in section 3.4.4). Situations with long periods of service malfunctioning were not considered.

Further assessment of the results shown in section 3.5 will be presented in chapter 6, along with a comparison between the proposed optimization model and relevant related work.

3.6. Summary

This chapter has presented a QoS optimization model for service composition. It has provided the motivation behind the development of this work, illustrated with a service composition scenario, followed by a discussion about approaches within the self-adaptation area in service composition.

An outline of the proposed solution is described from a general point of view. This is followed by a detailed portrayal of the different elements contained within the solution, system and models. Relevant implementation aspects are then explained, along with their interaction. Finally, the evaluation of the proposed model is detailed. It includes the description of experimental objectives, experiments and results.

The environment presented in this chapter enables the execution of composite services with QoS aware and adaptive capabilities. However, it performs changes every time there is a significant variation in the measured QoS, and is limited to the use of two parameters (response time and cost). To overcome these limitations, the following chapter will describe a QoS optimization model based on fuzzy logic, which extends the approach described within this chapter by implementing a decision making tool that evaluates the need of adaptation, and enables the use of more than two parameters.

Chapter 4

A Fuzzy Logic Based QoS Optimization Model for Service Composition

This chapter introduces a QoS optimization model for service composition based on fuzzy logic. The ideas that motivate the development of the model are explained, followed by a discussion on related approaches. The proposed solution is described, which includes the extensions performed to the models presented in the previous chapter. Implementation details are then provided. Finally, an evaluation of the model is presented, covering the experimental setup and results.

4.1. Motivation

Adaptation mechanisms aim to target situations where the behaviour of composite services deviated from what the consumer is expecting. Nevertheless, triggering adaptation after every variation in the behaviour of the composition does not warranty the best possible QoS values. Adaptation actions come with a cost [139], which can influence the application's QoS. The cost of performing a change can be at some point higher than the expected benefits. Reason why, before executing any adaptation action, it is important to consider the following questions:

- Is adaptation needed?
- What is the benefit of adaptation?
- When does the composite service need to adapt?
- What is the cost of adaptation?

In addition, it is important to detect which QoS parameters are affected when the system adapts (e.g. response time, cost, etc.) and consider the utility of change, which represents the relationship between cost and benefit [140].

Aiming to give an answer to some of the questions listed above, the work described in this chapter proposes the assessment of the behaviour of the composition, in order to determine the benefit of adaptation; and based on this value, decide whether adaptation is needed or not. The benefit of adaptation is obtained by analyzing the relationship between the values of the QoS parameters, during the different stages of the composite service execution.

4.2. Decision Support Systems in Service Selection and Service Composition

Different decision support methods used in autonomic computing solutions have been applied in the Web services field, aiming to provide new strategies to facilitate activities related to the Web service life cycle, like service selection and composition. Some of these methods include: genetic algorithms [107], [141], [142], [143], [144]; reinforcement learning [145], [146], [147]; decision trees [148]; and fuzzy logic [149], [150], [151], [152]; among others.

Because of its nature for solving problems and producing solutions for management purposes, fuzzy logic has been applied in different fields like networks, control systems and mobile applications. Examples of works that use fuzzy logic as a support tool in the context of Web services are presented as follows:

- The approach presented in [149] uses fuzzy logic for the selection of service adaptation strategies in service-based applications. The fuzzy systems applied in the selection process are based on: the overall QoS values, importance of QoS and cost of service substitution; and implement fixed membership functions and fuzzy rules defined by experts.
- The solution proposed in [150] applies a fuzzy decision making model to locate and select services based on customer's preference or satisfaction degree. The approach generates a dynamic ranking of services available on the market, based on different QoS parameters. It considers functional and non-functional service properties.

- The approach presented in [151] proposes a generic model based on fuzzy logic for representing and evaluating non-functional properties of composite services. It aims to enable the selection of service compositions fitting the user's requirements. The service behaviour (non-functional properties) is obtained by analyzing observations from previous executions.
- The methodology described in [152] performs service selection by combining imprecise QoS constraints (defined by the customer) and real QoS data (provided by the service over time). It relies on fuzzy logic, and uses fuzzy terms that are defined dynamically based on the service QoS values.

4.3. Proposed Solution

Fuzzy logic is an approximate reasoning technique suitable to deal with uncertainty [125], which can be used to evaluate imprecise parameters in software systems. In order to assess the behaviour of the composition, this research proposes the use of fuzzy logic as a tool to support the decision making process, helping determining whether adaptation is needed or not, and how to perform the service selection process. This is achieved using two fuzzy support systems. The first system assesses the QoS values of the composite service on each step of the composition, using the global QoS measured after the execution of the previous task and historical QoS data. The system takes the QoS parameters as inputs and based on fuzzy rules provides the benefit of adaptation. The second system determines the weights to apply to the different QoS attributes in the service selection process. It uses the value of the benefit of adaptation and the errors between estimated and measured QoS as inputs, providing as a result the values of the weights to be used during service selection.

The environment presented in chapter 3 enables the execution of composite services with QoS aware and adaptive capabilities. However, it does not consider the evaluation of the benefit of adaptation. In order to perform such evaluation, its QoS model and optimization model were modified, including the use of the fuzzy support systems described above.

4.3.1. QoS Model and Service Selection Model

The QoS model and service selection model used in this approach, extend the models described in sections 3.3.2 and 3.3.3. As a result, the quality parameters considered for each service are response time, cost and energy consumption.

- *Response time (Rt)*. Time consumed between the invocation and completion of the service operation [59].
- *Cost (C)*. Fee charged to the consumer when invoking a service [86].
- *Energy consumption (Ec)*. Amount of power consumed by a server over a period of time [153].

Energy consumption has been selected as the third parameter because of the importance of energy efficiency when managing computing infrastructure and services. The amount of energy used by data centres has not only economical but also environmental impacts. Energy efficiency is becoming a key topic due to high energy costs and governments' pressure to reduce carbon footprints [154].

Assuming that a service (s) only contains one operation, its QoS (Q) can be defined using Eq. 4.1.

$$Q(s) = (Rt(s), C(s), Ec(s)) \quad (4.1)$$

Computation of energy consumption is based on three situations within the composite service structure: single, sequential and concurrent service invocations, and is similar to response time and cost, as described in section 3.3.2. When computing the energy consumption (Ec) of a single service invocation, the energy consumption value of the activity that performs the invocation corresponds to the Ec of the invoked service, as shown in Eq. 4.2. For activities in sequential and concurrent/parallel structures, the value of Ec is summed for the different activities with service invocations, as defined in Eq. 4.3.

$$Ec(t_i) = Ec(s) \quad (4.2)$$

$$Ec(P) = \sum_{i=1}^n Ec(t_i) \quad (4.3)$$

In this set of equations, the value of t_i corresponds to an activity (task) with a service invocation within the composite service P .

Estimation of QoS values is a key step during the service selection process. Estimated values are calculated using historical QoS data recorded from previous executions. This data is filtered, discarding values considered as outliers and the average of the last N executions of the remaining subset is obtained (as performed in the model described in section 3.3.3).

Concrete services are searched in the registry by name, assuming that this parameter includes/describes the service's functionality. The resulting set of candidate services is sorted according to the relationship between their estimated QoS values. Due to these attributes having different units of measure, their raw values are normalized before being processed and ranked. The following formula is used to normalize the values of response time, cost and energy consumption, which are negative parameters (lower the value, higher the quality).

$$V_i = \frac{\max_i - q_i}{\max_i - \min_i} \quad (4.4)$$

Where:

\max_i corresponds to the maximum value of the evaluated QoS parameter,

\min_i corresponds to the minimum value of the evaluated QoS parameter,

q_i corresponds to the estimated value for the next execution.

When $\max_i = \min_i$, then $V_i = 1$.

After normalizing the values, the overall quality score (W) for each service is computed using Eq. 4.5.

$$W_i = w_1 \text{est}T_i + w_2 \text{est}C_i + w_3 \text{est}E_i \quad (4.5)$$

Where:

$\text{est}T_i$ corresponds to the service estimated response time,

$\text{est}C_i$ corresponds to the service estimated cost,

$\text{est}E_i$ corresponds to the service estimated energy consumption,

w_1, w_2 and w_3 correspond to assigned weights, where $0 \leq w_1, w_2, w_3 \leq 1$ and $w_1 + w_2 + w_3 = 1$.

Values for w_1 , w_2 and w_3 are provided by the QoS evaluation heuristic described in the following section. The set of candidate services is ranked based on the values of W_i , and the service with the highest value is selected.

4.3.2. QoS Optimization Model

Similar to the model presented in section 3.3.4, QoS information is collected from service, task and process perspectives, where service corresponds to a concrete Web service; task to an element within the composite service that invokes a service operation; and process to the entire composition (service workflow). Response time is measured during each stage of the composite service execution, while cost and server's power consumption are obtained from the WSDL¹ files of the services. The QoS values of a task are registered as an individual invocation and as the accumulated QoS of the composition at the time of executing the task.

The proposed optimization approach uses the service selection model previously described and it is based on fuzzy support systems to assess the QoS values of the composition (in order to decide if adaptation is needed or not), and to establish the weights to be used during the service selection process. It considers situations where a number of the accumulated QoS values of the previous activity in the process are better than expected, providing some slack that can be used while selecting the next service in the process, improving other QoS parameters.

The idea of using fuzzy logic is to understand the relationship between the QoS values of the composite service and the need of adaptation. QoS parameters are expressed using linguistic variables.

4.3.2.1. Fuzzy Logic Based Decision Support Systems

Fuzzy logic is a method based on multi-valued logic which aims to formalize approximate reasoning [125]. It is used to deal with different types of uncertainty in knowledge-based systems. Some of the relevant characteristics of fuzzy logic are fuzzy sets, linguistic variables and fuzzy rules.

¹ The WSDL standard was extended to include the service's QoS information.

- *Fuzzy set.* Is a collection of objects characterized by a membership function with a continuous grade of membership which can be ranged between zero and one [155].
- *Linguistic variable.* Is a type of variable that uses words instead of numbers to represent its values (e.g. slow, medium, fast) [125]. The values used to define linguistic variables are called terms and the collection of terms is called term set.
- *Fuzzy rules (IF – THEN).* Are used to represent human knowledge in fuzzy systems. A fuzzy *IF – THEN* rule is a conditional statement structured as [156]:

IF < fuzzy proposition >, THEN < fuzzy proposition >

where a *< fuzzy proposition >* is a statement used to associate linguistic variables and terms.

The basic configuration of a fuzzy system is illustrated in Figure 4.1. During the execution of a fuzzy system, crisp inputs are converted to linguistic variables, this process is known as fuzzification. The variables values are then evaluated using fuzzy rules, generating the linguistic values for the outputs. Finally, the defuzzification method uses these values to obtain crisp outputs values.

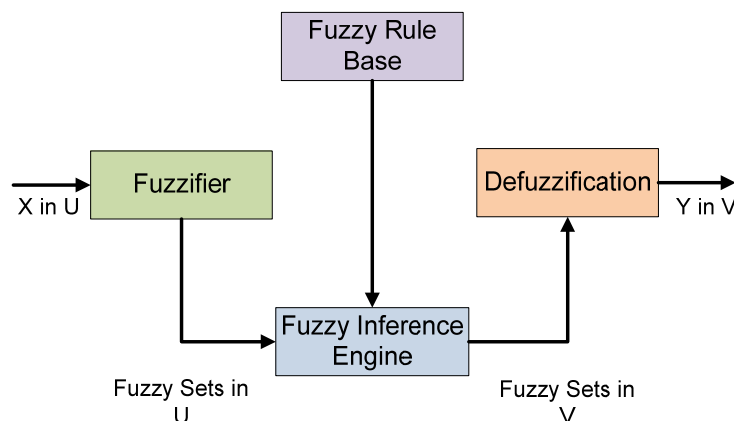


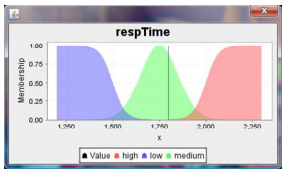
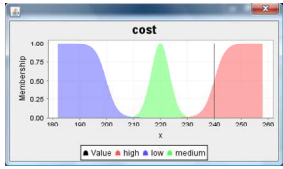
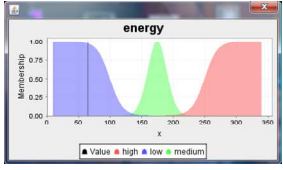
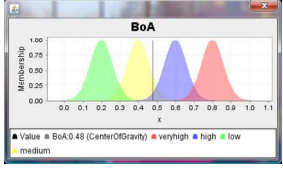
Figure 4.1. Basic configuration of fuzzy systems with fuzzifier and defuzzifier [156].

Fuzzy systems have been applied in different areas, mainly focussed in control and management problems [156]. In this research, two fuzzy support systems have been defined to 1) establish the benefit of adaptation, 2) obtain the weights to be used during service selection. Each of these systems uses its own linguistic variables and rules.

The first system assesses the QoS values of the composite service during each task of the composition. It uses as inputs the QoS values collected from the composite service prior to the moment of selecting a new service. The defined input variables are *response time*, *cost* and *energy consumption*, which are expressed with three terms *low*, *medium* and *high*. To establish these terms for each of the linguistic variables, an interval is defined at runtime using data collected from previous executions. Historical data is analyzed, obtaining maximum/minimum values and standard deviations from each of the QoS parameters. Sigmoidal functions (open to the left and right) are used to define the low and high terms, while a Gauss function is used to define the medium term.

The system takes the inputs and based on the corresponding fuzzy rules, provides the estimated benefit of adaptation. Four different levels of benefit of adaptation (*low*, *medium*, *high* and *very high*) were established, falling in the interval [0, 1], and defined with Gauss functions. The definition of the fuzzy variables is provided in Table 4.1.

Table 4.1. Fuzzy variables definition.

Variable	Terms	Type	Functions
Response time	Low = sigm ($-0.1, \min$) Medium = gauss (avg, std) High = sigm ($0.1, \max$)	Input	
Cost	Low = sigm ($-0.1, \min$) Medium = gauss (avg, std) High = sigm ($0.1, \max$)	Input	
Energy consumption	Low = sigm ($-0.1, \min$) Medium = gauss (avg, std) High = sigm ($0.1, \max$)	Input	
Benefit of adaptation (BoA)	Low = gauss ($0.2, 0.05$) Medium = gauss ($0.4, 0.05$) High = gauss ($0.6, 0.05$) Veryhigh = gauss ($0.8, 0.05$)	Output	

Where:

std is the standard deviation (after filtering outliers),

max is the maximum value obtained from the database (after filtering outliers),
min is the minimum value obtained from the database (after filtering outliers),
avg is the average value between maximum and minimum.

Four compound rules were constructed combining the input variables and their relationship with the different levels of benefit of adaptation. These rules describe the scenarios that can take place at runtime. The following table shows the rules used to obtain the benefit of adaptation.

Table 4.2. Benefit of adaptation related fuzzy rules.

1	IF (respTime IS high AND cost IS low AND energy IS low) OR (respTime IS low AND cost IS high AND energy IS low) OR (respTime IS low AND cost IS low AND energy IS high) THEN BoA IS veryhigh
2	IF (respTime IS high AND cost IS medium AND energy IS low) OR (respTime IS high AND cost IS low AND energy IS medium) OR (respTime IS medium AND cost IS high AND energy IS low) OR (respTime IS medium AND cost IS low AND energy IS high) OR (respTime IS low AND cost IS high AND energy IS medium) OR (respTime IS low AND cost IS medium AND energy IS high) THEN BoA IS high
3	IF (respTime IS high AND cost IS medium AND energy IS medium) OR (respTime IS medium AND cost IS high AND energy IS medium) OR (respTime IS medium AND cost IS medium AND energy IS high) OR (respTime IS medium AND cost IS medium AND energy IS low) OR (respTime IS medium AND cost IS low AND energy IS medium) OR (respTime IS medium AND cost IS low AND energy IS low) OR (respTime IS low AND cost IS medium AND energy IS medium) OR (respTime IS low AND cost IS medium AND energy IS low) OR (respTime IS low AND cost IS low AND energy IS medium) THEN BoA IS medium
4	IF (respTime IS high AND cost IS high AND energy IS high) OR (respTime IS high AND cost IS high AND energy IS medium) OR (respTime IS high AND cost IS high AND energy IS low) OR (respTime IS high AND cost IS medium AND energy IS high) OR (respTime IS high AND cost IS low AND energy IS high) OR (respTime IS medium AND cost IS high AND energy IS high) OR (respTime IS medium AND cost IS medium AND energy IS medium) OR (respTime IS low AND cost IS high AND energy IS high) OR (respTime IS low AND cost IS low AND energy IS low) THEN BoA IS low

The second system uses the value of the benefit of adaptation (output of the first system) and the errors between the estimated and the measured QoS as inputs. The error value is computed per each parameter using Eq. 4.6.

$$e(p_i) = \frac{x(p_i) - x_0(p_i)}{x_0(p_i)} \quad (4.6)$$

Where:

$x(p_i)$ is the estimated data,

$x_0(p_i)$ is the real measured data.

Input variables corresponding to the QoS errors are expressed with three terms: *low*, *medium* and *high*, falling in the interval [-1, +1]. Benefit of adaptation is expressed with four terms, as defined in the first fuzzy system. By evaluating the different errors and the benefit of adaptation, the system provides the values to be used as weights during the service selection process. Output variables (*response time weight*, *cost weight* and *energy consumption weight*) are expressed with five terms: *very low*, *low*, *medium*, *high* and *very high*, falling in the interval [0,1] and are defined using Gauss functions.

Parameters settings for both fuzzy systems were defined based on values obtained after performing several tests with different configurations.

4.3.2.2. QoS Optimization Heuristic

The heuristic presented in Figure 4.2 describes the QoS evaluation method applied during optimization, which involves the use of the fuzzy systems previously described. The notation used in the heuristic is shown as follows. Let,

- $T = \{t_1, t_2, \dots, t_n\}$ be the set of n tasks in process P .
- j be the task number, where $t_j \in T$.
- $T' = \{t'_1, t'_2, \dots, t'_k\}$ be the set of k ancestors of t_j , where $T' \in T$. When $j = 1$, then $T' = \{\emptyset\}$.
- $aRT, aRC, aRE, aestT, aestC, aestE$ be the accumulated values corresponding to real response time, real cost, real energy consumption, estimated response time, estimated cost and estimated energy consumption for a task.

- eT, eC, eE be the error values corresponding to response time, cost and energy consumption for a task (see Eq. 4.6).
- $S_j = \{s_1, s_2, \dots, s_m\}$ be the set of m services that can be used to implement t_j .
- i be the service number, where $s_i \in S_j$.
- $estT, estC, estE$ be estimated QoS values corresponding to response time, cost and energy consumption for a service.
- w_1, w_2, w_3 be weights used to obtain the score of a service (see Eq. 4.5).
- BoA be the value corresponding to the benefit of performing adaptation.
- fS^1, fS^2 be fuzzy systems (described in section 4.3.2.1).

Before invoking a Web service operation for t_j , the ancestors T' for t_j are obtained (step 3). w_1, w_2 and w_3 are set initially to 0.333, enabling a service ranking with no preference (step 4). This is used in case there are no meaningful differences between the QoS values of P before t_j . If t_j is not the first task in P , this task has ancestors and QoS evaluation takes place (steps 6 to 25). Values within T' are sorted based on aRT (step 6). The task with the highest aRT value is selected, its accumulated real response time is retrieved, and the error between its estimated and real response time is obtained (steps 7 to 9). Values within T' are sorted based on aRC (step 10). The task with the highest aRC value is selected, its accumulated real cost is retrieved, and the error between its estimated and real cost is obtained (steps 11 to 13). Values within T' are sorted based on aRE (step 14). The task with the highest aRE value is selected, its accumulated real energy consumption is retrieved, and the error between its estimated and real energy consumption is obtained (steps 15 to 17).

The accumulated real values are set as inputs for fS^1 (step 18). BoA is obtained and evaluated (steps 19 and 20); if it is medium or higher, then there is a need for adaptation. When adaptation is needed, the system determines the new weights to be used during the service selection process. This action is performed by fS^2 (step 21). The values of w_1, w_2 and w_3 are retrieved and adjusted, to fulfil the restriction $w_1 + w_2 + w_3 = 1$ (steps 22 to 25). Scores are obtained per each of the services within S_j (steps 26 and 27). Finally, S_j is sorted and the heuristic returns the service with the higher score (steps 28 and 29).

```

SelectService( $t, T, S_j$ )
1   let  $t'$  be a task
2   let  $T'$  be an empty list
3    $T' = \text{ObtainAncestors}(t, T, T')$ 
4    $w_1 = w_2 = w_3 = 0.333$ 
   // weights selection phase
5   if  $T'.length \neq 0$ 
6       sort  $T'$  by  $aRT$  descendent
7        $t' = T'[0]$ 
8        $rt = t'.aRT$ 
9        $eT = (t'.aestT - t'.aRT)/t'.aRT$ 
10      sort  $T'$  by  $aRC$  descendent
11       $t' = T'[0]$ 
12       $c = t'.aRC$ 
13       $eC = (t'.aestC - t'.aRC)/t'.aRC$ 
14      sort  $T'$  by  $aRE$  descendent
15       $t' = T'[0]$ 
16       $e = t'.aRE$ 
17       $eE = (t'.aestE - t'.aRE)/t'.aRE$ 
   //benefit of adaptation computation phase
18       $fs^1(rt, c, e)$ 
19       $BoA = fs^1.BoA$ 
20      if  $BoA \geq \text{medium}$ 
21           $fs^2(eT, eC, eE)$ 
22           $w_1 = fs^2.w_1$ 
23           $w_2 = fs^2.w_2$ 
24           $w_3 = fs^2.w_3$ 
25           $\text{AdjustWeights}(w_1, w_2, w_3)$ 
   //score computation and service ranking phase
26      for  $i = 0$  to ( $S_j.length - 1$ )
27           $S_j[i].W = w_1 S_j[i].estT + w_2 S_j[i].estC + w_3 S_j[i].estE$ 
28      sort  $S_j$  by  $W$  descendent
29      return  $S_j[0]$ 

ObtainAncestors( $t, T, T'$ )
1   for  $j = 0$  to ( $T.length - 1$ )
2       if  $T[j]$  is ancestor of  $t$ 
3           insert  $T[j]$  into  $T'$ 
4   return  $T'$ 

AdjustWeights( $w_1, w_2, w_3$ )
1    $w_t = w_1 + w_2 + w_3$ 
2    $w_1 = w_1/w_t$ 
3    $w_2 = w_2/w_t$ 
4    $w_3 = w_3/w_t$ 

```

Figure 4.2. QoS optimization heuristic.

After invoking the operation of the selected service, the obtained QoS values for service and task are stored in the historical database.

Accumulated QoS per task are calculated using the formulas presented in equations 3.2 to 3.7, 4.2 and 4.3.

4.4. Implementation

The composition framework used in this solution corresponds to the framework described in section 3.4, with modifications and extensions performed to some of its components, which enable the use of a new QoS parameter and the fuzzy support systems. The main changes are listed as follows:

- *Service binder.* The ranking process performed by the service binder uses the weights of three parameters instead of two, considering response time, cost and energy consumption. When the selected/invoked service has finished its execution, information about its QoS parameters is obtained and stored in the historical database.
- *Service selector.* After the functional service selection, estimated values of the different QoS parameters (response time, cost, energy consumption) are obtained by interacting with the predictor.
- *Adaptation manager.* The two fuzzy support systems (mentioned in section 4.3.2) were developed as part of the adaptation manager, distributed along the analyzer and the planner components. The java API used to implement these mechanisms is the jFuzzyLogic_v2.1a.jar [157], which is an open source package that implements a fuzzy control language. It allows the definition of fuzzy variables (input/output), fuzzy rules, and the use of different membership functions in order to fuzzify/defuzzify the variables.

