

UNIVERSITAT POLITÈCNICA DE CATALUNYA  
DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMÀTICS  
MÀSTER EN COMPUTACIÓ

TESI DE MÀSTER

# Quality of Service (QoS) in SOA Systems. *A Systematic Review*

ESTUDIANT: **Marc Oriol Hilari**  
DIRECTOR(S): **Xavier Franch, Jordi Marco**  
PONENT: -

DATA: **06/09/2009**

# Index

---

## **Introduction**

1	Objective of this work. ....	4
1.1	Development of a review .....	4
1.2	Development of a monitoring tool.....	5
2	Introduction to SOA Systems .....	6
3	Research in SOA .....	8
3.1	Timeline of the research in SOA.....	8
3.2	Current Research in SOA .....	9
3.2.1	European Research Platforms, Networks & Projects:.....	9
3.3	Conferences and Journals: .....	10
3.3.1	SOA specific conferences: .....	10
3.3.2	Other conferences that include SOA:.....	11
3.3.3	Journals: .....	12

## **Systematic review definition**

4	Research method for the systematic review. ....	13
4.1	Motivation .....	13
4.2	The review Process.....	13
5	Planning the review.....	14
5.1	Identification of the need for a review .....	14
5.2	Development of a Review Protocol.....	15
5.2.1	The research question .....	15
5.2.2	The search strategy .....	16
5.3	Study selection criteria.....	19

## **Systematic review results**

6	Introduction.....	20
7	Quality Models for web services .....	21
8	Typology of Quality attributes and metrics.....	25
9	Ontologies for Quality Of Service .....	26
10	Describing web services with QoS.....	34
10.1	WSDL extension with QoS .....	34
10.2	UDDI extension with QoS .....	38
11	Using a broker with QoS information.....	42
12	Trustworthiness of claimed QoS .....	44
13	Monitoring.....	47
13.1	Passive Monitoring.....	47

13.2	Testing .....	52
13.3	Simulation .....	54
14	Non-Functional requirements for web service selection .....	55
15	Ranking web services with QoS for web service Selection .....	58
15.1	Ranking algorithms .....	58
15.2	Ranking through Fuzzy requirements .....	61
16	QoS in composite web services .....	62
16.1	Brief history of Service Composition Languages .....	62
16.2	QoS aggregation. ....	64
17	Optimizing composite web services.....	68
18	Conclusions of the review .....	70

### **Monitoring tool**

19	Introduction.....	72
19.1	Self-Healing SOA System (MAESoS) .....	72
19.2	Web Service discovery with QoS information (WeSSQoS).....	73
20	Requirements .....	74
20.1	Functional requirements .....	74
20.2	Non functional requirements .....	75
21	Attributes and metrics .....	77
22	Platform Architecture.....	79
22.1	Monitor .....	80
22.2	Analyzer .....	81
22.3	Decision Maker.....	81
22.4	Users.....	81
23	Use Cases.....	82
24	Sequence diagrams .....	83
25	Services Interface specification .....	90
25.1	Initial Specification .....	90
25.2	Current Specification .....	91
25.3	Analyzer service interface .....	93
25.4	Decision Maker interface .....	93
25.5	Monitor interface .....	94
26	Some implementation details .....	95
26.1	Storage .....	95
26.2	Multithreading .....	96
27	Future Work of SALMon.....	96
28	References.....	97

# Introduction

## 1 Objective of this work.

---

In the last recent years a new technology called Web Services has emerged. The main characteristic of a web service is that it is a piece of software that the user can utilize but doesn't own, that is, the user doesn't install the software but uses it through the internet and standard protocols.

With this new technology, a new architecture paradigm called SOA (Service Oriented Architecture) has appeared. This architecture is based on combining several web services, each one responsible to develop a concrete task, in order to obtain full-operational software.

The web services that compose a SOA System might be able to perform a task in a certain time, might be unavailable in some cases, might have security policies, etc. All this attributes, named Quality attributes, are essential in order to choose the appropriate web service for a SOA System.

The objective of this Master Thesis is focused on two different but related subjects: (1) The development of a review regarding to the Quality Attributes for web services in a systematic manner and (2) the development of a tool for monitoring SOA Systems capable to be used in several frameworks such as for Self-Adaptive SOA Systems and for Web Service Discovery Systems.