4.5. Evaluation

In order to assess the effectiveness of the proposed optimization approach, two sets of experiments were performed, involving the test case presented in section 3.5.1 and the experimental environments described in sections 3.5.3.1 and 3.5.4.1. Experiments were carried out to address the following questions:

- How does the evaluation of the benefit of adaptation influence the adaptation process?
- Is there any improvement in the global QoS when using variable weights during service selection as part of a self-optimization mechanism?

The work performed to provide an answer to these questions is listed as follows:

- Assessment of the behaviour of adaptive composite services when evaluating the benefit of adaptation.
- Assessment and comparison of the behaviour of the proposed optimization model vs. a baseline approach that does not use optimization.
- Assessment of the behaviour of the optimization approach when executing component services in local and remote environments.

4.5.1. Service Selection Based on Fixed Weights

A service selection mechanism based on fixed weights was implemented to be compared with the proposed optimization approach. It extends the mechanism described in section 3.5.2 and follows similar steps. This approach uses Eq. 4.5 (presented in section 4.3.1) to obtain the services' score, where the values for w_1 , w_2 and w_3 are set equally to 0.333, and services are ranked looking for the one with the highest score. The change in the ranking criteria is due to the use of Eq. 4.4 for the normalization phase, which replaces the use of natural logarithms.

4.5.2. Dynamic QoS Parameters

To add dynamicity to the test environments, values of the QoS properties must change over time, or between services' executions. This helps to obtain sensible results and also avoid the invocation of only one service per each of the tasks in the composition. Two java applications have been designed and implemented with the aim of inserting such variations.

4.5.2.1. Cost

To turn the cost of the different services into dynamic QoS values, a model which affects cost based on demand was implemented. It is assumed that higher the cost, lower the demand. Demand is the number of times the service is invoked over a period of time. The algorithm that represents the cost model is shown in Figure 4.3. The notation used is as follows. Let,

- $S = \{s_1, s_2, \dots, s_m\}$ be the set of m services.
- i be the service number, where $s_i \in S$.
- $nInv$ be the number of times s_i has been invoked during a period of time.
- c be the value of cost for a service.
- max, min be default values set as the maximum and minimum number of service invocations.

The number of times a service has been invoked over a period of N minutes is evaluated continuously. Based on this information, and the values specified as the maximum and minimum number of invocations, it is possible to establish a new cost based on the demand. If the $nInv$ is equal or higher than max (step 1), the cost of s_i is increased (step 2). On the other hand, when $nInv$ is smaller than min (step 3), the cost of s_i is decreased (step 4). Finally, the algorithm returns the new value of cost for s_i (step 5). After each execution of the algorithm, the new cost is updated in the WSDL² file of the service.

```

EvaluateCost ( $s_i, nInv, max, min$ )
1   if  $nInv \geq max$ 
2        $s_i.c = s_i.c * 1.1$ 
3   else if  $nInv < min$ 
4        $s_i.c = s_i.c * 0.7$ 
5   return  $s_i.c$ 

```

Figure 4.3. Cost evaluation algorithm.

Additive increase with multiplicative decrease was used as the method to specify the changes in the values of cost. Increase rate was set to a 10%, while decrease rate to 30%.

² Extended version of the WSDL file.

4.5.2.2. Energy Consumption

Because of the importance of energy efficiency when managing computing infrastructure and services, the third QoS parameter included in this work is energy consumption, which represents the amount of watts-second (Ws) consumed by a server.

Using the linear model proposed in [153], which is based on the percentage of CPU usage, it is possible to determine an approximate value to the server energy consumption.

$$P(u) = P_{max} \cdot k + (1 - k) \cdot P_{max} \cdot U \quad (4.7)$$

Where:

$P(u)$ is the power consumed in an instance of time,

P_{max} is the power consumed when the server is fully utilized,

U is the utilization level,

k is the fraction of power consumed by the idle server.

Total energy consumption can be obtained using the following formula:

$$E = \int_t P(u(t)) \quad (4.8)$$

Where:

E is the total energy consumption,

t is the period of time.

Each of the servers where the Web services are executed, is assumed to have different hardware and software configurations (see Table 4.3). Servers and their characteristics were selected from the Energy Star report [158].

Table 4.3. Power consumption description per node.

Server	Hardware	Operative System	Idle (W)	Load (W)
Node 1	Acer Incorporated Gateway GT310 F1	Windows Server 2008 R2 64bit	50.75	129.5
Node 2	Hitachi - HA8000/SS10	Windows Server 2008 R2	45.27	81.97
Node 3	IBM - System X3650 M3	Red Hat Enterprise Linux 5 Update 4 x64 Edition	210.85	388.3

Servers' utilization is considered to be variable over time. The power consumed by a server is obtained periodically and exposed on the WSDL³ files of the corresponding services; it is computed using the data presented in Table 4.3. The energy consumed by a server at the moment the Web service is running, is calculated using the response time of the service.

4.5.3. QoS Parameters Configuration

The initial values of QoS parameters for the candidate services used in the experiments are established based on the node where the service is running and the corresponding set (as defined in chapter 3). The main difference between the previous configuration and the one used in this experiment is the definition of server's power consumption. Values for the initial setup of power consumption were obtained assuming that the utilization of the servers was 50% at the time when the first Web service was executed. This information is shown in Table 4.4.

Table 4.4. QoS parameters configuration.

Experiment 1	Experiment 2	Set	Time delays (ms)	Cost	Power consumption (W)
Node 1	Node 2	S1	0	120	90
		S2	350	80	
		S3	200	100	
Node 2	Node 3	S1	0	150	63
		S2	350	100	
		S3	200	120	
Node 3	Node 4	S1	0	100	299
		S2	350	60	
		S3	200	80	

Similar to the experiments executed in chapter 3, per each of the tasks in the process, there are 9 candidate services that fulfil the required functionality and offer different QoS, giving a total of 36 services, distributed in 9 sets among the servers (nodes). These services were previously registered into the service registry, and executed several times to populate the historical database and enable the estimation of their QoS attributes.

³ Extended version of the WSDL file.

The amount of historical information available previous to the execution of the experiments, corresponds to 1,000 executions.

4.5.4. First Stage of Evaluation

The first stage of evaluation was performed on the experimental environment described in section 3.5.3.1 (local area network). The travel planning service was executed 50 times to analyze the behaviour of the optimization approach and evaluate its overall benefit. The benefit of adaptation is evaluated in order to determine whether adaptation is needed, or not. To get a clear understanding on how the evaluation of the benefit of adaptation and the use of variable weights influence the results of service selection, the test case has also been executed using the service selection mechanism based on fixed weights described in section 4.5.1.

When using the proposed approach, executions were carried out applying dynamic QoS (based on the dynamic QoS models previously described). The evaluation of cost and power was performed every 3 minutes. Due to the randomness inserted in the QoS parameters, the experiment (set of 50 runs) was performed 5 times, to evaluate the consistency of the results based on statistical analysis.

4.5.4.1. Experimental Results

Results show that the proposed optimization approach improves the global QoS values (response time, cost and energy consumption) of the composition. The following plots show a comparison between the proposed approach and the fixed weights approach for each of the QoS parameters. When using the proposed approach, QoS values are dynamic, services' cost and servers' power consumption change over time, based on the models described in section 4.5.2. On the other hand, when using fixed weights, values for cost and energy consumption remain constant. For both cases, response time is dynamic.

When analyzing the collected results, it can be noted that the proposed approach provides smaller response times as compared with the fixed weights mechanism, as illustrated in Figure 4.4. This is due to the evaluation of the QoS values before a new service is selected. The system

aims to maintain or if possible, improve the global QoS of the composition. Measured response time values of the proposed approach provide a mean reduction of 3% and a highest reduction of 20.5%, with a mean standard deviation of 5.2%, and a 95% confidence interval between 13,683.3ms and 13,888.1ms. Standard deviation values presented per each of the 50 executions among the 5 runs are represented with error bars in Figure 4.5.

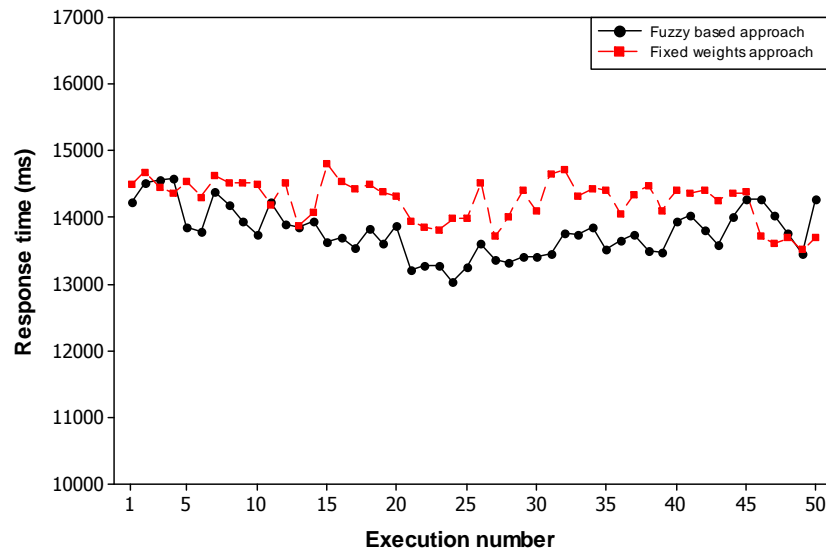


Figure 4.4. Response time comparison between fuzzy based and fixed weights approaches.

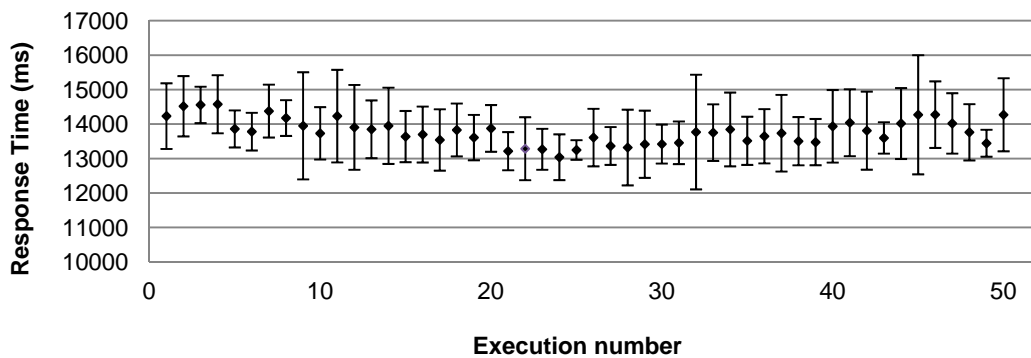


Figure 4.5. Response time evaluation - differences between executions.

The obtained mean cost values are shown in Figure 4.6. In comparison with the fixed weights approach, the use of the proposed fuzzy based system provides a mean reduction of 4.5% and a highest reduction of 33.4%, with a mean standard deviation of 6.9%, and a 95% confidence interval between 383.17 and 390.04. Differences presented among the different executions are illustrated in Figure 4.7.

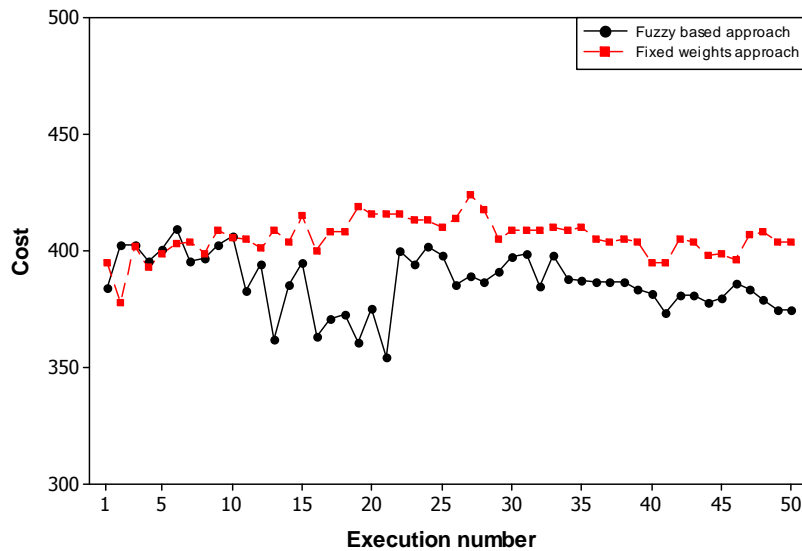


Figure 4.6. Cost comparison between fuzzy based and fixed weights approaches.

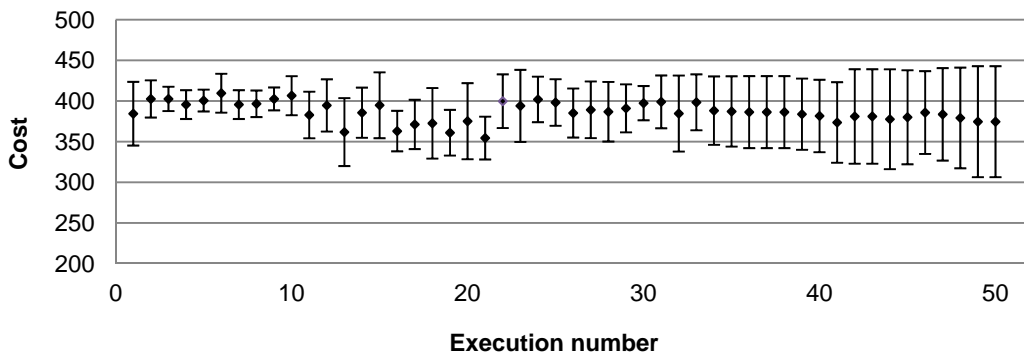


Figure 4.7. Cost evaluation - differences between executions.

Results also indicate that there is a significant reduction in the values of energy consumption, as illustrated in Figure 4.8, providing a mean reduction of 31.2%, with a standard deviation of 37.5%, and a 95% confidence interval between 180.78Ws and 198.17Ws. Figure 4.9 summarizes the standard deviation values of the 5 different sets of 50 executions using error bars. One important factor to consider is that energy consumption is not only based in power consumption, but also in time. A small response time value may generate a small energy consumption value.

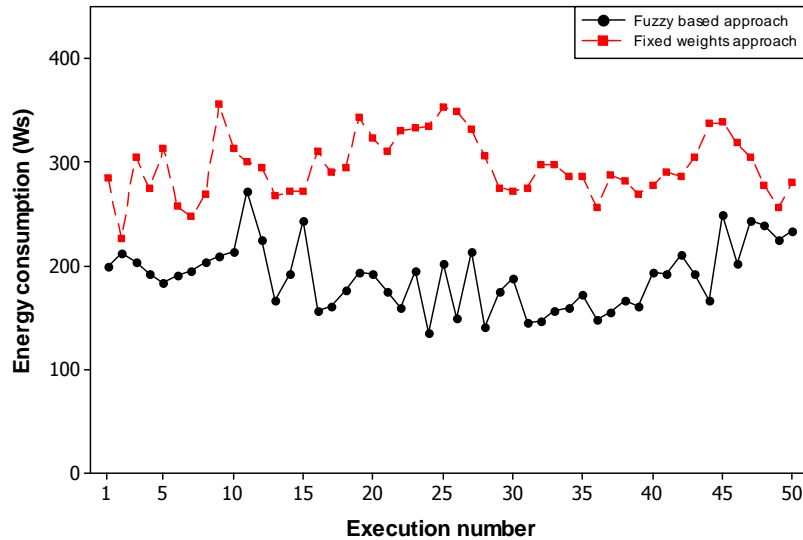


Figure 4.8. Energy consumption comparison between fuzzy based and fixed weights approaches.

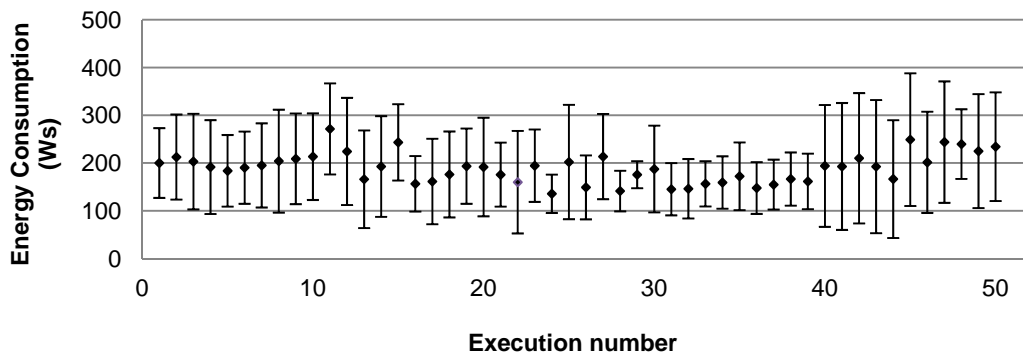


Figure 4.9. Energy consumption evaluation - differences between executions.

Values corresponding to the proposed approach with dynamic QoS present the highest standard deviations for cost and energy consumption. This behaviour is due to the inserted dynamicity. Even though the highest cost is found in the proposed approach, when it comes to average values, it is still lower than the fixed weights results.

The values of benefit of adaptation (BoA) collected per each task, during one set of 50 executions of the process are illustrated in Figure 4.10. These values were obtained using the proposed optimization model with dynamic QoS. For the first task of the process (card validation), as there is no QoS information from previous tasks, the BoA is equal to 0, setting the weights for service selection equal to 0.33. Hotel reservation and flight reservation are executed in parallel after card validation, reason why their BoA values are the same.

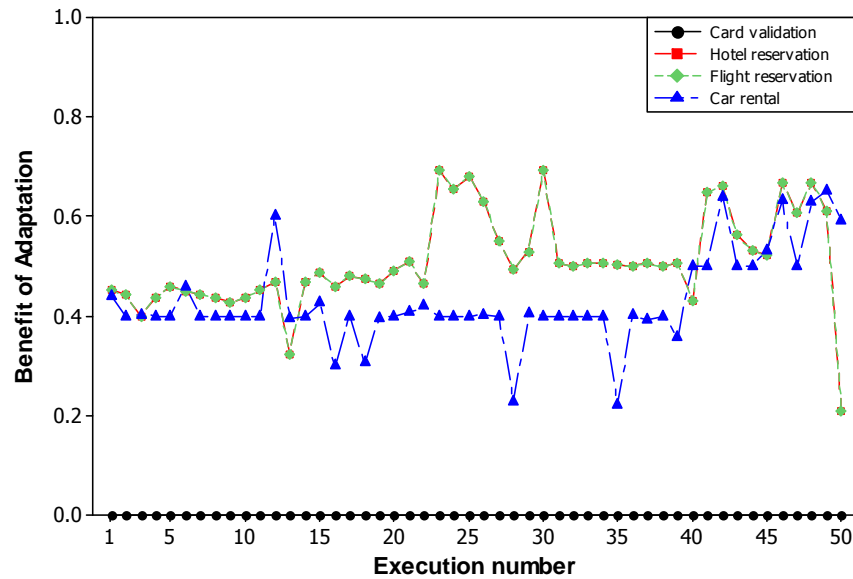


Figure 4.10. Benefit of adaptation per each task in the travel planning process.

Adaptation is performed per task when BoA is larger than 0.4, which is the highest value for the medium term defined in the fuzzy system. It was noted that in most of the cases where BoA was higher than 0.45 for hotel reservation/flight reservation tasks, BoA values were lower than medium for the last task of the process, therefore, adaptation was not needed.

4.5.5. Second Stage of Evaluation

The second stage of evaluation was performed on the experimental environment described in section 3.5.4.1 (wide area network). The travel planning service was executed 50 times to analyze the behaviour of the optimization approach and evaluate its overall benefit. The test case has also been executed using the service selection mechanism based on fixed weights described in section 4.5.1.

When using the proposed approach, executions were carried out applying dynamic QoS (based on the dynamic QoS models described in section 4.5.2). Cost and power evaluation was performed every 3 minutes. Due to the randomness inserted in the QoS parameters, the experiment (set of 50 runs) was performed 3 times to evaluate the consistency of the results based on statistical analysis.

4.5.5.1. Experimental Results

When analyzing the collected results of the second set of experiments, it was noted that not all the values of the QoS parameters were enhanced. The proposed optimization approach improves two of the QoS values of the composition, but in order to provide such improvements, there is an increment in the third parameter, as shown in the following table.

Table 4.5. Results summary.

Execution	Response time (ms)		Cost		Energy consumption (Ws)	
	Fuzzy	Fixed weights	Fuzzy	Fixed weights	Fuzzy	Fixed weights
1	3249.44	3665.58	419.46	451.2	118.56	102.89
2	3136.76	3695.62	559.18	471.4	53.19	69.31
3	3441.16	3753.26	390.82	375.8	138.49	153.14

The plot illustrated in Figure 4.11 depicts the behaviour of the composite service in terms of response time, where can be noted that the proposed approach shows smaller values when comparing to the fixed weights approach. The average values of the executions correspond to 3,275.78ms and 3,704.82ms, for the proposed approach (fuzzy) and the fixed weights approach, respectively. It was obtained a significant mean reduction of 13%, with a mean standard deviation of 18.63%, and a 95% confidence interval for the mean between 3,172.8ms and 3,378.8ms. Differences among the 3 sets of executions are illustrated in Figure 4.12.

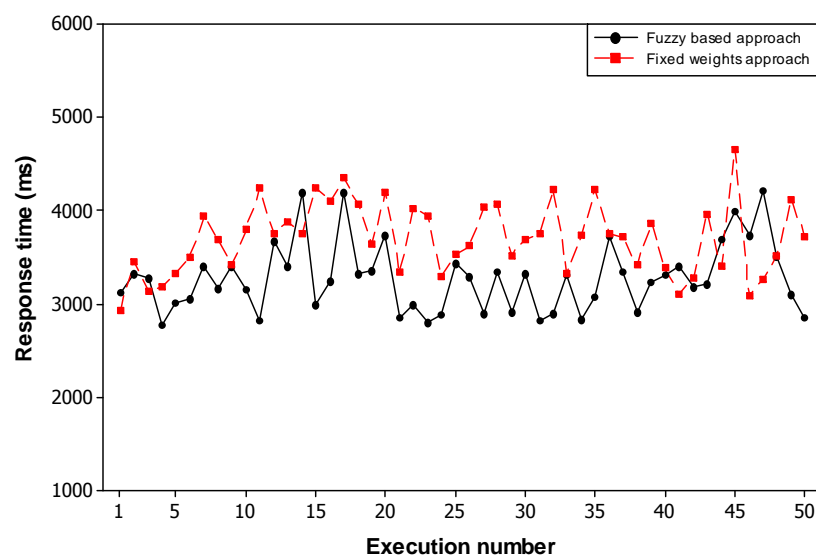


Figure 4.11. Response time comparison between fuzzy based and fixed weights approaches.

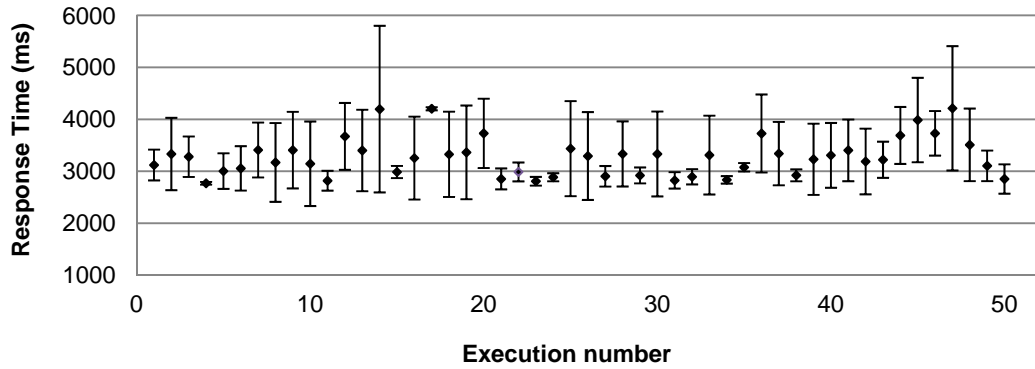


Figure 4.12. Response time evaluation - differences between executions.

In terms of cost, the proposed approach shows higher values, with a registered average of 456.48, which reflects an increment of 5% when comparing to the fixed weights approach average of 432.8. This was obtained with a mean standard deviation of 12.5%, and a 95% confidence interval between 445.86 and 467.12. The following plots depict the behaviour of the composition’s cost. Figure 4.13 illustrates a comparison between the two approaches; while Figure 4.14 summarizes the standard deviations values of the 3 different sets of 50 executions using error bars.

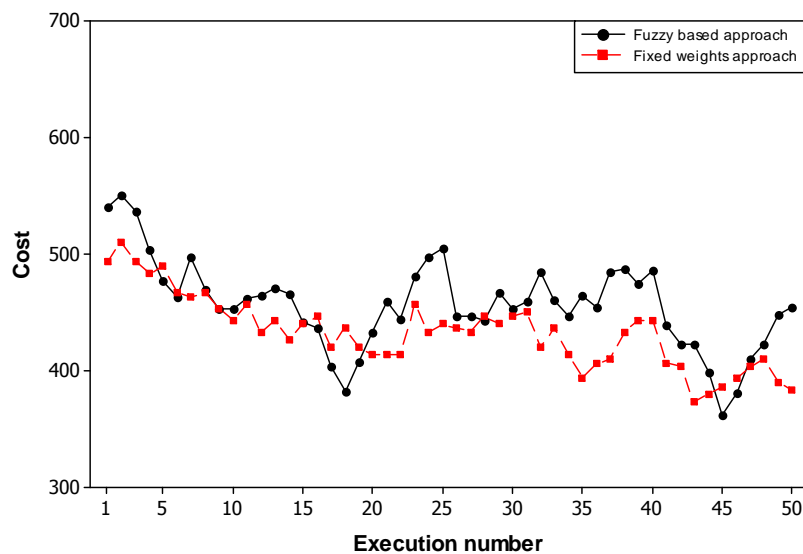


Figure 4.13. Cost comparison between fuzzy based and fixed weights approaches.

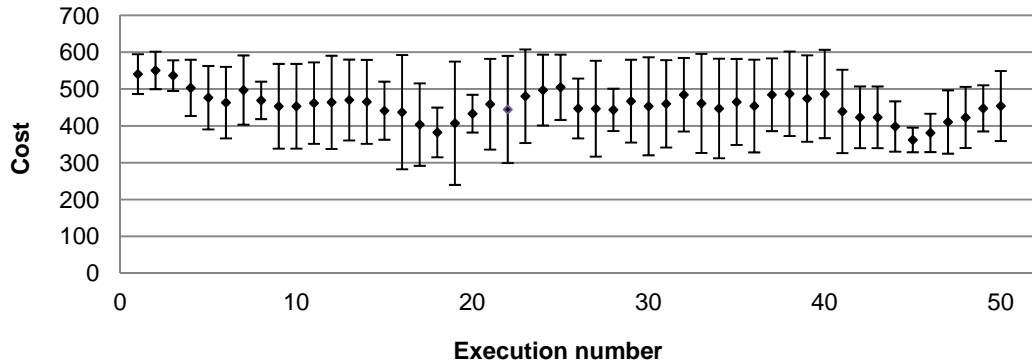


Figure 4.14. Cost evaluation - differences between executions.

Results also indicate that there is a significant reduction regarding energy consumption, as illustrated in Figure 4.15. The collected values provide an average of 103.41Ws, with a mean reduction of 9.21%, and a 95% confidence interval between 93.23Ws and 113.61 Ws.

Differences in terms of energy consumption among the 3 sets of executions are illustrated in Figure 4.16. It can be noted that in some executions, the average of energy consumed by a composite service can exhibit a high variability. This is caused by the use of the dynamic QoS models described section 4.5.2.

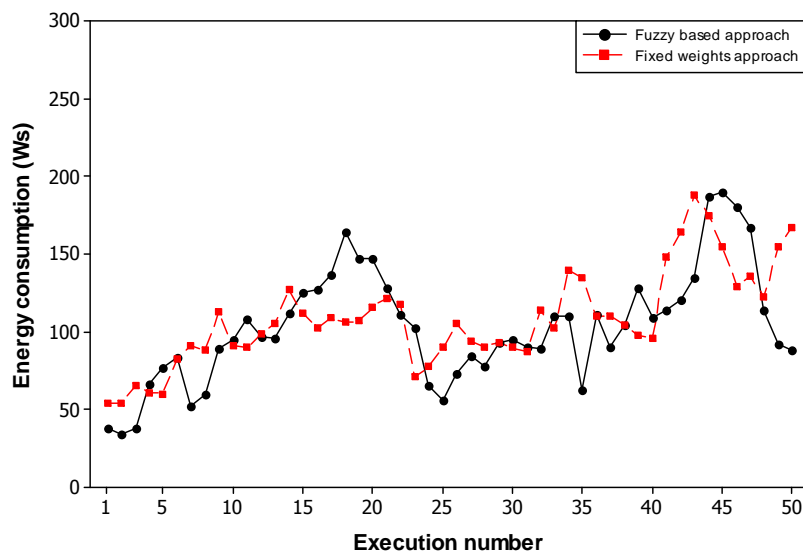


Figure 4.15. Energy consumption comparison between fuzzy based and fixed weights approaches.

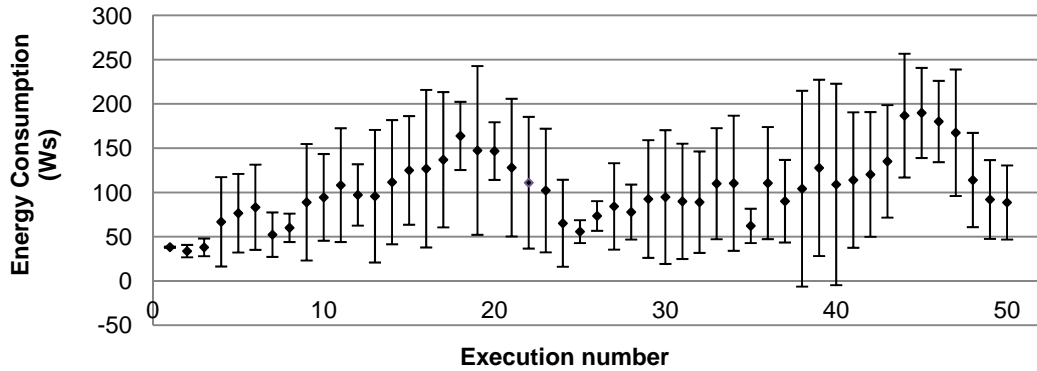


Figure 4.16. Energy consumption evaluation - differences between executions.

The plot illustrated in Figure 4.17 depicts the values of benefit of adaptation (BoA) per task, obtained from a set of 50 executions of the travel planning process, using the proposed optimization approach. Values for the card validation task are equal to 0, as this is the first activity in the process and there is no information to evaluate its QoS values before execution.

It was observed a similar behaviour as compared to the experiment performed in the local environment, where most of the values of BoA collected for hotel and flight reservation tasks are higher than the BoA values of car rental. Hotel and flight reservation are executed in parallel after card validation, reason why they have the same BoA values.

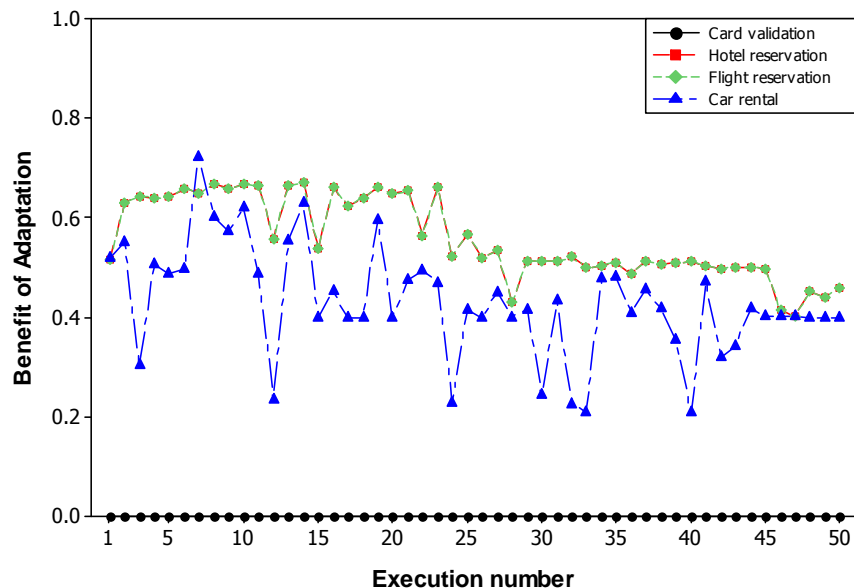


Figure 4.17. Benefit of adaptation per each task in the travel planning process.

Values of BoA for hotel and flight reservation are higher as compared to the results obtained in a local area network, causing the process to adapt

in most of its executions. Even though values of BoA obtained for the last activity are higher as compared with those obtained in a local environment, they are still smaller to those obtained in the previous activity, and several executions did not require to perform adaptation actions.

4.5.6. Discussion

The proposed optimization model performs service selection based on the analysis of historical and real QoS data, gathered at different stages during the execution of composite services. The use of fuzzy inference systems enables the evaluation of the measured QoS values, helps deciding whether adaptation is needed or not, and how to perform service selection. Fuzzy logic has demonstrated to be a useful tool during the evaluation process of the QoS attributes. By obtaining and analyzing the benefit of adaptation, adaptation is not carried out each time a QoS value changes. It was noted that in most of the cases, when adaptation is triggered at certain stage of the composition, the benefit of performing adaptation is a small value (with no need of adaptation) in the next task of the process.

The use of the optimization approach presented in this chapter has provided meaningful improvements in the global QoS of the test case scenario, with reductions up to 20.5% in response time, 33.4% in cost and 31.2% in energy consumption. It was observed that when using a WAN as part of the execution environment, in order to improve the overall QoS of the composition there is an increment in one of the three parameters considered. This is caused by the additional variations in response time inserted by the network.

When looking at performance, the use of the proposed model generates an average increment of 581ms in the invocation time per task (information obtained using a database with 10 candidate services and 100 records per service). Overheads increase following a quadratic model. This behaviour was determined after performing various sets of executions increasing the number of candidate services and analyzing the measured execution time (see Figure 4.18).

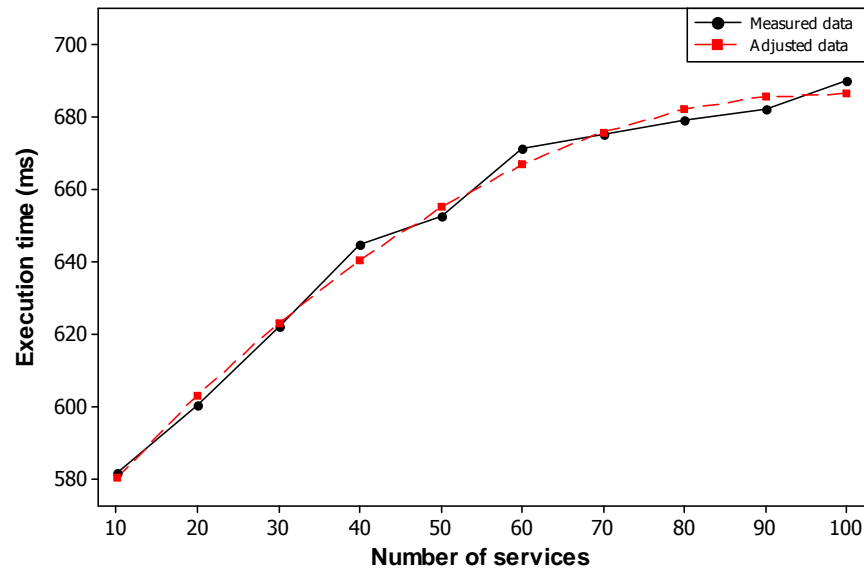


Figure 4.18. Execution time for different number of available candidate services.

Further assessment of the results presented in section 4.5 will be discussed in chapter 6, along with a comparison between the proposed optimization model and relevant related work.

4.6. Summary

This chapter has presented a QoS optimization model for service composition based on fuzzy logic. Motivation behind the development of the approach is provided, followed by a discussion on approaches that use decision support systems in service selection and composition.

The proposed solution is then described, including details about the QoS model and optimization model. Modifications applied to the implementation of the composition framework described in chapter 3 are then given. Finally, evaluation details are provided, covering the description of the experimental setup, dynamic QoS parameters and results.

The main difference between the related work found in the literature and the approach described in this chapter, is the purpose of the use of fuzzy logic. In the proposed approach, fuzzy logic is used as a tool to evaluate the measured QoS values in order to determine the benefit of performing adaptation.

The environment presented in this chapter enables the execution of composite services with QoS aware and adaptive capabilities, and evaluates the need of performing adaptation using fuzzy logic. However, it triggers the QoS evaluation and adaptation strategies from a reactive perspective. The following chapter will describe a proactive adaptation mechanism for service composition. This mechanism is built as an extension to the QoS optimization model described in this chapter.

Chapter 5

A Proactive Adaptation Mechanism for Service Composition

This chapter describes a proactive adaptation mechanism for service composition based on fuzzy logic. Motivation towards the development of the approach is discussed. Then, a review on work related to the provision of proactive adaptation in service composition is presented. Following this, the proposed solution is described, providing details related to the service composition framework, QoS model and optimization model, along with information regarding implementation. Finally, the experiments performed to evaluate the proposed approach are discussed in detail.

5.1. Motivation

As discussed in previous chapters, there are different situations that can trigger adaptation in service composition (e.g. failures, changes in QoS levels, new services, etc.). Approaches focussed on ensuring/maintaining the functional and quality levels of composite services, can be classified based on the time when adaptation takes place into the categories: reactive and proactive. The former corresponds to adaptation actions performed in response to an incident, while the later is related to actions taken in advance, before an incident impacts the system [100].

When adaptation in service composition is performed from a reactive perspective, as it works after unwanted situations already occur, it may cause increments in the execution time of the composition, leading to unwanted consequences like financial loss and business dissatisfaction [99]. In some situations, the event that trigger the need for a change may arrive when adaptation is not possible any longer [98]. The aim of proactive adaptation approaches is to mitigate some of these negative aspects, by detecting the need for a change before reaching a point where a problem may occur. Some of the benefits offered by proactive adaptation include [98]:

- Variations in the QoS levels of the composition can be identified and targeted before having any consequences.
- Adaptation actions do not affect the execution of the composition.
- The need for adaptation is identified in advance, providing the service with enough time to adapt.

The scope of the work described in this chapter is mainly focussed on the development of a proactive adaptation mechanism for service composition based on fuzzy logic. This mechanism is proactive in the sense that it identifies the need for adaptation (QoS degradation/improvement) before the composite service itself, addressing optimization at engine level. The engine is constantly monitoring and analyzing the services' behaviour at runtime and triggers adaptation actions when needed. The approaches presented in previous chapters detect the need for adaptation within the composite service.

5.2. Proactive Adaptation in Service Composition

Some approaches that support reactive adaptation implement self-* properties. Self-healing mechanisms aim to prevent composite services from failing, from functional and non-functional perspectives. Projects like those presented in [59], [90], [107], [108], [128], [129] and [159] apply self-healing approaches, where new services are selected and invoked after a functional failure or a QoS constraint violation. Self-optimization mechanisms are closely related to the selection of services at runtime, in order to maintain the expected QoS of the entire composition. Examples of works belonging to this category are described in [86], [87], [95] and [130].

Approaches that support proactive adaptation in service-based applications are presented in [98], [99], [100], [160], [161], [162] and [163], and summarized as follows:

- The work presented in [99] introduces a proactive adaptation approach that enables service replacement ($1 - 1$, $1 - n$, $n - 1$, $n - m$) when it detects situations that may cause the composition to stop its execution (unavailable or malfunctioning services); or that allow the composition to continue its execution, but not in its best

way. Also, it considers the emergence of better services and new requirements. The approach uses a composition template as start point and selects a set of candidate services to be used in the composition and their replacements.

- The approach introduced in [162] combines runtime information with design-time specifications (of each component service within a composition), in order to construct a k-step model of the current service states. The resulted model can be used to be compared with the desired behaviour of the composition.
- The framework described in [161] aims to minimize Service Level Agreement (SLA) violations in service compositions. It uses predictions of SLA violations generated with regressions of monitored and estimated data. These predictions are evaluated at defined checkpoints.
- The framework presented in [98] uses online testing to trigger proactive adaptation in service-based applications. Test objects can be single or composite services. While performing online testing, if an online test fails or deviates from its expected behaviour, the framework will trigger adaptation to avoid undesirable consequences. One of the application scenarios for this approach is composite services.
- The work described in [160] proposes a self-adaptive mechanism based on the use of test cases to obtain possible mismatches between requested and provided services. When the diagnosis mechanism reveals mismatches, it triggers adaptation strategies that update the structure and behaviour of the client application, solving the identified problems. Even though this approach is not mainly focused in service composition, it presents a proactive mechanism that works in service-based applications.
- The approach introduced in [163] combines monitoring, online testing and quality prediction to enable proactive adaptation in service-based applications. When a service is likely to be used with a high frequency, it is selected to be tested. The use of pre-defined test cases (concrete data inputs) enables the system to collect information

about the behaviour of the services and complement the data gathered during monitoring.

- The work described in [100] discusses two main directions that can be followed in order to perform proactive adaptation in service oriented systems. The first direction is to improve the failure predictions techniques. Some prediction techniques identified by the authors include data mining, online testing, runtime verification, statics analysis and simulation. The second direction is to dynamically estimate the accuracy of the predicted failures during runtime.

The proactive approaches found in the literature are mainly focused on adaptation targeting failures (e.g. unavailable service, QoS violation, performance decrease). They do not consider the possibility of improving the QoS levels of the service-based systems. In terms of QoS parameters, most of these approaches are centred on response time and cost. The work in this chapter presents a proactive adaptation mechanism for service composition that aims to overcome these limitations by targeting failure prevention and QoS improvement, considering multiple QoS parameters, which include: response time, cost, energy consumption and availability. Further comparison between related approaches and the mechanism proposed in this chapter will be discussed in chapter 6.

5.3. Proposed Solution

The environment presented in chapters 3 and 4 enables the execution of composite services with QoS aware and adaptive capabilities. However, adaptation is performed from a reactive perspective. In order to enable proactive adaptation, modifications to the interaction among components within the composition framework were performed, along with extensions to the QoS model and optimization model.

5.3.1. System Model

An overview of the system model considered in the work described in this chapter is illustrated in Figure 5.1, which shows its core components: composition engine, adaptation manager, service binder, service selector,

predictor and the sensors; and their interactions. This model was implemented with the aim of enabling the execution of QoS aware service compositions in an environment with proactive capabilities. It is built as an extension of the composition framework described in chapter 3.

The composition engine is the software platform responsible for executing the composite services (processes' definitions) and hosting the components in charge of the adaptation process. Composite services are considered to consist of a series of abstract tasks that will be linked to executable services at runtime. In this version of the composition framework, the adaptation manager works semi-independent of the rest of the components, and is constantly monitoring and analyzing not only information collected by the sensors, but also historical data. The use of historical data helps the understanding of the behaviour of the service and enables the detection of any possible deviation in the values of the QoS parameters.

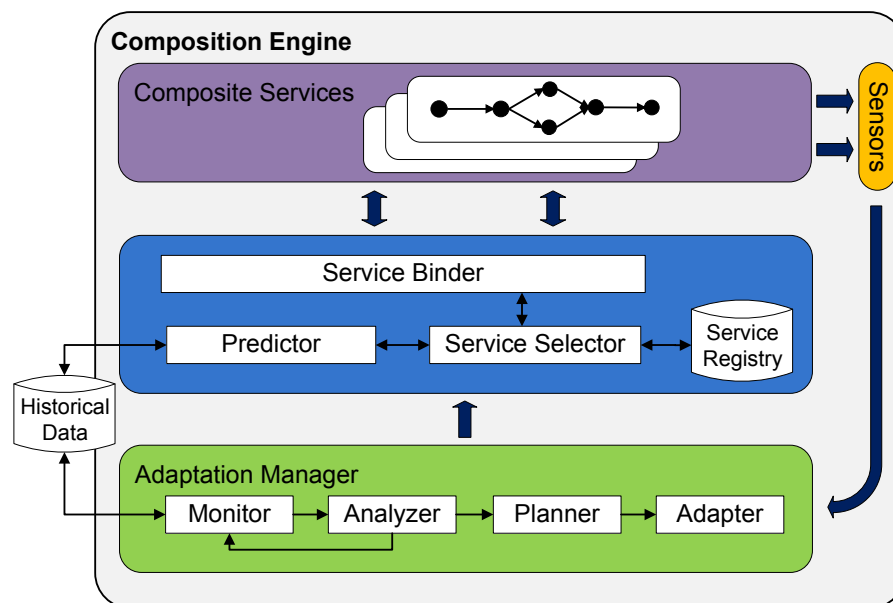


Figure 5.1. System model.

During the execution of a composite service, sensors collect fresh data, looking at activity and service levels, and send this information to the monitor. The monitor queries the historical database to obtain information about previous executions and states of the current service, then, sends this information to the analyzer, which evaluates both, fresh and historical data, in order to determine the need of adaptation. If adaptation is needed, the analyzer sends a request of adaptation to the planner, which obtains the adaptation values that will be sent to the adapter. This information is

forwarded to the service binder, in order to maintain/improve the QoS of the composition.

For each task in the composite service, the service binder invokes the service selector with the desired characteristics that the component service should provide. The service selector performs a search in the service registry based on the provided functional requirements. For each of the pre-selected services (candidates), the service selector invokes the predictor to obtain its estimated QoS. This information is sent to the service binder, which compares the candidates and selects the service that suits the request. If the need of a change was identified by the adaptation manager, the binder uses the adaptation values to perform the ranking and selection tasks.

5.3.2. QoS Model and Service Selection Model

The QoS model and service selection model used in this approach, extend the models described in section 4.3.1. As a result, the quality parameters considered for each service are response time, cost, energy consumption and availability.

- *Response time (Rt)*. Time consumed between the invocation and completion of the service operation [59].
- *Cost (C)*. Fee charged to the consumer when invoking a service [86].
- *Energy consumption (Ec)*. Amount of power consumed by a server over a period of time [153].
- *Availability (Av)*. Probability that the service is up and ready for immediate consumption [4].

The last parameter that has been selected as part of this research is availability. By knowing the availability values of the different services, it is possible to select a subset of components that will provide a composition with high probabilities to be fulfilled. Work that considers availability has been presented in [62], [68] and [107].

Assuming that a service (s) only contains one operation, its QoS (Q) can be defined using Eq. 5.1.

$$Q(s) = (Rt(s), C(s), Ec(s), Av(s)) \quad (5.1)$$

Computation of availability is based on three situations within the composite service structure. When computing the availability (Av) of a single service invocation, the availability value of the activity that performs the invocation corresponds to the Av of the invoked service, as shown in Eq. 5.2.

$$Av(t_i) = Av(s) \quad (5.2)$$

For activities in sequential and concurrent/parallel structures, the value of availability (Av) is multiplied for the activities with service invocations contained in the structure, as defined in Eq. 5.3.

$$Av(P) = \prod_{i=1}^n Av(t_i) \quad (5.3)$$

In this set of equations, the value of t_i corresponds to an activity (task) with a service invocation within the composite service P .

Service selection is performed according to the model described in section 4.3.1. After filtering services (based on their functionality), the obtained subset is ranked according to the relationship among their estimated QoS values. Estimations are obtained from historical data using the average of the last N executions, after filtering values considered as outliers. Response time, cost and energy consumption are negative parameters (lower the value, higher the quality); while availability is a positive parameter (higher the value, higher the quality). As the service rank process is performed using normalized values, and the nature of availability is opposite to the other parameters, a different formula was required to normalize its values, described in Eq. 5.4.

$$V_i = \frac{q_i - \min_i}{\max_i - \min_i} \quad (5.4)$$

Where:

\max_i corresponds to the maximum value of the evaluated QoS parameter,

\min_i corresponds to the minimum value of the evaluated QoS parameter,

q_i corresponds to the estimated value for the next execution.

When $\max_i = \min_i$, then $V_i = 1$.

After normalizing the values, the overall quality score (W) for each service is computed using Eq. 5.5.

$$W_i = w_1 \text{est}T_i + w_2 \text{est}C_i + w_3 \text{est}E_i + w_4 \text{est}A_i \quad (5.5)$$

Where:

$\text{est}T_i$ is the service estimated response time,

$\text{est}C_i$ is the service estimated cost,

$\text{est}E_i$ is the service estimated energy consumption,

$\text{est}A_i$ is the service estimated availability,

w_1, w_2, w_3 and w_4 correspond to assigned weights, where

$0 \leq w_1, w_2, w_3, w_4 \leq 1$ and $w_1 + w_2 + w_3 + w_4 = 1$.

Values for w_1, w_2, w_3 and w_4 are provided by the QoS evaluation heuristic described in the following section. The set of candidate services is ranked based on the values of W_i , and the service with the highest value is selected.

5.3.3. QoS Optimization Model

The proposed optimization model works as part of a proactive adaptation mechanism. It combines the analysis of historical and fresh data. Similar to the models presented in previous chapters, QoS information of the different services and states of the composition is collected from service, task and process perspectives, where service corresponds to concrete Web services; task to elements within the composite service that invoke services; and process to the entire composition (service workflow). Based on this information, it is possible to take decisions about future actions.

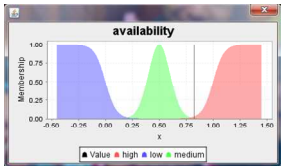
The QoS parameters are obtained when the service invocation is performed. Response time is measured during the service's execution; the values of cost and server's power consumption are retrieved from the service's WSDL¹ file; while the value of availability is obtained based on historical data. According to the structures of the composite service, the QoS values of each task are computed and stored in the historical QoS database,

¹ The WSDL standard was extended to include the service's QoS information.

considering both individual values and accumulated. These values are used in order to obtain the global QoS of the composite service.

The service selection model previously described uses as weights for the ranking process the results of the optimization model evaluation. This model is based on extended versions of the two fuzzy support systems described in section 4.3.2.1. The optimization mechanism identifies when the QoS of the composition is degrading. It also considers situations where a number of the accumulated QoS values of the previous activity in the process are better than expected, which provides the possibility of improving other QoS parameters. Both fuzzy support systems were extended by adding information regarding availability as part of their variables. In the first fuzzy support system, in order to evaluate the benefit of adaptation, *availability* was added as an input parameter, using the same linguistic terms defined for response time, cost and energy consumption (*low*, *medium* and *high*), as shown in Table 5.1.

Table 5.1. Fuzzy variable definition - availability.

Variable	Terms	Type	Functions
Availability	Low = sigm (-50, min) Medium = gauss (avg, std) High = sigm (50, max)	Input	

Where:

std is the standard deviation (after filtering outliers),
avg is the average value between maximum and minimum.

The rules used to evaluate the benefit of adaptation (see section 4.3.2.1), were modified by adding the terms of the new input variable. The set of rules, used in the development of the proactive mechanism described in this chapter, is shown in appendix C.

A similar approach was taken with the second fuzzy support system. The error obtained between the estimated and measured value of availability was included as the fifth input variable, expressed with three terms: *low*, *medium* and *high*, falling in the interval [-1, +1]. The *availability weight* (new output variable), is expressed with five terms, *very low*, *low*, *medium*, *high* and *very high*, falling in the interval [0,1], and is defined using Gauss functions.

The heuristic presented in Figure 5.2 describes the QoS evaluation method applied during the optimization process, which involves the use of the extended versions of the fuzzy systems. The notation used is shown as follows. Let,

- $T = \{t_1, t_2, \dots, t_n\}$ be the set of n tasks in process P .
- j be the task number, where $t_j \in T$.
- $T' = \{t'_1, t'_2, \dots, t'_k\}$ be the set of k ancestors of t_j , where $T' \in T$. When $j = 1$, then $T' = \{\emptyset\}$.
- $aRT, aRC, aRE, aRA, aestT, aestC, aestE, aestA$ be the accumulated values corresponding to real response time, real cost, real energy consumption, real availability, estimated response time, estimated cost, estimated energy consumption and estimated availability for a task.
- eT, eC, eE, eA be the error values corresponding to response time, cost energy consumption and availability for a task (see Eq. 4.6).
- w_1, w_2, w_3, w_4 be weights used to obtain the score of a service (see Eq. 5.5).
- BoA be the value corresponding to the benefit of performing adaptation.
- fS^1, fS^2 be fuzzy systems.

Once the execution of P starts, the adaptation manager constantly evaluates its QoS, by looking at the behaviour of its tasks. The ancestors T' for t_j are obtained (step 3). w_1, w_2, w_3 and w_4 are set initially to 0.25, enabling a service ranking with no preference (step 4). This is used in case there are no meaningful differences between the QoS values of P before t_j . If t_j , is not the first task in P , this task has ancestors and QoS evaluation takes place (steps 6 to 30). Values within T' are sorted based on aRT (step 6). The task with the highest aRT value is selected, its accumulated real response time is retrieved, and the error between its estimated and real response time is obtained (steps 7 to 9). Values within T' are sorted based on aRC (step 10). The task with the highest aRC value is selected, its accumulated real cost is retrieved, and the error between its estimated and real cost is obtained (steps 11 to 13). Values within T' are sorted based on aRE (step 14). The task with the highest aRE value is selected, its accumulated real energy consumption is retrieved, and the error between its estimated and real energy consumption is obtained (steps 15 to 17). Values within T' are sorted based on aRA (step 18). The task with the lowest aRA

value is selected, its accumulated real availability is retrieved, and the error between its estimated and real availability is obtained (steps 19 to 21).

```

EvaluateQoS( $t_j, T$ )
1  let  $t'$  be a task
2  let  $T'$  be an empty list
3   $T' = \text{ObtainAncestors}(t_j, T, T')$ 
4   $w_1 = w_2 = w_3 = w_4 = 0.25$ 
   // weights selection phase
5  if  $T'.length \neq 0$ 
6      sort  $T'$  by  $aRT$  descendent
7       $t' = T'[0]$ 
8       $rt = t'.aRT$ 
9       $eT = (t'.aestT - t'.aRT)/t'.aRT$ 
10     sort  $T'$  by  $aRC$  descendent
11      $t' = T'[0]$ 
12      $c = t'.aRC$ 
13      $eC = (t'.aestC - t'.aRC)/t'.aRC$ 
14     sort  $T'$  by  $aRE$  descendent
15      $t' = T'[0]$ 
16      $e = t'.aRE$ 
17      $eE = (t'.aestE - t'.aRE)/t'.aRE$ 
18     sort  $T'$  by  $aRA$  descendent
19      $t' = T'[0]$ 
20      $av = t'.aRA$ 
21      $eA = (t'.aestA - t'.aRA)/t'.aRA$ 
   //benefit of adaptation computation phase
22      $fs^1(rt, c, e, av)$ 
23      $BoA = fs^1.BoA$ 
24     if  $BoA \geq \text{medium}$ 
25          $fs^2(eT, eC, eE, eA)$ 
26          $w_1 = fs^2.w_1$ 
27          $w_2 = fs^2.w_2$ 
28          $w_3 = fs^2.w_3$ 
29          $w_4 = fs^2.w_4$ 
30          $\text{AdjustWeights}(w_1, w_2, w_3, w_4)$ 
31 return  $w_1, w_2, w_3, w_4$ 

ObtainAncestors( $t, T, T'$ )
1  for  $j = 0$  to ( $T.length - 1$ )
2      if  $T[j]$  is ancestor of  $t$ 
3          insert  $T[j]$  into  $T'$ 
4  return  $T'$ 

AdjustWeights( $w_1, w_2, w_3, w_4$ )
1   $w_t = w_1 + w_2 + w_3 + w_4$ 
2   $w_1 = w_1/w_t$ 
3   $w_2 = w_2/w_t$ 
4   $w_3 = w_3/w_t$ 
5   $w_4 = w_4/w_t$ 

```

Figure 5.2. QoS evaluation heuristic.

The accumulated real values are set as inputs for f_s^1 (step 22). *BoA* is obtained and evaluated (steps 23 and 24); if it is medium or higher, then there is a need for adaptation. When adaptation is needed, the system determines the new weights to be used during the service selection process. This action is performed by f_s^2 (step 25). The values of w_1, w_2, w_3 and w_4 are retrieved and their values are adjusted, to fulfil the restriction $w_1 + w_2 + w_3 + w_4 = 1$ (steps 26 to 30). Finally, the heuristic returns the weight values w_1, w_2, w_3 and w_4 (step 31). These values are sent to the service binder to be used at the moment of selecting the next service. When adaptation is not needed, the service binder ranks the services using fixed weight values.

After invoking the operation of the selected service, the obtained QoS values for service and task are stored in the historical database. Accumulated QoS per task are calculated using the formulas presented in equations 3.2 to 3.7, 4.2, 4.3, 5.2 and 5.3, based on the structure of the process.

5.4. Implementation

The composition framework used to implement the proactive adaptation mechanism, described along this chapter, contains the same components described in section 3.4. However, interaction among them shows some differences, as depicted in Figure 5.3.

Interaction between the *Service Binder* and the *Adaptation Manager* does not occur each time the *Service Binder* is going to select a new service (as in the framework used in previous chapters). The *Adaptation Manager* monitors and analyzes the behaviour of the composite service at runtime (while the service and its components are being executed). It uses historical information combined with new information about the service execution. When it identifies the need for a change, sends the weights to be used during service ranking and selection to the *Service Binder*.

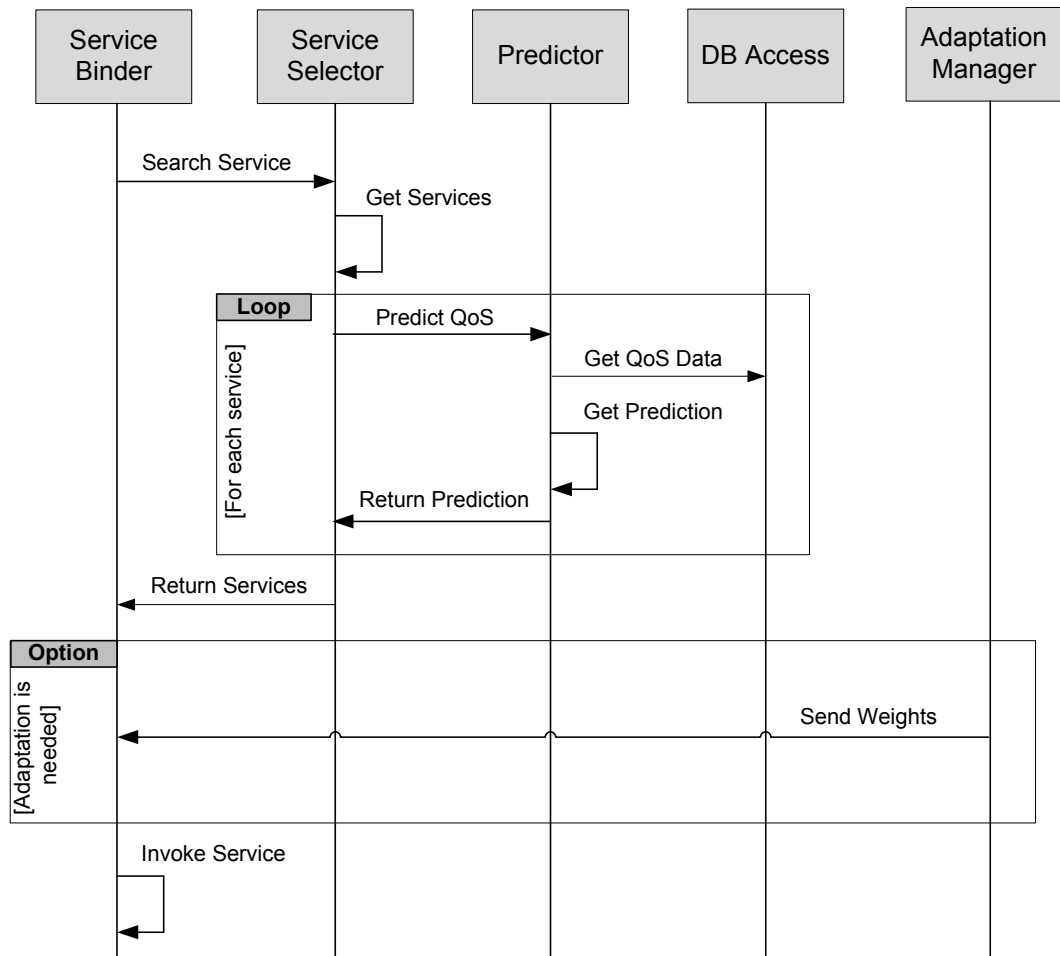


Figure 5.3. Components interaction.

The main changes performed to the components of the framework, in order to enable proactive adaptation and consider availability are listed as follows:

- *Adaptation manager.* Instead of being invoked from the service binder, the adaptation manager identifies when a new process (composite service) is being executed, and starts monitoring its behaviour. When adaptation is needed, the adaptation manager obtains the weights to be used during the ranking process and sends them to the service binder.
- *Service binder.* The ranking process performed by the service binder uses the weights of four parameters instead of three, considering the value of availability. After execution, information that indicates the service was available (or not) is also registered in the historical

database, along with the obtained availability for the executed service.

- *Service selector.* The estimated value of availability is obtained by interacting with the predictor, and stored in the service profile along with the estimated values of the other QoS parameters (response time, cost and energy consumption).

5.5. Evaluation

To evaluate the proposed optimization approach, two test cases were executed on the experimental environment described in section 3.5.4.1 (wide area network). Experiments were carried out to address the following question:

- Does the use of a proactive adaptation approach based on self-optimization helps improving the global QoS of composite services?

The work performed to provide an answer to this question involves the assessment of the behaviour of composite services when using the proposed proactive adaptation approach.

5.5.1. Test Cases

Two test cases have been used in order to assess the proposed approach. These models are BPEL processes that represent typical examples for service composition scenarios. Test case 1 is illustrated in Figure 5.4a, it implements an order booking process that validates the product availability, obtains the best price of the product from two different providers, selects the best provider, performs the payment, and finally completes the order. Test case 2 implements a travel planning process, as described in section 3.5.1. It is illustrated in Figure 5.4b. For simplicity, both diagrams only depict those activities that involve service invocations.

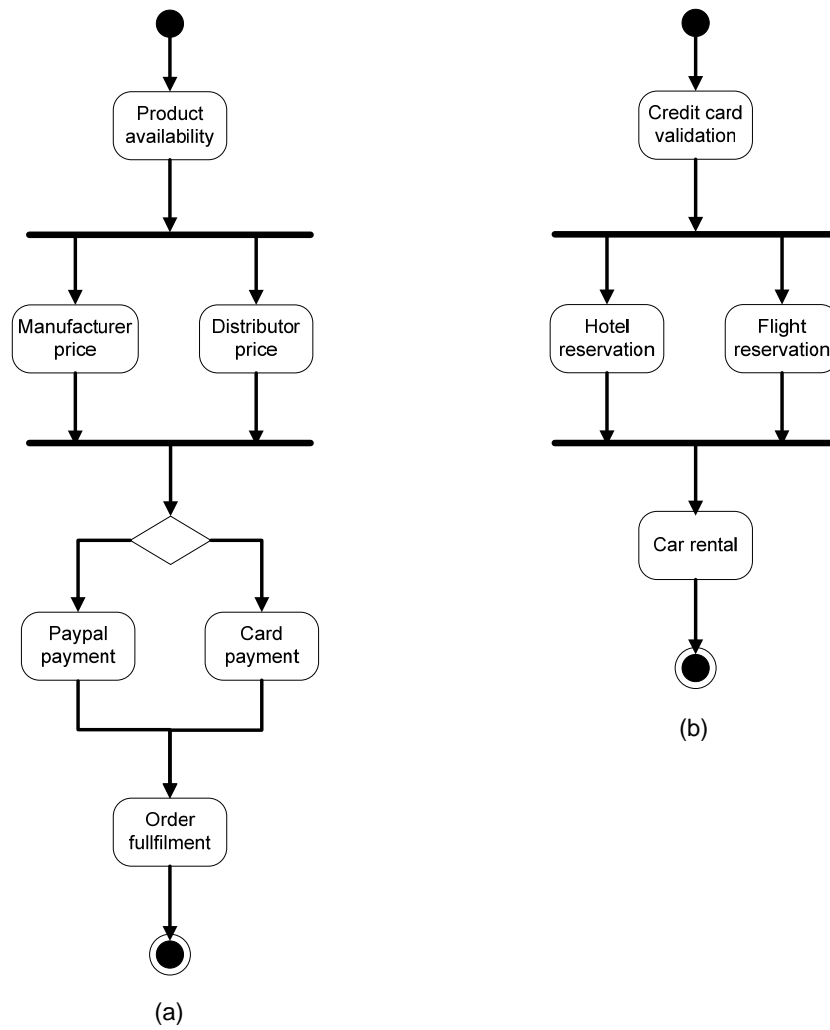


Figure 5.4. Test cases. (a) Order booking process. (b) Travel planning process.

Table 5.2. QoS parameters configuration.

Server	Set	Time delays (ms)	Cost	Power Consumption (W)	Availability
Node 2	S1	0	120	90	0.9
	S2	350	80		0.9
	S3	200	100		0.9
Node 3	S1	0	150	63	0.64
	S2	350	100		0.62
	S3	200	120		0.63
Node 4	S1	0	100	299	0.5
	S2	350	60		0.46
	S3	200	80		0.48

The initial QoS parameters configuration is similar to the one presented in previous chapters, where values were established based on the node where the service is running and the corresponding set. The main difference between previous configurations and the one used in this experiment is the definition of availability values. Information is shown in Table 5.2.

Similar to the experiments executed in chapters 3 and 4, per each of the tasks in the processes, there are 9 candidate services, distributed among the servers (nodes) that fulfil the required functionality, and offer different QoS; giving a total of 45 candidate services to be used in test case 1 and 36 for test case 2. These services were previously registered into the service registry (UDDI), and executed several times to populate the historical database and enable the estimation of their QoS attributes. The amount of information available in the historical database, before the execution of the experiments, corresponds to 1,000 records.

5.5.2. Service Selection Based on Fixed Weights

A service selection mechanism based on fixed weights was implemented to be compared with the proposed optimization approach. It extends the mechanism described in section 4.5.1 and follows similar steps. This approach uses Eq. 5.5 (presented in section 5.3.2) to obtain the services' score, where the values for w_1, w_2, w_3 and w_4 are set equally to 0.25. Services are ranked looking for the one with the highest score.

5.5.3. Experiment Description

In order to evaluate the proposed approach, both test cases were executed 100 times. These executions were performed using services deployed on remote servers (experimental environment described in section 3.5.4.1). The experiment was carried out using the proactive optimization mechanism, described in this chapter, and the service selection mechanism based on fixed weights described in the previous section.

In the proactive mechanism, the behaviour of the composition was monitored every second, and service selection used variable weight. Each

set of 100 runs was repeated 5 times to assess the consistency of the results based on statistical analysis.

5.5.4. Evaluation Results

Results show improvements in the global QoS values of the composition when using the proposed approach. Global QoS refers to the final values of the different QoS properties (response time, cost, energy consumption and availability). The plots shown in Figures 5.5 and 5.6 depict the behaviour of the order booking process, showing the mean values of the different QoS parameters after performing 5 sets of runs. For the proposed approach, the values of services' cost and servers' power consumption change over time, while for the fixed weights approach, remain constant. The evaluation of cost and power was performed every 3 minutes. For both cases, the value of availability changes according to the behaviour of the component services.

After analyzing the value of each of the QoS parameters, in both processes, it was identified that, in order to improve response time, energy consumption and availability, there was an increment in the composition's cost. In test case 1, results show that the proposed approach provides a mean reduction of 2% with a standard deviation of 6.7%, and a 95% confidence interval between 1,188.93ms and 1,203.59ms, in the measured response time values. Also, it can be noticed from Figure 5.5a, that it presents a more stable behaviour, without showing high peaks, as compared to the fixed weights approach. This is due to the constant evaluation of the QoS parameters during execution.

In terms of energy consumption, it is important to notice that this value is not only based on power consumption, but also influenced by time. As a result, a small response time may produce a small energy consumption value. Figure 5.5b shows the values corresponding to energy consumption, which have a similar behaviour to response time, and provide a mean reduction of 14.7% with a standard deviation of 18.9%, and a 95% confidence interval between 181Ws and 186.49Ws.

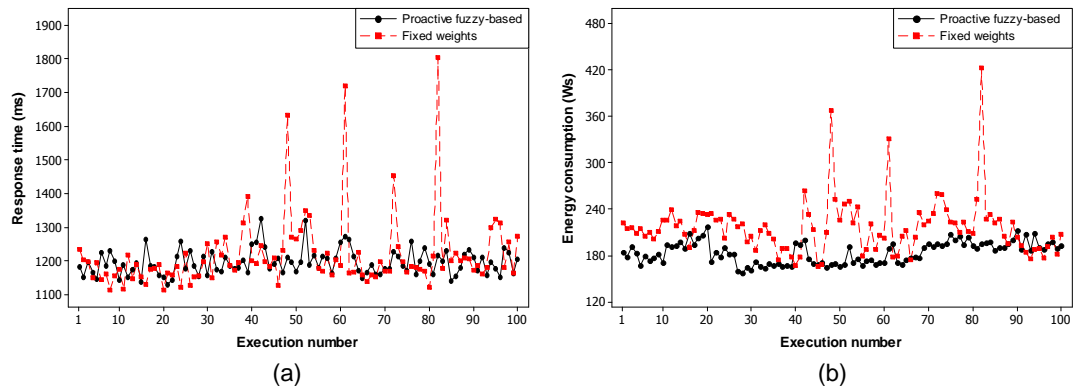


Figure 5.5. Order booking process results. (a) Response time. (b) Energy consumption.

Results also indicate that there is a significant improvement in the processes' availability, presenting a mean increase of 41% with a standard deviation of 35%, and a 95% confidence interval between 0.3675 and 0.4169. The availability values corresponding to the order booking process are illustrated in Figure 5.6a. Regarding cost, it can be noticed from Figure 5.6b that the use of the proposed approach turns into more expensive composite services. It shows a mean increase of 11% with a standard deviation of 8.4%, and a 95% confidence interval between 525.68 and 541.06.

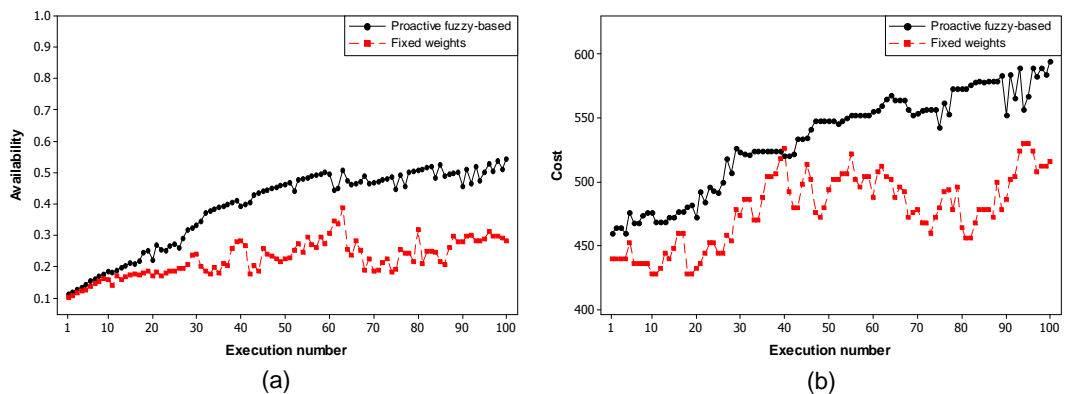


Figure 5.6. Order booking process results. (a) Availability. (b) Cost.

Summarized results are illustrated in Figure 5.7. The plot depicts the overall behaviour of the QoS parameters during the execution of test case 1 in both scenarios (proactive and fixed weights). It can be noted the increment in terms of cost (coloured in orange), and the improvements achieved with respect to the values of response time, energy consumption and availability (coloured in green).

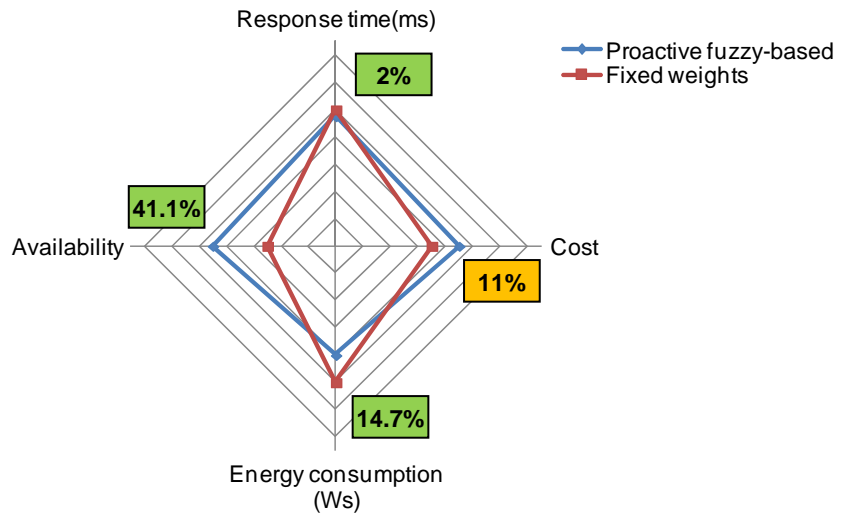


Figure 5.7. Summary of results - order booking process.

Results obtained from test case 2 show a similar behaviour; where response time, energy consumption and availability values are improved, while cost increases. In terms of response time, depicted in Figure 5.8a, it shows a mean reduction of 8.9% with a standard deviation of 16% and a 95% confidence interval between 622.24ms and 639.21ms. For energy consumption, shown in Figure 5.8b, the obtained mean reduction is 4.6% with a standard deviation of 29% and a 95% confidence interval between 60.75 Ws and 64.95 Ws.

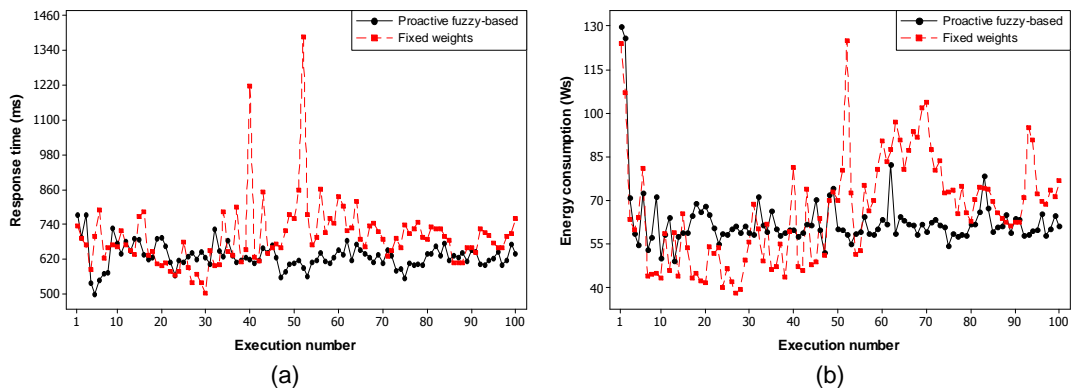


Figure 5.8. Travel planning process results. (a) Response time. (b) Energy consumption.

Regarding availability, the proposed approach provides an improvement of 18% with a standard deviation of 25% and a 95% confidence interval between 0.4909 and 0.5392. Finally, in terms of cost, it generates an increment of 12.5% with standard deviation of 6.8% and a 95% confidence interval between 545.34 and 557.09. The plots depicted in Figures 5.9a and 5.9b, illustrate the behaviour of the compositions' availability and cost, respectively.

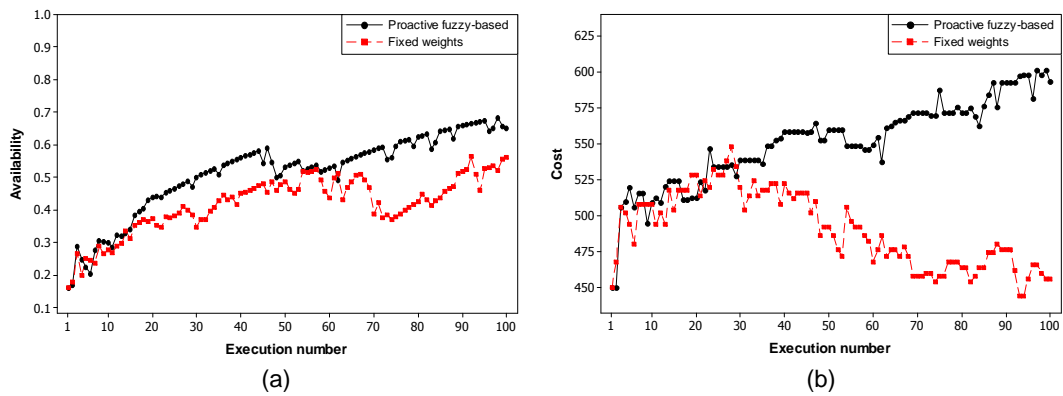


Figure 5.9. Travel planning process results. (a) Availability. (b) Cost.

In this set of experiments, the behaviour of the composite services in terms of response time has been considered based on the response time of component services, discarding the overheads caused by the engine. This overhead is around 3,200ms and 2,500ms, for test case 1 and test case 2, respectively.

5.5.5. Discussion

Based on the analysis of the weight values obtained by the optimization model and sent to the service binder, the parameter that had the higher impact within the adaptation process was energy consumption, followed by response time. Because of this, at the moment of selecting new services to be invoked, priority would be given to those that are being executed on servers with lower power consumption, and that show better performance (lower response time). Which, based on the QoS configuration, are the services that also involve higher costs. Different QoS configurations may give different results; however, because of the use of multiple QoS criteria, it is likely to find that not all the parameters can be improved.

When analyzing the results obtained during the experimental stage, it can be noted that the use of the proactive adaptation approach presented in this chapter has enhanced significantly the global QoS of the use case scenarios, with reductions of up to 8.9% in response time and 14.7% in energy consumption, and an improvement of 41% in availability; this is achieved with an average increment in cost of 11.75 %.

In terms of performance, the use of the proposed mechanism causes an average increment of 596ms in the invocation time per task (information

obtained using a database with 10 candidate services and 100 records per service). Overheads increase following a quadratic model. This behaviour was determined after performing various sets of executions increasing the number of candidate services and analyzing the measured execution time (see Figure 5.10).

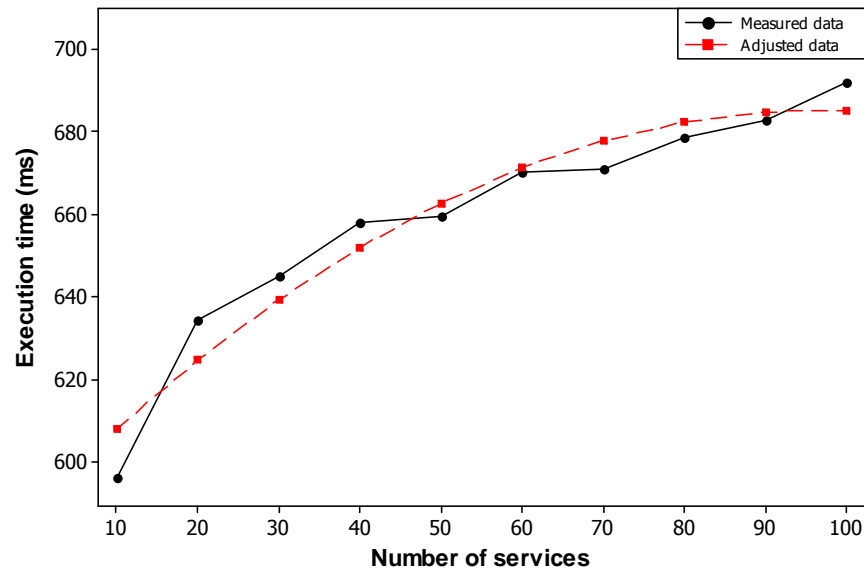


Figure 5.10. Execution time for different number of available candidate services.

Further assessment of the results shown in section 5.5 will be presented in chapter 6, along with a comparison between the proposed optimization model and relevant related work, highlighting their main differences.

5.6. Summary

This chapter has presented a proactive adaptation mechanism for service composition based on fuzzy logic. Ideas that motivate the development of the approach are discussed, followed by a review on work focused on providing proactive adaptation in service composition and service-based applications.

The proposed solution is then described in detail. It includes information regarding the service composition framework, and the extensions performed to the QoS model and optimization model. Following this, implementation aspects are then provided. Finally, evaluation of the

proposed approach is discussed in detail, including test cases definition, QoS parameters configuration, and results.

The following chapter will discuss the main contributions of this Thesis, providing a comparison between relevant related approaches and the research described in chapters 3, 4 and 5, and the overall assessment of the evaluation performed to the different models and mechanisms provided along those chapters.

Chapter 6

Comparison, Discussion and Overall Assessment of the Evaluation

This chapter presents the overall assessment of the evaluation performed to the adaptation approaches presented in previous chapters. A general overview of the research motivation is presented. This is followed by a comparison between related work and the research presented in this Thesis. Finally, the assessment of the evaluation is provided, including an overview of the experiments described in chapters 3, 4 and 5, along with the analysis of the gathered results and their limitations.

6.1. Overview

Development in the field of service composition has resulted in a set of dataflow models (orchestration and choreography), approaches (static, dynamic, manual and automatic) and techniques (model-driven, declarative, workflow-based, ontology-driven and AI-Planning) that enable composition from different perspectives. However, some challenges still remain open, which are closely related to automatic-dynamic service composition and include the implementation of mechanisms that enable: Quality of Service awareness, adaptive capabilities, risk awareness, conformance, security and interoperability.

The behaviour offered by services exhibits frequent variations, therefore, obtaining the expected results while running a service is not guaranteed. This situation has caused the need of mechanisms and tools focused on helping providers to ensure the provision of services with certain quality levels. When looking at Quality of Service awareness and adaptive capabilities, it can be considered that they complement each other, making possible to combine them while developing composition approaches focused on maintaining/improving the quality levels of composite services. QoS awareness refers to the capability of a composite service of being aware of its QoS aspects and those of the components involved; while adaptive capabilities aim to target changes within the composition, enabling it to

morph regarding those changes, in order to satisfy the consumer's requirements.

As described in chapter 2, adaptive mechanisms provide software systems with capabilities to self-heal, self-configure, self-optimize, self-protect, etc., in order to deal and mitigate the impact of unexpected events that can occur during service executions. The scope of this research is mainly focused in the development of models (mechanisms), that provide a service composition framework with capabilities to help providers in delivering services with the expected quality levels. These mechanisms react when: the QoS levels of the composition can be improved, the QoS levels of the composition are degrading, a component service is unavailable, and a component service fails.

Adaptation has been targeted primarily from a self-optimization perspective, looking at the QoS levels of the composition during the different stages of its execution. The optimization approaches consider situations where a number of the accumulated QoS values of the previous activity in the process are better than expected, providing the possibility of improving other QoS parameters. Also, they identify when the QoS of the composition is degrading. In situations where a service is unavailable or there is a service failure, a conservative self-healing approach was undertaken, preventing composite services from stopping their executions. However, performing changes every time there is a variation in the expected behaviour of the composition does not ensure the acquirement of the most favourable QoS values. Reason why, as part of this work it was considered the benefit of adaptation as a parameter to decide whether to adapt or not. In order to perform such evaluation, it was proposed the use of fuzzy logic as a tool to support the decision making process (described in chapter 4). The value of benefit of adaptation is obtained by analyzing the relationship between the values of the QoS parameters during the different stages of the composite service execution.

The use of reactive adaptation approaches may lead to increments in the response time and cost of composite services. In order to avoid such increments and identify the need of adaptation in advance, this research also targets self-optimization from a proactive perspective. As a result, a proactive adaptation mechanism for service composition based on fuzzy logic was developed (described in chapter 5).

6.2. Adaptation in Service Composition - Comparison and Discussion

The use of adaptation solutions may involve different aspects, based on the context where adaptation is being applied. In the context of service composition, some of the aspects that can be considered when using adaptation solutions include: goal, level, action, mechanism, stage of adaptation, and awareness level (described in detail in section 2.5.2).

The research presented in this work has targeted adaptation in service composition mainly from a self-optimization perspective. Work found in the literature related to the development of adaptation approaches in the area of Web service systems is presented in Tables 6.1 and 6.2, and summarized based on the adaptation aspects mentioned above. QoS parameters and self-adaptation properties are also considered as part of the criteria.

Table 6.1. Adaptation in service composition - part 1.

Authors & citations	Adaptation goal	Adaptation level	Adaptation action	Adaptation mechanism
Cardellini et al. [86]	Non functional	Web service	<ul style="list-style-type: none"> Service selection Coordination pattern 	Policy-based
Ardagna et al. [129]	Non functional	Web service	Service selection	Rules-based
Calinescu et al. [87]	Non functional	<ul style="list-style-type: none"> Web service Architectural 	<ul style="list-style-type: none"> Service selection Coordination pattern Resource allocation 	Policy-based
Bianculli et al. [92]	<ul style="list-style-type: none"> Functional Non functional 	Web service	Service selection	Feedback-based
Canfora et al. [107]	Non functional	<ul style="list-style-type: none"> Web service Workflow 	Service selection	---
Wenjuan et al. [90]	Non functional	Web service	Service selection	<ul style="list-style-type: none"> Agent-based Policy-based
Erradi et al. [91], [108]	Non functional	Web service	Service selection	Policy-based
This work	Non functional	Web service	Service selection	Rule-based

Table 6.2. Adaptation in service composition - part 2.

Authors & citations	Stage of adaptation	Awareness level	QoS	Self-adaptation properties
Cardellini et al. [86]	Runtime/ reactive	Event-aware	<ul style="list-style-type: none"> • Response time • Cost • Reliability 	<ul style="list-style-type: none"> • Self-adaptation • Self-healing
Ardagna et al. [129]	Runtime/ reactive	<ul style="list-style-type: none"> • Event-aware • Goal-aware 	<ul style="list-style-type: none"> • Response time • Cost • Reputation 	Self-healing
Calinescu et al. [87]	Load time	<ul style="list-style-type: none"> • Event-aware • Goal-aware 	<ul style="list-style-type: none"> • Performance • Reliability 	<ul style="list-style-type: none"> • Self-configuration • Self-optimization
Bianculli et al. [92]	Runtime/ proactive	---	Reputation	Self-healing
Canfora et al. [107]	Runtime/ reactive	Event-aware	<ul style="list-style-type: none"> • Time • Price • Availability • Reliability 	<ul style="list-style-type: none"> • Self-healing • Self-optimization
Wenjuan et al. [90]	<ul style="list-style-type: none"> • Runtime/ proactive • Runtime/ reactive 	Context-aware	Defined by user	<ul style="list-style-type: none"> • Self-healing • Self-management
Erradi et al. [91], [108]	Runtime/ reactive	Event-aware	Reliability	<ul style="list-style-type: none"> • Self-configuration • Self-healing
This work	<ul style="list-style-type: none"> • Runtime/ proactive • Runtime/ reactive 	Event-aware	<ul style="list-style-type: none"> • Response time • Cost • Energy consumption • Availability 	<ul style="list-style-type: none"> • Self-healing • Self-optimization

In terms of the aspects considered in Table 6.1, the research presented in this Thesis has a similar approach in comparison with the related work. The main differences are found in Table 6.2, where the use of proactive and reactive adaptation is only targeted by Wenjuan et al. in [90]. When looking at QoS parameters, this research proposes the use of energy consumption as a new quality attribute in service composition.

From a general perspective, adaptation approaches that implement self-optimization are mainly focused in the selection of services that offer high quality values and the use of utility functions. However, they only consider situations where quality levels decay. Besides, some of the adaptation strategies apply in the next execution of the composition, or require human specifications.

Table 6.3. Proactive adaptation/monitoring in service-based systems.

Authors & citations	Target situations	Adaptation actions	Validation	QoS parameters
Aschoff et al. [99], [164]	<ul style="list-style-type: none"> • Unavailable service • Malfunctioning service • Decrease in response time • Emergence of better services 	Service operation replacement ($1 - 1, 1 - n, n - 1, n - m$)	<ul style="list-style-type: none"> • Experiments in LAN • Prototype • Simulation 	<ul style="list-style-type: none"> • Response time • Cost
Leitner et al. [161]	Service Level Agreements violations	<ul style="list-style-type: none"> • Data manipulation • Service rebinding • Parameterization 	<ul style="list-style-type: none"> • Experiments in LAN • Prototype 	Response time
Yu et al. [59]	Performance decrease	<ul style="list-style-type: none"> • Service replacement • Backup in selection & reselection in execution 	Simulation	<ul style="list-style-type: none"> • Performance • Reliability • Cost
Tosi et al. [160]	Integration mismatches	Predefined adaptation strategies	Manual specification in prototype	---
Sammodi et al. [163]	<ul style="list-style-type: none"> • QoS violations • Malfunctioning service 	Dynamic service binding	<ul style="list-style-type: none"> • Simulation • Prototype 	Response time
Yuelong et al. [162]	<ul style="list-style-type: none"> • Missing output message • Missing input message • Un matching function invocation • Property violated 	--	<ul style="list-style-type: none"> • Experiments in LAN • Prototype 	---
This work	<ul style="list-style-type: none"> • QoS Degradation • Malfunctioning service • Unavailable service • Improvement in QoS (global perspective) 	<ul style="list-style-type: none"> • Service selection • Dynamic service binding 	<ul style="list-style-type: none"> • Experiments in WAN • Prototype 	<ul style="list-style-type: none"> • Response time • Cost • Energy consumption • Availability

The approach followed in this work proposed that self-optimization can be also targeted when one of the QoS values of the entire composition is better than expected in certain point of the execution. It considers that this behaviour provides some slack that can be used while selecting the next service in the process, enabling the improvement of other QoS attributes.

As mentioned in previous chapters, this research has proposed the use of fuzzy logic as a tool to perform the decision making process when evaluating the QoS values of composite services, and determine the benefit of performing adaptation. Approaches found in the literature that use fuzzy logic in service-based systems are mainly focused in service selection, and even though they evaluate the QoS values of the services, they do not consider the benefit of adaptation as a parameter.

The optimization approach presented in chapter 5 works as part of a proactive adaptation mechanism for service composition. A comparison between work related to the provision of proactive mechanisms in service-based systems is presented in Table 6.3. This comparison was performed based on different criteria, which include: target situations, adaptation actions, form of validation and QoS parameters.

It was found that this work is the only one that considers as a target situation the improvement of the global QoS of composite services, and was validated by performing experiments on a wide area network. Regarding QoS parameters, most of the related approaches are focused on response time, while this work also considers cost, energy consumption and availability.

6.3. Assessment of the Evaluation

Results obtained after evaluating the three approaches proposed and described along this Thesis, show that the use of the optimization mechanisms while executing composite services provide meaningful improvements in their global QoS values, when comparing to a service selection mechanism based on fixed weights.

6.3.1. Overview of the Experiments

The evaluation and assessment of the optimization approaches involved the use of test cases with various candidate services, and the configuration of two experimental environments. As a result, two BPEL processes were modelled. They represent typical examples for service composition scenarios: travel planning and order booking (for further details see section 5.5.1). Per each of the tasks in the processes, there were 9 candidate services available, giving a total of 36 services for travel planning and 45 for order booking.

The experimental environments were setup with the following characteristics:

- *Environment 1 (local area network)*. It consists of three nodes, one computer with Windows Vista (node 1); and two virtual machines with lubuntu 11.10 (nodes 2 and 3). Node 1 hosts the BPEL engine, service registry, historical database and one application server. Nodes 2 and 3, host one application server each. Web services are allocated in the application servers.
- *Environment 2 (wide area network)*. It consists of 4 nodes configured on a wide area network, distributed between United Kingdom and Germany, with estimated values for bandwidth and latency around 32 Mbit/s and 29ms, respectively. Node 1 is a computer with Windows Vista (located in United Kingdom). This node hosts the BPEL engine, service registry and historical database. Nodes 2 to 4 are virtual machines setup on remote servers (located in Germany). These nodes host one application server each, which contains 3 sets of Web services.

Initial values of QoS parameters for the candidate services used in the experiments were established based on the node where the service was running and the corresponding set, as described in Table 6.4.

Table 6.4. QoS parameters configuration.

Server	Set	Time delays (ms)	Cost	Power Consumption (W)	Availability
Node 1 (*2)	S1	0	120	90	0.9
	S2	350	80		0.9
	S3	200	100		0.9
Node 2 (*3)	S1	0	150	63	0.64
	S2	350	100		0.62
	S3	200	120		0.63
Node 3 (*4)	S1	0	100	299	0.5
	S2	350	60		0.46
	S3	200	80		0.48

* Nodes corresponding to environment 2.

Each of the proposed optimization approaches evaluates a different subset of QoS parameters. Table 6.5 shows the experimental setup used to evaluate each of the approaches, subset of QoS parameters considered and number of executions performed.

Table 6.5. Summary of experiments configuration.

Approach	ID	Environment	Test case	QoS parameters	No. of executions
Variable weights	1A	LAN	Travel planning	<ul style="list-style-type: none"> • Response time • Cost 	50
	1B	WAN			50 (x3)
Fuzzy based	2A	LAN	Travel planning	<ul style="list-style-type: none"> • Response time • Cost • Energy consumption 	50 (x5)
	2B	WAN			50 (x3)
Proactive	3A	WAN	Order booking	<ul style="list-style-type: none"> • Response time • Cost • Energy consumption • Availability 	100 (x5)
	3B	WAN	Travel planning		100 (x5)

6.3.2. Analysis of Results

A summary of the results obtained when evaluating the effectiveness of the proposed approaches is presented as follows. These results are compared against measured data obtained when executing the test cases with a

service selection mechanism based on fixed weights, using the same initial QoS parameters configuration and execution environment.

The first set of experiments (1A and 1B) corresponds to the evaluation of the use of variable weights during service selection as part of a self-optimization mechanism. This approach was described in chapter 3. Figure 6.1 illustrates the QoS values measured after performing experiment 1A, which corresponds to the execution of the travel planning process over a local area network.

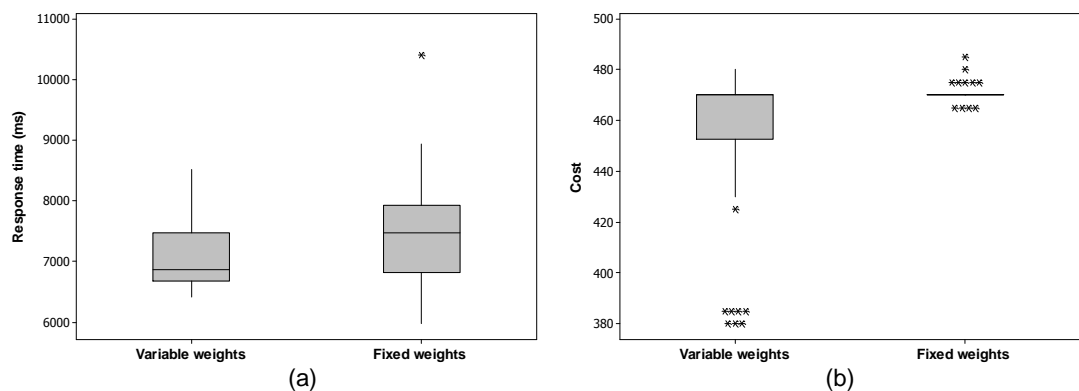


Figure 6.1. Experiment 1A- comparison between variable weights and fixed weights approaches. (a) Response time. (b) Cost.

Values regarding response time are depicted in Figure 6.1a. It can be noted that for the optimization approach, the obtained values are smaller as compared to the fixed weights mechanism. Even though, the smallest response time was found in the fixed weights approach, it also presents the highest value and a higher median. In terms of cost (Figure 6.1b), the highest value was found in the fixed weights approach, while the smallest on the optimization one. Most of the values obtained using fixed weights fall on the median, with few outliers. This is caused by the lack of variation in the cost of Web services and the use of the same service in multiple executions.

When analyzing the results obtained from executing the travel planning process in a wide area network (experiment 1B), differences in response time are not as notorious as compared with those found in cost, as illustrated in Figure 6.2. Similar to the behaviour found when executing the experiment in the local area network, response time values measured for the optimization approach are smaller as compared with the fixed weights approach, as depicted in Figure 6.2a. Regarding cost, there is can be noted a meaningful reduction, where most of the values obtained with the

optimization approach are smaller than those obtained with fixed weights (see Figure 6.2b).

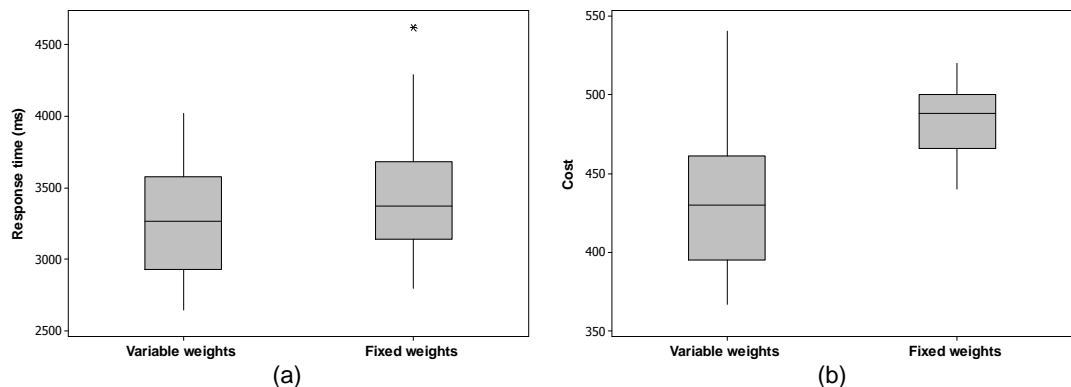


Figure 6.2. Experiment 1B- comparison between variable weights and fixed weights approaches. (a) Response time. (b) Cost.

Based on the gathered results from experiments 1A and 1B, it can be concluded that by using the proposed approach, there can be achieved significant improvements in the global QoS of composite services, with reductions up to 14% in response time and 16% in cost.

The second set of experiments (2A and 2B) was performed in order to evaluate the fuzzy logic based optimization mechanism described in chapter 4. In these experiments, there were inserted variations regarding service's cost and servers' power consumption when using the proposed approach. These variations were based on the models presented in section 4.5.2. For the fixed weights approach, both values remained constant according to their initial configuration.

Figure 6.3 illustrates the summary of results obtained when performing experiment 2A, where the travel planning process was executed on a local area network. As can be noted from Figure 6.3a, the proposed fuzzy approach provided the smaller response time values, as compared with the fixed weights approach. Gathered results regarding cost and energy consumption, depicted in Figures 6.3b and 6.3c, also provided smaller values.

A summary of results collected after performing the travel planning process in a wide area network (experiment 2B) is depicted in Figure 6.4. When executing the composite services over a WAN, there was a trade-off, where one of the QoS parameters was degraded in order to maintain/improve the values of the others.

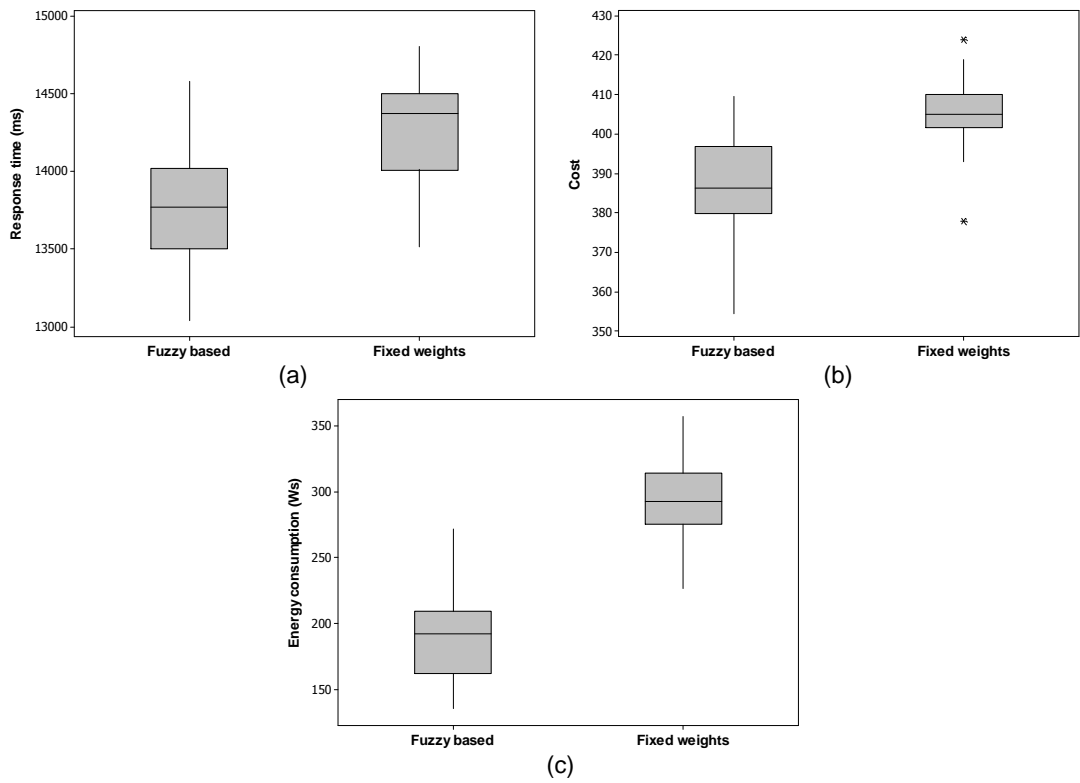


Figure 6.3. Experiment 2A- comparison between fuzzy based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption.

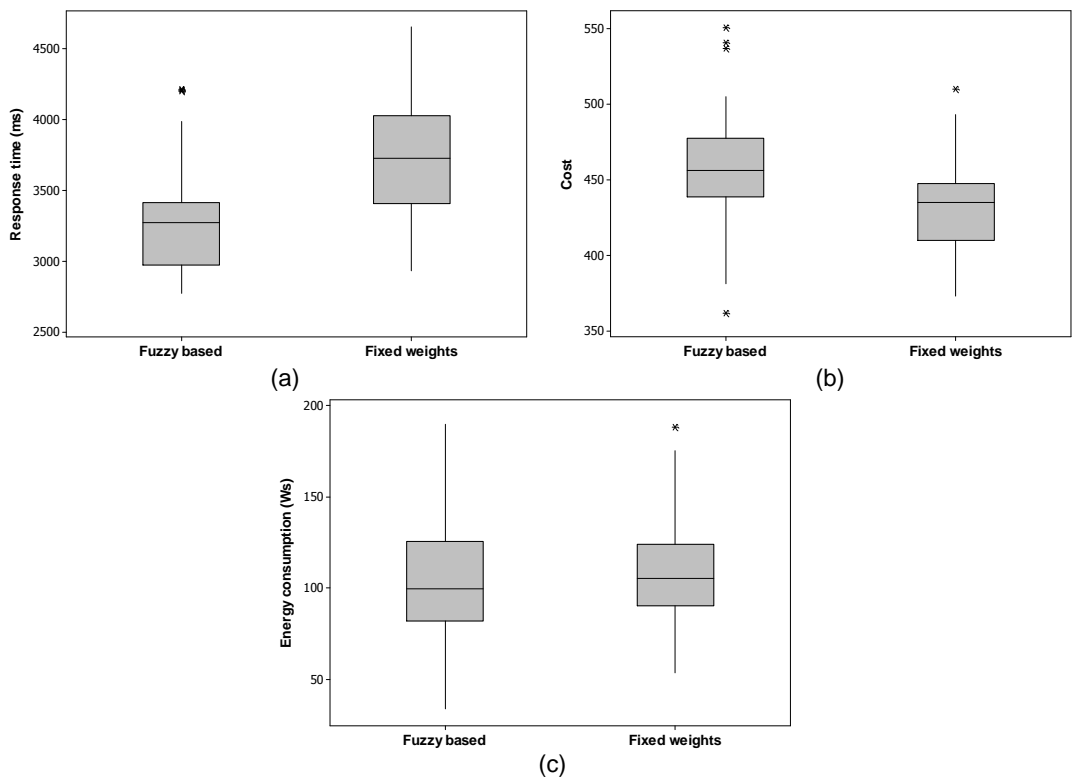


Figure 6.4. Experiment 2B- comparison between fuzzy based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption.

Average value of cost increased, while response time was reduced and energy consumption remained very close to the values obtained when using fixed weights (Figure 6.4). This situation may be caused by variations in the response time of component services. As response time influences the value of energy consumption, a large response time may generate a large energy consumption value. When looking at results from individual executions, they showed a similar behaviour (described in section 4.5.5.1).

In conclusion, results collected in experiments 2A and 2B indicate that the use of the proposed fuzzy logic based optimization approach, helps to obtain meaningful improvements in the global QoS of composite services, providing reductions up to 20.5% in response time, 33.4% in cost and 31.2% in energy consumption.

The third set of experiments (3A and 3B) was carried out to evaluate the proactive adaptation mechanism described in chapter 5. In these experiments, the order booking process and the travel planning process were executed over a wide area network.

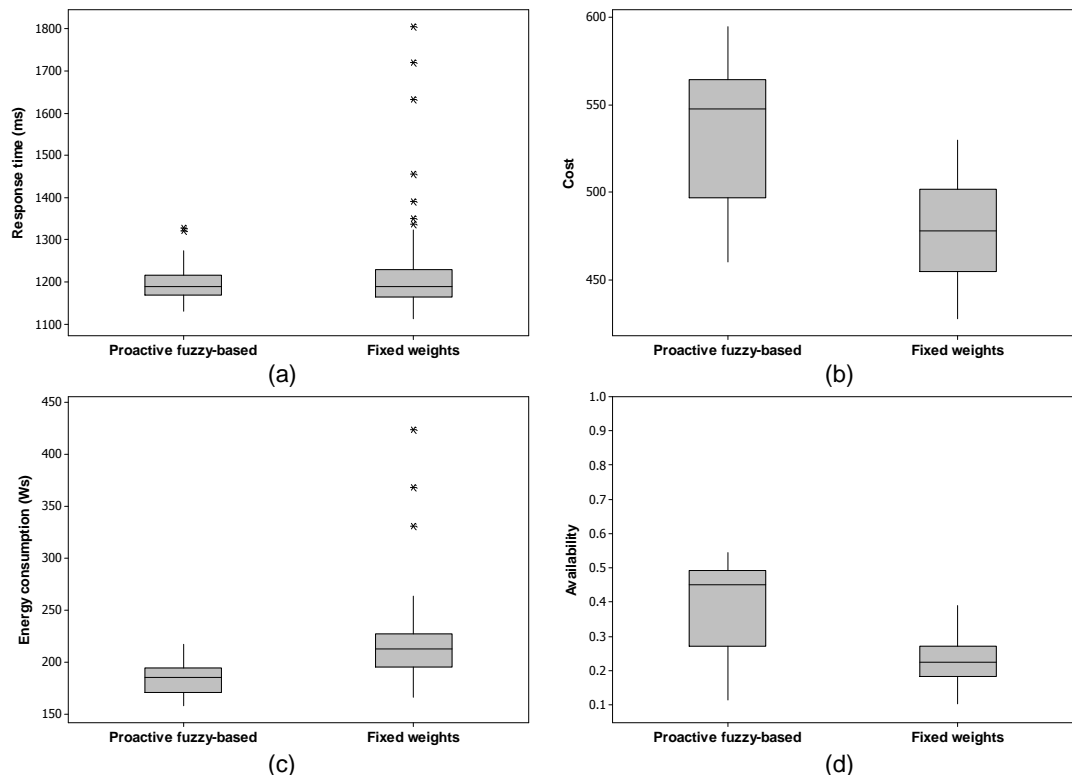


Figure 6.5. Experiment 3A- comparison between proactive fuzzy-based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption. (d) Availability.

Figure 6.5 illustrates the summary of results obtained after performing experiment 3A. It can be observed a trade-off between QoS values. When comparing the proactive fuzzy-based approach with the fixed weights approach, it can be noted that in order to improve response time, energy consumption and availability, there is an increment in terms of cost (Figure 6.5b). Results obtained from experiment 3B showed a similar behaviour, as depicted in Figure 6.6.

One reason that may influence this behaviour is the relationship between the values of quality parameters exhibit by the services, as those services with lower energy consumption and higher availability, also display higher costs.

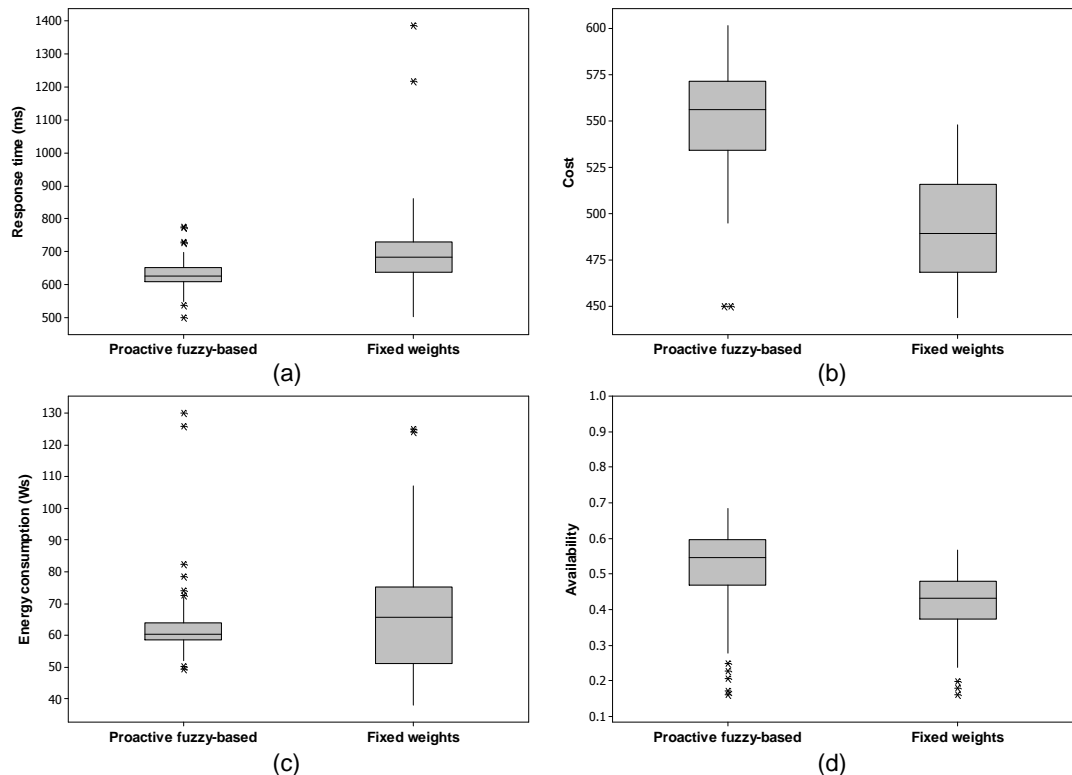


Figure 6.6. Experiment 3B- comparison between proactive fuzzy-based and fixed weights approaches. (a) Response time. (b) Cost. (c) Energy consumption. (d) Availability.

Results obtained after performing experiments 3A and 3B show that by using the proposed proactive adaptation approach, it is possible to enhance significantly the global QoS of the use case scenarios, with reductions of up to 8.9% in response time and 14.7% in energy consumption, and an improvement of 41% in availability; this is achieved with an average increment in cost of 11.75 %.

6.3.3. Limitations

The experiments summarized along this section show very encouraging results regarding the effectiveness of the proposed QoS optimization mechanisms within the context of service composition. However, the experimentation has some limitations, which include:

- *The use of QoS values that were not obtained from real services.* Initial configuration values for cost and availability were assigned based on assumptions, while energy consumption on servers' power consumption selected from the Energy Star report [158]. The use of real services and real QoS data during the experimentation stage is needed to assess the effectiveness of the approaches in real world scenarios.
- *The use of test cases with limited number of elements that involved service invocations.* The test cases used to perform the evaluation of the optimization approaches were modelled inspired in composite services found in the literature, but they have a limited number of service invocations and structures. Experimentation with more complex and realistic test cases is necessary to analyze the behaviour of the optimization mechanisms and ensure they are suitable not only for small/medium size scenarios.
- *The use of a limited number of parameters when applying fuzzy logic.* The number of parameters has a strong influence in the number of rules used by the fuzzy support systems. The number of rules increases considerably when using more than 4 parameters, which turns the management of the rules engine into a highly complex task. The use of fuzzy logic may be unfeasible when considering scenarios that involve the analysis of a high number of QoS criteria.

6.4. Summary

This chapter has presented the overall assessment of the evaluation performed to the adaptation approaches described in previous chapters. A general overview of the research motivations was presented, which include the use of QoS variations in order to determine adaptation, the need of

performing adaptation and the use of proactive adaptation in service composition.

A comparison between related work and the research presented in this Thesis is then provided. This comparison was performed from two perspectives. Firstly, from a general point of view, based on different criteria which included: goal, level of adaptation, action, mechanism, stage of adaptation, QoS parameters and awareness level. Secondly, from a proactive point of view, summarizing work focused in providing proactive mechanisms for service-based systems. The criteria considered include: target situations, adaptation actions, form of validation and QoS parameters. The main differences between this research and related approaches are then highlighted.

Finally, the assessment of the evaluation is provided. It includes an overview of the experiments described in chapters 3, 4 and 5, along with the analysis of the gathered results and their limitations. The next chapter will present a summary of the work described in this Thesis, followed by the key contributions and directions for future work.

Chapter 7

Conclusion and Future Work

This chapter provides a summary of the work presented in the Thesis. Then, the major contributions of the research are given. Following this, a discussion on some directions that can be explored as part of future work is presented.

7.1. Summary

The work presented in this Thesis is focused on the research of Quality of Service awareness and adaptation in service composition. It is primarily centred on self-optimization, looking at changes in the QoS levels of composite services during the different stages of their execution. Self-optimization has been targeted with three approaches, which consider different QoS parameters and look at adaptation from reactive and proactive perspectives. The approaches were implemented in service composition frameworks and evaluated through the execution of test cases.

chapter 2 presents background concepts that help in the understanding of the research described in this document. It begins by describing Service Oriented Architectures to help introducing Web services and service composition. Then, the concept of Web services is explored in detail, along with service related standards, the service life cycle, and some of the benefits of using services when developing software solutions. Relevant concepts related to service composition are then presented, including dataflow models, composition languages and main challenges in the field.

The concepts of Quality of Service and Service Level Agreements were provided in the context of service oriented environments, and followed by a discussion on related work in QoS management and QoS estimation in service composition. Adaptive service composition is described from the perspective of autonomic computing. Self-* properties are defined and related to events that can occur while executing composite services. Other relevant adaptive approaches applied in the area of service oriented

environments are then discussed. Finally, various decision support systems applied within adaptive mechanisms are described.

After presenting the background concepts related to service composition and adaptation, chapter 3 goes into the description of a QoS optimization model for service composition. It begins by providing the motivation behind the development of the work, followed by a discussion of work related to the provision of adaptation in service composition. An outline of the proposed solution is then described, along with a detailed portrayal of its elements. Alongside, implementation aspects regarding the elements of the solution are provided. This chapter concludes by presenting the evaluation of the proposed model. It includes details concerning the description of experimental objectives, experiments and results.

Chapter 4 describes a QoS optimization model for service composition based on fuzzy logic. This model is an extension of the approach described within chapter 3. It provides the motivation behind the development of the approach, followed by a discussion on related work. The proposed solution is described, including details about the QoS model, decision support systems and optimization model. Implementation of the extensions performed to the composition framework are given. Finally, the evaluation of the model is provided, covering the description of the experimental setup, dynamic QoS parameters and results.

A proactive adaptation mechanism for service composition is presented in chapter 5. This mechanism is built as an extension of the QoS optimization model described in chapter 4. It begins by providing the ideas that motivate the development of the approach. Then, a review on work focused on providing proactive adaptation in service composition and service-based applications is presented. Following this, the proposed solution is described, providing details about the service composition framework and modifications performed to the QoS model and optimization model presented in previous chapters. Implementation aspects are provided. This chapter concludes by presenting the evaluation of the proposed approach, including test cases definition, QoS parameters configuration and results.

Chapter 6 presents an overall assessment of the evaluation performed to the different models and mechanisms proposed in chapters 3, 4 and 5. It provides a summary of the research motivations for the different

models. This is followed by a comparison between related work and the research presented along this Thesis, and a discussion that underlines the main differences. From a general point of view, the criteria used to establish the comparison included: goal, level of adaptation, action, mechanism, stage of adaptation, QoS parameters and awareness level. From a proactive point of view, the criteria considered include: target situations, adaptation actions, form of validation and QoS parameters. The assessment of the evaluation is then provided, along with the analysis of the gathered results. It includes an overview of the experiments described in chapters 3, 4 and 5. Evaluation results showed that the proposed mechanisms enhanced the global QoS values of the compositions, with significant improvements regarding the evaluated QoS parameters. Finally, the limitations of the experimentation are presented.

7.2. Contributions

The main contributions of the work presented in this Thesis are summarized in the following points:

- *QoS optimization mechanisms for service composition.* This research proposes three QoS optimization mechanisms which consider diverse QoS criteria from a global perspective. These mechanisms are not only focused on targeting QoS degradation, they also consider when some of the measured QoS values at certain point of the composite service execution are better than expected, enabling the improvement of other QoS attributes. Two of these mechanisms involve the use of fuzzy logic as a decision making tool (described in chapters 4 and 5). They take into consideration the benefit of adaptation, value which is obtained by analyzing the measured values of the QoS attributes. The use of the benefit of adaptation helps determining whether adaptation is needed or not, avoiding to trigger adaptation after every variation in the behaviour of the composition.
- *Conceptual frameworks that enable QoS aware and adaptive service composition.* This research presents two abstract system models that enable QoS aware and adaptive composition. The first framework (described in section 3.3.1) provides adaptation from a reactive perspective. Its main components can be summarized as:

composition engine, adaptation manager, service binder, service selector, predictor, sensors and effectors. On the other hand, the second framework (described in section 5.3.1) provides adaptation from a proactive perspective. Its core components are similar to those used in the reactive framework, but they interact in a different manner. In the proactive framework, the adaptation manager works semi-independent to the rest of the components and sends information to the service binder when adaptation is needed. In the reactive framework, it is invoked within the binder.

- *Prototypes implementations for reactive and proactive service composition.* Prototyping helps performing experiments in real environments, which provide sensible results when evaluating adaptation mechanisms. In order to assess the proposed QoS optimization mechanisms, two prototypes were implemented as extensions of an open source composition engine. A reasonable understanding of the composition language and the execution engine was necessary to extend the engine's functionality and enable both reactive and proactive adaptation.
- *Discovery of benefits offered by the use of the QoS optimization mechanisms in service composition.* The effectiveness of the proposed QoS optimization mechanisms presented in this Thesis was demonstrated through a series of experiments, which involved the use of two experimental environments (local area network and wide area network), and two test cases (travel planning and order booking). Results showed that the mechanisms were effective, providing significant improvements in terms of global QoS when executing composite services. In some situations a trade-off was found, where one of the QoS parameters is degraded in order to maintain/enhance the values of the others.

The contributions provided by this work aim to bring new solutions to QoS awareness and adaptation in the area of service composition, targeting QoS optimization focussed not only in maintaining, but improving the QoS parameters of composite services. The evaluation of the QoS optimization mechanisms demonstrated that the QoS parameters of composite services, at some point of their execution, can be better than expected. Based on this information, decisions can be made in order to improve the global QoS of

the composition. In addition, it was identified that when using multiple QoS criteria, it is likely to find that not all the parameters can be improved using the proposed approaches.

7.3. Future Work

There are further directions that can be considered in order to extend the research work presented in this Thesis. Some of these directions are described as follows:

- *The use of Dynamic Service Level Agreements on top of the composition framework.* The composition framework is not considering the use of SLAs and user's QoS requirements. It is focussed on providing the best possible global QoS, based on the available information it has on the component services behaviour. The use of SLAs between the framework and the customer would provide certain limits to the QoS parameters regarding the customer's requirements. If these SLAs are dynamic, it will also enable the composition to re-negotiate with the customers when the composition's global QoS is deviating from the original request. Dynamic SLAs [165] could provide the composition with a flexible approach to handle QoS requirements, helping to ensure customer's satisfaction.
- *The development of realistic models to define the behaviour of component services.* The QoS values used during this research were not obtained from real services. Cost and availability were assigned aiming to support a wide range of values, while energy consumption on servers' power consumption selected from the Energy Star report [158]. The use of QoS values obtained from real services would enable the development of models that can be applied when predicting the services behaviour, helping to assess the effectiveness of the proposed approaches in real world scenarios.
- *The assessment of the proposed approaches using different decision support systems.* The use of fuzzy logic may be unfeasible when considering scenarios that involve the analysis of a high number of QoS criteria. This is caused by the increase rate in the number of

rules involved in the system, which can turn the management of the rules engine into a highly complex task. The use of different decision making tools (such as decision trees [148], reinforcement learning [145], genetic algorithms [107], etc.) to assess the proposed approach, may provide some flexibility when the evaluation of QoS parameters involves a high number of criteria. The approaches can also be evaluated using different decision support systems with the same number of QoS parameters, looking at performance, usage of resources and obtained results.

- *The use of diverse QoS parameters.* This research considers four QoS parameters (response time, cost, energy consumption and availability). However, there is an extensive list of QoS parameters that can be applied in service oriented environments [4], [58]. New QoS criteria can be considered, based on the objectives of composite services and users' requirements.
- *The use of other estimation mechanisms.* Estimation is limited to the use of the average of the last N elements. The use of other estimation mechanisms (like those described in [72]) may provide more accurate predictions, which could have an impact on the global QoS values of the composite services.
- *The extension of the adaptation mechanism.* During the execution of composite services, adaptation is performed by using service selection/re-selection. The adaptation mechanism could be extended by adding features that enable service replacements considering different structures with the forms: $1 - n$, $n - 1$ and $n - m$ (similar to the approach presented in [99]). This would remove the limitation of having at least one component service to fulfil every task within the composition by using a functional equivalent structure.

Appendix A

Comparison of Predictive Algorithms to Support QoS Estimation

One set of experiments was developed with the aim of evaluating predictive algorithms capabilities to obtain the estimated QoS of Web services. In this context QoS data is limited to response time. The evaluated algorithms are:

- Single last observation (SLO).
- Running average (RA).
- Running average for the last 10 executions (RA-10).
- Low pass filter (LPF).

Two atomic services (WS1 and WS2) were deployed on Apache Tomcat with Axis, historical data was stored in a MySQL database and the client, which includes the algorithms implementation, was developed as a java application. Historical data was collected by invoking each service 1000 times and measuring response time on the client side. Using the predictive algorithms mentioned above, response time was forecasted 40 times per service. For each prediction the WS was invocated and data recorded, in order to compare real vs. estimated response time.

Figures A.1 and A.2 illustrate the deviation between estimated and actual response time obtained on the executions of WS1 and WS2, respectively. Results obtained from the execution of WS1 show that the running average of all the historical data brought the set of values that differ most with the actual response time. Single last observation results, presented some accurate predictions, however, when abrupt changes occur, estimated values were not close to the measured ones. On the other hand, low pass filter and the running average of the last 10 executions showed estimations with closer values to the observed behaviour of the services. Results obtained from the execution of WS2 show a similar behaviour.

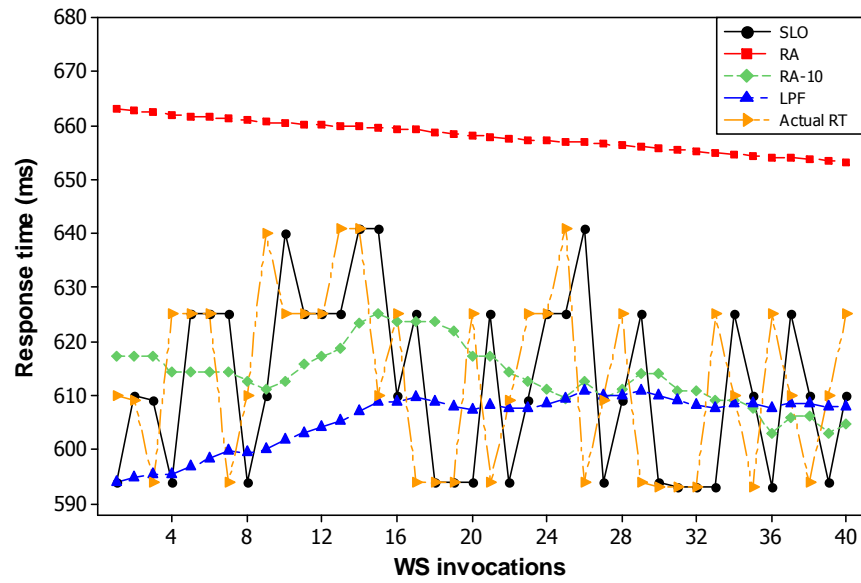


Figure A.1. Comparison of estimated values v_s , real (WS1).

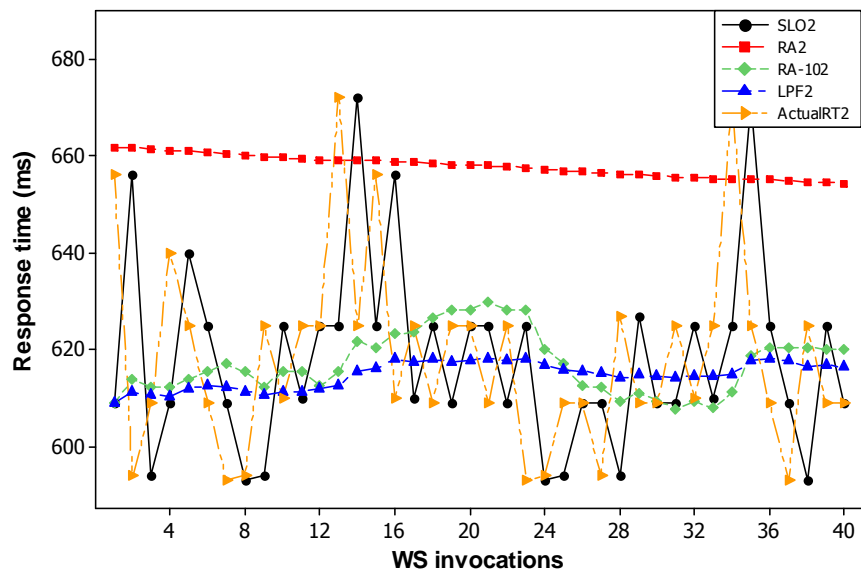


Figure A.2. Comparison of estimated values v_s , real (WS2).

To obtain a better understanding on the results (estimations vs. real response time), the relative error was computed per each estimated value using the following formula [72]:

$$e(p_i) = \frac{|x(p_i) - x_0(p_i)|}{x_0(p_i)} \quad (\text{A.1})$$

Where:

$x(p_i)$ corresponds to the estimated data,
 $x_0(p_i)$ corresponds to the real measured data.

After analyzing the results of WS1, the algorithm that presented the largest error rate is the running average of all the collected data, with an average error rate of 7.43%. Single last observation values showed an average error of 2.55%; running average of the last 10 invocations 2.44%; and low pass filter 2.64%. In the case of WS2, the algorithm that presented the largest error rate is the running average of all the collected data, with an average of 6.72%. Single last observation values showed an average error rate of 3.39%; running average of the last 10 invocations 2.55%; and low pass filter 2.43%.

Appendix B

Self-Healing Features

The use of self-healing capabilities has been considered as part of this work, with the aim of preventing composite services from stopping their executions in situations where a component service is unavailable or a service failure occurs.

- *Unavailable service.* The service is down or has no network connection.
- *Service failure.* The service does not finish its execution or sends an error message.

In order to provide the features that enable such capabilities, a secondary adaptive mechanism was designed and developed within the composition frameworks. The heuristic behind this mechanism is presented in Figure B.1. The notation used is shown as follows. Let,

- $T = \{t_1, t_2, \dots, t_n\}$ be the set of n tasks in process P .
- j be the task number, where $t_j \in T$.
- $S_j = \{s_1, s_2, \dots, s_m\}$ be the set of m services that can be used to implement t_j .
- i be the service number, where $s_i \in S$.
- $estT, estC, estE, estA$ be estimated QoS values corresponding to response time, cost, energy consumption and availability for a service.
- $execS, monT$ be the values corresponding to the execution status and monitored execution time for a service.
- max be the default value set as the maximum execution time for a service.
- msg be a response message obtained after executing s_i .
- w_1, w_2, w_3, w_4 be weights used to obtain the score of a service (see Eq. 5.5).

```

EvaluateService( $s_i, t_j$ )
1  let  $s$  be a service
2  if  $s_i.execS = finish$ 
3      if  $s_i.msg = error$ 
4           $s = GetService(s_i, t_j)$ 
5  else if  $monT > max$ 
6       $s = GetService(s_i, t_j)$ 
7  return  $s$ 

GetService( $s_i, t_j$ )
1   $S_j = RetrieveServices(t_j)$ 
2  remove  $s_i$  from  $S_j$ 
3   $w_1 = 0.85$ 
4   $w_2 = w_3 = w_4 = 0.05$ 
   //score computation and service ranking phase
5  for  $i = 0$  to ( $S_j.length - 1$ )
6       $S_j[i].W = w_1 S_j[i].estT + w_2 S_j[i].estC +$ 
         $w_3 S_j[i].estE + w_4 S_j[i].estA$ 
7  sort  $S_j$  by  $W$  descendent
8  return  $S_j[0]$ 

```

Figure B.1. Self-healing evaluation heuristic.

When a component service has been invoked, its response message and execution time are monitored by the system. If the execution of the s_i has finished (step 2), the value of its msg is evaluated (step 3). If it contains an error message, a new service is retrieved from the list of equivalent services (step 4). If the service is still running and $monT$ is longer than max (step 5), it is considered as a failure and a new service is selected from the service list (step 6). Finally, the heuristic returns the service replacement (step 7).

The replacement is obtained from S_j , after removing the faulty service s_i . Scores of the elements within S_j are computed giving priority to response time, since a new execution will increment the response time of P (see *GetService* function).

Appendix C

Fuzzy Rules

The following table contains the set of rules used to evaluate the benefit of adaptation when using four QoS parameters as input (response time, cost, energy consumption and availability).

Table C.1. Benefit of adaptation fuzzy rules - extended.

1	<p>IF (respTime IS high AND cost IS low AND energy IS low AND availability IS high) OR (respTime IS low AND cost IS high AND energy IS low AND availability IS high) OR (respTime IS low AND cost IS low AND energy IS high AND availability IS high) OR (respTime IS low AND cost IS low AND energy IS low AND availability IS low) THEN BoA IS veryhigh</p>
2	<p>IF (respTime IS high AND cost IS high AND energy IS low AND availability IS high) OR (respTime IS high AND cost IS medium AND energy IS low AND availability IS high) OR (respTime IS high AND cost IS low AND energy IS high AND availability IS high) OR (respTime IS high AND cost IS low AND energy IS medium AND availability IS high) OR (respTime IS high AND cost IS low AND energy IS low AND availability IS medium) OR (respTime IS high AND cost IS low AND energy IS low AND availability IS low) OR (respTime IS medium AND cost IS high AND energy IS low AND availability IS high) OR (respTime IS medium AND cost IS low AND energy IS high AND availability IS high) OR (respTime IS medium AND cost IS low AND energy IS low AND availability IS low) OR (respTime IS low AND cost IS high AND energy IS high AND availability IS high) OR (respTime IS low AND cost IS high AND energy IS medium AND availability IS high) OR (respTime IS low AND cost IS high AND energy IS low AND availability IS medium) OR (respTime IS low AND cost IS high AND energy IS low AND availability IS low) OR (respTime IS low AND cost IS medium AND energy IS high AND availability IS high) OR (respTime IS low AND cost IS medium AND energy IS low AND availability IS low)</p>

	<p>availability IS low) OR (respTime IS low AND cost IS low AND energy IS high AND availability IS medium) OR (respTime IS low AND cost IS low AND energy IS high AND availability IS low) OR (respTime IS low AND cost IS low AND energy IS medium AND availability IS low) THEN BoA IS high</p>
3	<p>IF (respTime IS high AND cost IS high AND energy IS medium AND availability IS high) OR (respTime IS high AND cost IS high AND energy IS low AND availability IS medium) OR (respTime IS high AND cost IS medium AND energy IS high AND availability IS high) OR (respTime IS high AND cost IS medium AND energy IS medium AND availability IS high) OR (respTime IS high AND cost IS medium AND energy IS low AND availability IS medium) OR (respTime IS high AND cost IS medium AND energy IS low AND availability IS low) OR (respTime IS high AND cost IS low AND energy IS high AND availability IS medium) OR (respTime IS high AND cost IS low AND energy IS medium AND availability IS medium) OR (respTime IS high AND cost IS low AND energy IS medium AND availability IS low) OR (respTime IS medium AND cost IS high AND energy IS high AND availability IS high) OR (respTime IS medium AND cost IS high AND energy IS medium AND availability IS high) OR (respTime IS medium AND cost IS high AND energy IS low AND availability IS medium) OR (respTime IS medium AND cost IS high AND energy IS low AND availability IS low) OR (respTime IS medium AND cost IS medium AND energy IS high AND availability IS high) OR (respTime IS medium AND cost IS medium AND energy IS low AND availability IS high) OR (respTime IS medium AND cost IS medium AND energy IS low AND availability IS low) OR (respTime IS medium AND cost IS low AND energy IS high AND availability IS medium) OR (respTime IS medium AND cost IS low AND energy IS high AND availability IS low) OR (respTime IS medium AND cost IS low AND energy IS medium AND availability IS high) OR (respTime IS medium AND cost IS low AND energy IS medium AND availability IS low) OR (respTime IS medium AND cost IS low AND energy IS low AND availability IS medium) OR (respTime IS low AND cost IS high AND energy IS high AND</p>

	<p>availability IS medium) OR (respTime IS low AND cost IS high AND energy IS medium AND availability IS medium) OR (respTime IS low AND cost IS high AND energy IS medium AND availability IS low) OR (respTime IS low AND cost IS medium AND energy IS high AND availability IS medium) OR (respTime IS low AND cost IS medium AND energy IS high AND availability IS low) OR (respTime IS low AND cost IS medium AND energy IS medium AND availability IS high) OR (respTime IS low AND cost IS medium AND energy IS medium AND availability IS low) OR (respTime IS low AND cost IS medium AND energy IS low AND availability IS medium) OR (respTime IS low AND cost IS low AND energy IS medium AND availability IS medium) THEN BoA IS medium</p>
4	<p>IF (respTime IS high AND cost IS high AND energy IS high AND availability IS high) OR (respTime IS high AND cost IS high AND energy IS high AND availability IS medium) OR (respTime IS high AND cost IS high AND energy IS high AND availability IS low) OR (respTime IS high AND cost IS high AND energy IS medium AND availability IS medium) OR (respTime IS high AND cost IS high AND energy IS medium AND availability IS low) OR (respTime IS high AND cost IS high AND energy IS low AND availability IS low) OR (respTime IS high AND cost IS medium AND energy IS high AND availability IS medium) OR (respTime IS high AND cost IS medium AND energy IS high AND availability IS low) OR (respTime IS high AND cost IS medium AND energy IS medium AND availability IS medium) OR (respTime IS high AND cost IS medium AND energy IS medium AND availability IS low) OR (respTime IS high AND cost IS low AND energy IS high AND availability IS low) OR (respTime IS medium AND cost IS high AND energy IS high AND availability IS medium) OR (respTime IS medium AND cost IS high AND energy IS high AND availability IS low) OR (respTime IS medium AND cost IS high AND energy IS medium AND availability IS medium) OR (respTime IS medium AND cost IS high AND energy IS medium AND availability IS low) OR (respTime IS medium AND cost IS medium AND energy IS high AND availability IS medium) OR (respTime IS medium AND cost IS medium AND energy IS high AND availability IS low)</p>

	<p>availability IS low) OR (respTime IS medium AND cost IS medium AND energy IS medium AND availability IS high) OR (respTime IS medium AND cost IS medium AND energy IS medium AND availability IS medium) OR (respTime IS medium AND cost IS medium AND energy IS medium AND availability IS low) OR (respTime IS medium AND cost IS medium AND energy IS low AND availability IS medium) OR (respTime IS medium AND cost IS low AND energy IS medium AND availability IS medium) OR (respTime IS medium AND cost IS low AND energy IS low AND availability IS high) OR (respTime IS low AND cost IS high AND energy IS high AND availability IS low) OR (respTime IS low AND cost IS medium AND energy IS medium AND availability IS medium) OR (respTime IS low AND cost IS medium AND energy IS low AND availability IS high) OR (respTime IS low AND cost IS low AND energy IS medium AND availability IS high) OR (respTime IS low AND cost IS low AND energy IS low AND availability IS high) OR (respTime IS low AND cost IS low AND energy IS low AND availability IS medium) THEN BoA IS low</p>
--	---

References

- [1] W3C Working Group. (2004, Dec. 2013). *Web Services Architecture*. Available: <http://www.w3.org/TR/ws-arch/>
- [2] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *International Journal on Web and Grid Services*, vol. 1, pp. 1–30, Aug. 2005.
- [3] D. Ardagna and R. Mirandola, "Per-Flow Optimal Service Selection for Web Services Based Processes," *Journal of Systems and Software*, vol. 83, pp. 1512-1523, Aug. 2010.
- [4] W3C Working Group. (2003, Dec. 2013). *QoS for Web Services: Requirements and Possible Approaches*. Available: <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>
- [5] B. Cheng, *et al.*, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Engineering for Self-Adaptive Systems*, vol. 5525, pp. 1-26 2009.
- [6] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*: Prentice Hall, 2006.
- [7] M. Mansukhani, "Service Oriented Architecture White Paper," Hewlett-Packard Development Company, June 2005.
- [8] R. Welke, *et al.*, "Service-Oriented Architecture Maturity," *IEEE Computer*, vol. 44, pp. 61-67, Feb. 2011.
- [9] M. P. Papazoglou and J.-j. Dubray, "A Survey of Web Service Technologies," University of Trento, Trento, Italy, Technical Report DIT-04-058, 2004.
- [10] S. Tyagi. (2006, Dec. 2013). *RESTful Web Services*. Available: <http://www.oracle.com/technetwork/articles/javase/index-137171.html>
- [11] B. Sotomayor. (2005, Aug. 2013). *The Globus Toolkit 4 Programmer's Tutorial*. Available: <http://gdp.globus.org/qt4-tutorial/multiplehtml/index.html>
- [12] A. Alamri, *et al.*, "Classification of the State-of-the-Art Dynamic Web Services Composition Techniques," *International Journal of Web and Grid Services*, vol. 2, pp. 148–166, Sept. 2006.
- [13] R. Vaculin and K. Sycara, "Semantic Web Services Monitoring: An OWL-S based Approach," presented at the Hawaii International Conference on System Sciences (HICSS'08), Hawaii, USA, 2008.
- [14] F. Mustafa and T. L. McCluskey, "Dynamic Web Services Composition," presented at the International Conference on Computer Engineering and Technology (ICCET'09), Singapore, 2009.
- [15] OASIS. (2013, Dec. 2013). *Advancing Open Standards for the Information Society*. Available: <https://www.oasis-open.org/>
- [16] W3C. (2013, Dec. 2013). *World Wide Web Consortium*. Available: <http://www.w3.org/>
- [17] WS-I. (2009, Dec. 2013). *Web Services Interoperability Organization*. Available: <http://www.ws-i.org/>
- [18] Internet Society. (n.d., Dec. 2013). *The Internet Engineering Task Force*. Available: <http://www.ietf.org/>
- [19] W3C. (2008, Dec. 2013). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Available: <http://www.w3.org/TR/2008/REC-xml-20081126/>
- [20] W3C. (2001, Dec. 2013). *Web Services Description Language (WSDL) 1.1*. Available: <http://www.w3.org/TR/wsdl>
- [21] A. Tsalgatidou and T. Pilioura, "An Overview of Standards and Related Technology in Web Services," *Journal of Distributed and Parallel Databases*, vol. 12, pp. 135-162, Sept./Nov. 2002.

-
- [22] W3C. (2007, Dec. 2013). *SOAP Version 1.2 Part 0: Primer (Second Edition)*. Available: <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>
- [23] OASIS. (2004, Dec. 2013). *UDDI Version 3.0.2*. Available: http://www.uddi.org/pubs/uddi_v3.htm#_Toc85907967
- [24] innoQ. (1999-2008, Aug. 2013). *Web Services Standards Overview*. Available: <http://www.innoq.com/resources/ws-standards-poster/>
- [25] C. Pautasso, et al., "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision," in *Proceedings of the 17th International World Wide Web Conference (WWW'08)*, Beijing, China, 2008, pp. 805-814.
- [26] S. Zhaohao, et al., "A Demand Driven Web Service Lifecycle," in *Proceedings of the International Conference on New Trends in Information and Service Science (NISS '09)*, Beijing, China, 2009, pp. 8-14.
- [27] L.-J. Zhang and M. Jeckle, "The Next Big Thing: Web Services Collaboration," in *Proceedings of the International Conference on Web Services - Europe (ICWS'03)*, Erfurt, Germany, 2003, pp. 1-10.
- [28] W. Ren and Z. Xu, "A New Web Service Discovery Method Based on Semantic," presented at the Workshop on Power Electronics and Intelligent Transportation System (PEITS'08), Guangzhou, China, 2008.
- [29] G. Ye, et al., "A QoS-Aware Model for Web Services Discovery," in *Proceedings of the First International Workshop on Education Technology and Computer Science (ETCS'09)*, Wuhan, China, 2009, pp. 740-744.
- [30] A. Erradi, et al., "WS-Policy Based Monitoring of Composite Web Services," in *Proceedings of the 5th European Conference on Web Services (ECOWS '07)*, Halle, Germany, 2007, pp. 99-108.
- [31] Microsoft. (2013, Dec. 2013). *Web Service Benefits*. Available: <http://msdn.microsoft.com/en-us/library/cc508708.aspx>
- [32] M. Eid, et al., "A Reference Model for Dynamic Web Service Composition Systems," *International Journal of Web and Grid Services*, vol. 4, pp. 149–168, Jun. 2008.
- [33] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, pp. 46-52, Oct. 2003.
- [34] Q. Yu, et al., "Deploying and Managing Web Services: Issues, Solutions, and Directions," *International Journal on Very Large Data Bases*, vol. 17, pp. 537-572, May 2008.
- [35] myGrid. (2009, Dec. 2013). *Taverna Workflow Management System*. Available: <http://www.taverna.org.uk/>
- [36] Kepler. (n.d., Dec. 2013). *The Kepler Project*. Available: <https://kepler-project.org/>
- [37] Active Endpoints Inc. (2004–2006, July 2010). *ActiveBPEL Designer and Eclipse Web Tools Project*. Available: http://www.activebpel.org/samples/samples-3/eclipseWTP_and_BPEL/doc/index.html
- [38] Oracle Corporation. (2010, Dec. 2013). *Oracle JDeveloper*. Available: <http://www.oracle.com/technology/products/jdev/index.html>
- [39] IBM. (n.d., Dec. 2013). *IBM Software - WebSphere*. Available: <http://www-01.ibm.com/software/websphere/>
- [40] Geeknet Inc. (2010, Dec. 2013). *Pi Calculus for SOA*. Available: <http://sourceforge.net/projects/pi4soa/>
- [41] B. Orriëns, et al., "Model Driven Service Composition," in *Proceedings of the First International Conference on Service Oriented Computing (ICSOC'03)*, Trento, Italy, 2003, pp. 75-90.
- [42] B. Benatallah, et al., "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," in *Proceedings of 18th International Conference on Data Engineering*, San Jose, USA, 2002, pp. 297-308.
-

-
- [43] D. VanderMeer, et al., "FUSION: a System Allowing Dynamic Web Service Composition and Automatic Execution," in *Proceedings of the IEEE International Conference on E-Commerce (CEC'03)*, Newport Beach, USA, 2003, pp. 399-404.
- [44] J. L. Ambite, et al., "Argos: Dynamic Composition of Web Services for Goods Movement Analysis and Planning," in *Proceedings of the 2006 International Conference on Digital Government Research*, San Diego, USA, 2006, pp. 319-320
- [45] F. Casati, et al., "Adaptive and Dynamic Service Composition in eFlow," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, Stockholm, Sweden, 2000, pp. 13-31.
- [46] K. Fujii and T. Suda, "Dynamic Service Composition Using Semantic Information," in *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, USA, 2004, pp. 39-48.
- [47] D. Wu, et al., "Automating DAML-S Web Services Composition Using SHOP2," in *Proceedings of the 2nd International Semantic Web Conference, Lecture Notes in Computer Science*, 2003, pp. 195-210.
- [48] T. Weise, et al., "Different Approaches to Semantic Web Service Composition," presented at the Third International Conference on Internet and Web Applications and Services (ICIW'08), Athens, Greece, 2008.
- [49] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *Semantic Web Services and Web Process Composition, Lecture Notes in Computer Science*, vol. 3387, pp. 43-54, 2005.
- [50] OASIS. (2007, Dec. 2013). *Web Services Business Process Execution Language Version 2.0*. Available: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [51] W3C. (2004, Dec. 2013). *Web Services Choreography Description Language Version 1.0*. Available: <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>
- [52] W3C. (2002, Dec. 2013). *Web Service Choreography Interface (WSCI) 1.0*. Available: <http://www.w3.org/TR/wsci/>
- [53] R. Cover. (2002, Dec. 2013). *Cover Pages: Web Services Flow Language (WSFL)*. Available: <http://xml.coverpages.org/wsfl.html>
- [54] W3C. (2004, Dec. 2013). *OWL-S: Semantic Markup for Web Services*. Available: <http://www.w3.org/Submission/OWL-S/>
- [55] M. P. Papazoglou, et al., "Service-Oriented Computing: A Research Roadmap," *International Journal of Cooperative Information Systems*, vol. 17, pp. 223-255, 2008.
- [56] N. Kokash and V. D'Andrea, "Evaluating Quality of Web Services: A Risk-Driven Approach," *Business Information Systems*, vol. 4439, pp. 180-194, 2007.
- [57] T. Hong-Linh, et al., "Towards a Framework for Monitoring and Analyzing QoS Metrics of Grid Services," in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (e-Science '06)*, Amsterdam, Netherlands, 2006, pp. 65-65.
- [58] R. Sumra and A. D. (2003, Dec. 2013). *Quality of Service for Web Services - Demystification, Limitations, and Best Practices*. Available: <http://www.developer.com/services/article.php/2027911/Quality-of-Service-for-Web-ServicesmdashDemystification-Limitations-and-Best-Practices.htm>
- [59] Y. Dai, et al., "QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction," *Journal of Computer Science and Technology*, vol. 24, pp. 250-261, Mar. 2009.
- [60] S. Kalepu, et al., "Verity: a QoS Metric for Selecting Web Services and Providers," in *Proceedings of the Fourth International Conference on Web*
-

- Information Systems Engineering Workshops (WISEW'03)*, 2003, pp. 131-139.
- [61] D. Verma, *Supporting Service Level Agreements on IP Networks*: Sams, 1999.
- [62] L. Zeng, *et al.*, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, pp. 311-327, 2004.
- [63] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, pp. 57-81, 2003.
- [64] Open Grid Forum, "Web Services Agreement Specification (WS-Agreement)," Open Grid Forum, GFD-R-P.107, 2007.
- [65] J. Cardoso, *et al.*, "Quality of Service for Workflows and Web Service Processes," *Journal of Web Semantics*, vol. 1, pp. 281-308, 2004.
- [66] S.-Y. Hwang, *et al.*, "A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows," *International Journal of Information Sciences*, vol. 177, pp. 5484-5503, Dec. 2007.
- [67] G. Canfora, *et al.*, "QoS-Aware Replanning of Composite Web Services," in *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS'05)*, Orlando, USA, 2005, pp. 121-129.
- [68] A. Huang, *et al.*, "An Optimal QoS-based Web Service Selection Scheme," *Information Sciences*, vol. 179, pp. 3309-3322, 2009.
- [69] F. Casati, *et al.*, "Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection," in *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, USA, 2004, pp. 316-321.
- [70] Q. Yu, *et al.*, "A Two-Phase Framework for Quality-Aware Web Service Selection," *Service Oriented Computing and Applications*, vol. 4, pp. 63-79, 2010.
- [71] W. Dong and L. Jiao, "QoS-Aware Web Service Composition Based on SLA," in *Proceedings of the 2008 Fourth International Conference on Natural Computation (ICNC'08)*, Jinan, China, 2008, pp. 247-251.
- [72] B. Cavallo, *et al.*, "An Empirical Comparison of Methods to Support QoS-Aware Service Selection," in *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS'10)*, Cape Town, South Africa, 2010, pp. 64-70.
- [73] A. D'Ambrogio and P. Bocciarelli, "A Model-Driven Approach to Describe and Predict the Performance of Composite Services," in *Proceedings of the 6th International Workshop on Software and Performance (WOSP'07)*, Buenos Aires, Argentina, 2007, pp. 78-89.
- [74] A. Nasridinov, *et al.*, "A QoS-Aware Performance Prediction for Self-Healing Web Service Composition," in *Proceedings of the Second International Conference on Cloud and Green Computing (CGC'12)*, Xiangtan, China, 2012, pp. 799-803.
- [75] A. Goldman and Y. Ngoko, "On Graph Reduction for QoS Prediction of Very Large Web Service Compositions," in *Proceedings of the Ninth International Conference on Services Computing*, Honolulu, USA, 2012, pp. 258-265.
- [76] L. Jing, *et al.*, "Towards Adaptive Web Services QoS Prediction," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'10)*, Perth, Australia, 2010, pp. 1-8.
- [77] R. Vilalta, *et al.*, "Predictive Algorithms in the Management of Computer Systems," *IBM Systems Journal*, vol. 41, pp. 461-474, Jul. 2002.
- [78] N. Dushay, *et al.*, "Predicting Indexer Performance in a Distributed Digital Library," in *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries*, 1999, pp. 142-166.
- [79] G. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco, USA, 1970.

-
- [80] S. Makridakis, *et al.*, *Forecasting: Methods and Applications*: John Wiley & Sons, 1998.
- [81] F. Zulkernine, *et al.*, "An Autonomic Web Services Environment using a Reflective and Database-Oriented Approach," *Ubiquitous Computing and Communication Journal: Special Issue on Autonomic Computing Systems and Applications*, pp. 33-44, 2008.
- [82] M. Salehie and L. Tahvildari, "Self-adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, pp. 1-42, 2009.
- [83] IBM, "An Architectural Blueprint for Autonomic Computing," White Paper. June, 2005.
- [84] S. S. Laster and A. O. Olatunji, "Autonomic Computing: Towards a Self-Healing System," in *Proceedings of the Spring 2007 American Society for Engineering Education Conference*, Illinois, USA, 2007.
- [85] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, pp. 41-50, Jan. 2003.
- [86] V. Cardellini, *et al.*, "MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 38, pp. 1138-1159, Sept./Oct. 2012.
- [87] R. Calinescu, *et al.*, "Dynamic QoS Management and Optimization in Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 37, pp. 387-409, May/Jun. 2011.
- [88] L. Baresi, *et al.*, "Towards Self-healing Composition," *Studies in Computational Intelligence*, vol. 42, pp. 27-46, 2007.
- [89] G. Huipeng, *et al.*, "ANGEL: Optimal Configuration for High Available Service Composition," in *Proceedings of the International Conference on Web Services (ICWS'07)*, Salt Lake City, USA, 2007, pp. 280-287.
- [90] L. Wenjuan, *et al.*, "A Framework to Improve Adaptability in Web Service Composition," in *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET'10)*, Chengdu, China, 2010, pp. V1-616 - V1-621.
- [91] A. Erradi, *et al.*, "Policy-Driven Middleware for Self-Adaptation of Web Services Compositions," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (Middleware'06)*, Melbourne, Australia, 2006, pp. 62-80.
- [92] D. Bianculli, *et al.*, "Automated Dynamic Maintenance of Composite Services Based on Service Reputation," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, Vienna, Austria, 2007, pp. 449-455.
- [93] S. Dustdar, *et al.*, "A Roadmap Towards Sustainable Self-Aware Service Systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*, Cape Town, South Africa, 2010, pp. 10-19.
- [94] WS-Diamond Team, "WS-DIAMOND: Web Services- DiAgnosability, MONitoring and Diagnosis," *MIT press*, pp. 213-239, 2009.
- [95] D. Ardagna, *et al.*, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, pp. 39-46, Nov./Dec. 2007.
- [96] A. C. Huang and P. Steenkiste, "Building Self-Configuring Services Using Service-Specific Knowledge," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, Honolulu, USA, 2004, pp. 45-54.
- [97] S. Kounev, *et al.*, "Self-Aware QoS Management in Virtualized Infrastructures," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*, Karlsruhe, Germany, 2011, pp. 175-176.
-

-
- [98] J. Hielscher, *et al.*, "A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing," in *Proceedings of the 1st European Conference Service Wave*, Madrid, Spain, 2008, pp. 122-133.
- [99] R. Aschoff and A. Zisman, "Proactive Adaptation of Service Composition," in *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12)*, Zürich, Switzerland, 2012, pp. 1-10.
- [100] A. Metzger, "Towards Accurate Failure Prediction for the Proactive Adaptation of Service-Oriented Systems," in *Proceedings of the 8th Workshop on Assurances for Self-adaptive Systems (ASAS'11)*, Szeged, Hungary, 2011, pp. 18-23.
- [101] P. Châtel, *et al.*, "QoS-based Late-Binding of Service Invocations in Adaptive Business Processes," in *Proceedings of the International Conference on Web Services (ICWS'10)*, Miami, USA, 2010, pp. 227-234.
- [102] Q. Mahmoud. (2004, Dec. 2013). *Developing Web Services with Java 2 Platform, Enterprise Edition (J2EE) 1.4 Platform*. Available: <http://www.oracle.com/technetwork/articles/javaee/j2ee-ws-140408.html>
- [103] A. Sargeant, *et al.*, "Testing the Effectiveness of Dynamic Binding in Web Services," in *Proceedings of the 10th International Conference on Computer and Information Technology (CIT'10)*, Bradford, UK, 2010, pp. 1263-1268.
- [104] C. Pautasso and G. Alonso, "Flexible Binding for Reusable Composition of Web Services," in *Proceedings of the 4th International Conference on Software Composition (SC'05)*, Edinburgh, UK, 2005, pp. 151-166.
- [105] M. D. Penta, *et al.*, "WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services," in *Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering (SOSE'06)*, Shanghai, China, 2006, pp. 74-80.
- [106] U. Küster and B. König-Ries, "Dynamic Binding for BPEL Processes - A Lightweight Approach to Integrate Semantics into Web Services," in *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC'06)*, Chicago, USA, 2006, pp. 116-127.
- [107] G. Canfora, *et al.*, "A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services," *Journal of Systems and Software*, vol. 81, pp. 1754-1769, Oct. 2008.
- [108] A. Erradi and P. Maheshwari, "Dynamic Binding Framework for Adaptive Web Services," in *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services (ICIW'08)*, Athens, Greece, 2008, pp. 162-167.
- [109] N. Looker, *et al.*, "Determining the Dependability of Service-Oriented Architectures," *International Journal of Simulation and Process Modelling*, vol. 3, pp. 88-97 2007.
- [110] S. Bruning, *et al.*, "A Fault Taxonomy for Service-Oriented Architecture," in *Proceedings of the 10th High Assurance Systems Engineering Symposium (HASE'07)*, Plano, TX, USA, 2007, pp. 367-368.
- [111] K. S. Chan, *et al.*, "A Fault Taxonomy for Web Service Composition," in *Proceedings of the 5th International Conference on Service Oriented Computing - Workshops (ICSOC'07)*, Vienna, Austria, 2009, pp. 363-375.
- [112] S. S. Sathya and K. S. Babu, "Survey of Fault Tolerant Techniques for Grid," *Computer Science Review*, vol. 4, pp. 101-120, May 2010.
- [113] J. Yu and R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *Special Interest Group on Management of Data*, vol. 34, pp. 44-49, Sept. 2005.
- [114] L. Liu, *et al.*, "A Fault-Tolerant Framework for Web Services," in *Proceedings of the WRI World Congress on Software Engineering (WCSE'09)*, Xiamen, China, 2009, pp. 138-142.
-

-
- [115] C.-L. Fang, *et al.*, "Fault Tolerant Web Services," *Journal of Systems Architecture*, vol. 53, pp. 21-38, Jan. 2007.
- [116] J. Lau, *et al.*, "Designing Fault Tolerant Web Services Using BPEL," in *Proceedings of the 7th International Conference on Computer and Information Science (ICIS'08)*, Portland, USA, 2008, pp. 618-623.
- [117] N. Laranjeiro and M. Vieira, "Towards Fault Tolerance in Web Services Compositions," in *Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems (EFTS'07)*, Dubrovnik, Croatia, 2007.
- [118] M. J. Druzdzel and R. R. Flynn, "Decision Support Systems," in *Encyclopedia of Library and Information Science*, Second Edition ed, 2002.
- [119] I. Ben-Gal, "Bayesian Networks," in *Encyclopedia of Statistics in Quality and Reliability*, ed: John Wiley & Sons, Ltd, 2008.
- [120] T. M. Mitchell, *Machine Learning*: McGraw-Hill, 1997.
- [121] P. Geurts, *et al.*, "A Machine Learning Approach to Improve Congestion Control Over Wireless Computer Networks," in *Proceedings of the 4th International Conference on Data Mining (ICDM'04)*, Brighton, UK, 2004, pp. 383-386.
- [122] M. Maggio, *et al.*, "Decision Making in Autonomic Computing Systems: Comparison of Approaches and Techniques," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*, Karlsruhe, Germany, 2011, pp. 201-204.
- [123] A. J. Ramirez, *et al.*, "Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems," in *Proceedings of the 6th International Conference on Autonomic Computing (ICAC'09)*, Barcelona, Spain, 2009, pp. 97-106.
- [124] C. Szepesvári, *Algorithms for Reinforcement Learning*: Morgan & Claypool Publishers, 2010.
- [125] L. A. Zadeh, "The Role of Fuzzy Logic in Modeling, Identification and Control," *Modeling, Identification and Control*, vol. 15, pp. 191-203, 1994.
- [126] A. Aamodt and E. Plaza, "Case Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Journal of AI Communications*, vol. 7, pp. 39-59, 1994.
- [127] W3C Working Group. (2004, Dec. 2013). *Web Services Architecture Usage Scenarios*. Available: <http://www.w3.org/TR/ws-arch-scenarios/>
- [128] G. Wu, *et al.*, "Towards Self-Healing Web Services Composition," in *Proceedings of the First Asia-Pacific Symposium on Internetware (Internetware'09)*, Beijing, China, 2009.
- [129] D. Ardagna, *et al.*, "A Service-Based Framework for Flexible Business Processes," *IEEE Software*, vol. 28, pp. 61-67, Mar./Apr. 2011.
- [130] C. Liu and D. Liu, "QoS-Oriented Web Service Framework by Mixed Programming Techniques," *Journal of Computers*, vol. 8, pp. 1763-1770, July 2013.
- [131] M. Sedaghat, *et al.*, "Unifying Cloud Management: Towards Overall Governance of Business Level Objectives," in *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'11)*, Newport Beach, USA, 2011, pp. 591-597.
- [132] Y. Ying, *et al.*, "A Self-Healing Composite Web Service Model," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC'09)*, Biopolis, Singapore, 2009, pp. 307-312.
- [133] The Apache Software Foundation. (n.d., Dec. 2013). *Apache ODE*. Available: <http://ode.apache.org/>
- [134] The Eclipse Foundation. (2013, Dec. 2013). *BPEL Designer Project*. Available: <http://www.eclipse.org/bpel/>
-

-
- [135] Active Endpoints Inc. (2010, May 2013). *BPEL Open Source Engine - The ActiveBPEL Engine*. Available: <http://www.activevos.com/community-open-source.php>
- [136] JBoss Community. (n.d., Dec. 2013). *JBoss Application Server*. Available: <http://jboss.org/jbossas>
- [137] JOpera.org. (2011, Dec. 2013). *JOpera for Eclipse*. Available: <http://www.iopera.org/>
- [138] The Apache Software Foundation. (2004-2013, Dec. 2013). *jUDDI*. Available: <http://juddi.apache.org/>
- [139] G. Jung, *et al.*, "A Cost-Sensitive Adaptation Engine for Server Consolidation of Multitier Applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware'09)*, Urbana Champaign, USA, 2009, pp. 163-183.
- [140] M. Salehie, *et al.*, "Change Support in Adaptive Software: A Case Study for Fine-Grained Adaptation," in *Proceedings of the 6th IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASE'09)*, San Francisco, USA, 2009, pp. 35-44.
- [141] L. Ai, "QoS-Aware Web Service Composition Using Genetic Algorithms," PhD thesis, Faculty of Science and Technology, Queensland University of Technology, Brisbane, Australia, 2011.
- [142] Y. Vanrompay, *et al.*, "Genetic Algorithm-Based Optimization of Service Composition and Deployment," in *Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments (SIPE'08)*, Sorrento, Italy, 2008, pp. 13-18.
- [143] L. Cao, *et al.*, "Genetic Algorithm Utilized in Cost-Reduction Driven Web Service Selection," in *Proceedings of the International Conference on Computational Intelligence and Security (CIS'05)*, Xi'an, China, 2005, pp. 679-686.
- [144] T. Senivongse and N. Wongsawangpanich, "Composing Services of Different Granularity and Varying QoS Using Genetic Algorithm," in *Proceedings of the World Congress on Engineering and Computer Science (WCECS'11)*, San Francisco, USA, 2011, pp. 388-393.
- [145] H. Wang, *et al.*, "Adaptive Service Composition Based on Reinforcement Learning," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'10)*, San Francisco, USA, 2010, pp. 92-107.
- [146] J. Zhou and S. Chen, "Reinforcement Learning Based Web Service Compositions for Mobile Business," in *Proceedings of the International Conference on Web Information Systems and Mining (WISM'09)*, Shanghai, China, 2009, pp. 527-534.
- [147] W. Hongbing, *et al.*, "RLPLA: A Reinforcement Learning Algorithm of Web Service Composition with Preference Consideration," in *Proceedings of the IEEE Congress on Services Part II (SERVICES-2)*, Beijing, China, 2008, pp. 163-170.
- [148] B. Wetzstein, *et al.*, "Preventing KPI Violations in Business Processes based on Decision Tree Learning and Proactive Runtime Adaptation," *Journal of Systems Integration*, vol. 3, pp. 3-18, 2012.
- [149] B. Pernici and S. H. Siadat, "Selection of Service Adaptation Strategies Based on Fuzzy Logic," in *Proceedings of the IEEE World Congress on Services (SERVICES'11)*, Washington DC, USA, 2011, pp. 99-106.
- [150] P. Wang, *et al.*, "A Fuzzy Model for Selection of QoS-Aware Web Services," in *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE'06)*, Shanghai, China, 2006, pp. 585-593.
- [151] H. Pfeffer, *et al.*, "A Fuzzy Logic Based Model for Representing and Evaluating Service Composition Properties," in *Proceedings of the 3rd*
-

-
- International Conference on Systems and Networks Communications (ICSNC'08)*, Sliema, Malta, 2008, pp. 335-342.
- [152] R. Torres, *et al.*, "Self-Adaptive Fuzzy QoS-Driven Web Service Discovery," in *Proceedings of the IEEE International Conference on Services Computing (SCC'11)*, Washington DC, USA, 2011, pp. 64-71.
- [153] R. Buyya, *et al.*, "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges," in *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'10)*, Las Vegas, USA, 2010.
- [154] J. Kaplan, *et al.*, "Revolutionizing Data Center Energy Efficiency," McKinsey, July 2009.
- [155] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353, 1965.
- [156] Li-Xin Wang, *A Course in Fuzzy Systems and Control*: Prentice Hall, 1997.
- [157] P. Cingolani. (n.d., Dec. 2013). *jFuzzyLogic*. Available: <http://jfuzzylogic.sourceforge.net/html/index.html>
- [158] Energy Star, "Computer Servers Product List - Families," August, 2012.
- [159] T. Baker, *et al.*, "Towards Autonomic Cloud Services Engineering via Intention Workflow Model," in *Proceedings of the 10th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON'13)*, Zaragoza, Spain, 2013, pp. 212-227.
- [160] D. Tosi, *et al.*, "Towards Autonomic Service-Oriented Applications," *International Journal of Autonomic Computing*, vol. 1, pp. 58-80, 2009.
- [161] P. Leitner, *et al.*, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'10)*, Miami, USA, 2010, pp. 369-376.
- [162] Z. Yuelong, *et al.*, "Predicting Failures in Dynamic Composite Services with Proactive Monitoring Technique," in *Proceedings of the IEEE Eighth World Congress on Services (SERVICES'12)*, Honolulu, USA, 2012, pp. 92-99.
- [163] O. Sammodi, *et al.*, "Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications," in *Proceedings of the IEEE 35th Annual Computer Software and Applications Conference (COMPSAC'11)*, Munich, Germany, 2011, pp. 582-587.
- [164] R. Aschoff and A. Zisman, "QoS-Driven Proactive Adaptation of Service Composition," in *Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC'11)*, Paphos, Cyprus, 2011, pp. 421-435.
- [165] S. Sharaf and K. Djemame, "Enabling Service Level Agreement Renegotiation Through Extending WS-Agreement Specification," *Journal on Service-Oriented Computing and Applications (to appear)*, 2014.