### 1.1 Development of a review

The objective of the review is to identify and evaluate the most significant available research relevant to Quality Attributes for Web Services. Particularly, we focus on the following aspects:

- Quality Models for web services.
- Ontologies for web services with Quality of Service (QoS).
- Web services definitions with quality attributes.
- QoS Brokering architectures.
- Monitoring.
- Web Service Discovery and ranking with QoS.
- QoS aggregation in composite web services

We have to mention that Service Level Agreements (SLAs) are out of the scope of this project. This is because of time and space limitations, and we considered focusing deeply on these subtopics rather than explaining briefly each of a huge set of topics.

Based on the given research areas, our work will aim on these objectives:

- Summarize the existing work related to the given topics.
- Identify the gaps in current research in order to detect areas for further investigation
- Provide a framework/background in order to appropriately position new research activities.

## 1.2 Development of a monitoring tool

Apart from the review, this work also includes a more applied part, the development of a tool for (1) monitoring the QoS of a Service, (2) report the obtained Quality information and (3) report SLA violations.

The development of this tool is the continuation and materialization of a monitoring architecture initiated by David Ameller [1].

The initial conception of the development of this tool was to provide a service as part of a Framework for supporting self-adaptive SOA systems which aims to ensure the QoS required by the user. During the development of the Master Thesis, another goal emerged, to include the monitor in a Framework for Web Service selection, which aims at ranking Web Services accordingly to the QoS and the user's requirements.

Therefore, the final objectives of this tool are:

- 1) To provide QoS information on run-time to detect SLA violations for Self-adaptive SOA Systems.
- 2) To provide QoS information on run-time in order to give trustworthy quality information for a Service Ranking tool.

## 2 Introduction to SOA Systems

---

“Web services and e-services have been announced as the next wave of Internet-based business applications that will dramatically change the use of the Internet”. This is an eloquent sentence stated by Casati et al. [2] that may not be far from the real expectations of web services.

Before the existence of this technology, the use of services from the World Wide Web was through a browser and a Web server using the HTTP protocol. Nowadays, this new model of interaction in the web given by web services has involved a primary subject of study which has had several contributions in the last few years.

Several organizations have been putting a big effort for standardization of this technology. The current results of these efforts are the web services as we understand nowadays:

The main actors of this new model are web service providers and web service consumers. Web Service providers are the ones who develop the service and publish the Web Service Description Language (WSDL). A WSDL is an XML interface used to invoke the service in a programming-language independent manner; details of this standard are further explained in the following chapters. Service providers are also responsible to register their services in a public service registry called Universal Description, Discovery, and Integration (UDDI). This registry would be used afterwards for the clients to discover the web services which fulfill their requirements.

Once the client has discovered the web services in the registry and has obtained the URL for the WSDL file that describes the service, the client is able to invoke the web service using the XML-based Simple Object Access Protocol (SOAP). SOAP is a standard used to either send or receive messages between clients and providers of services.

In traditional web sites, the implementation of the whole business logic was done in the same system (See Figure 1): the client invokes an operation through the http protocol to the server. The server then carries out all the business Logic, querying if necessary the data information in a data source (such as a Database).

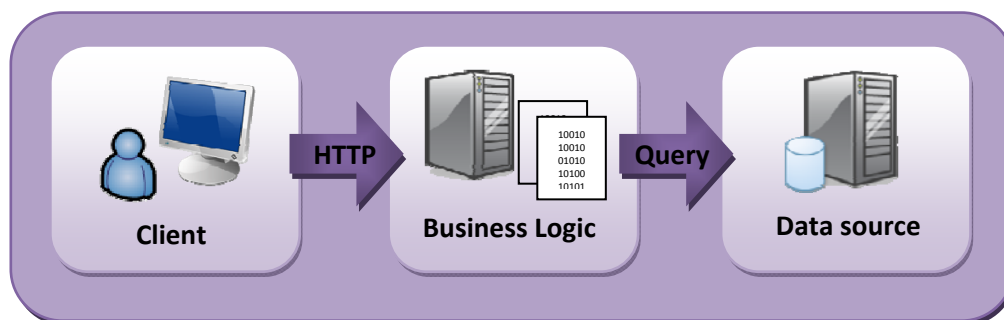


Figure 1: Traditional web invocation

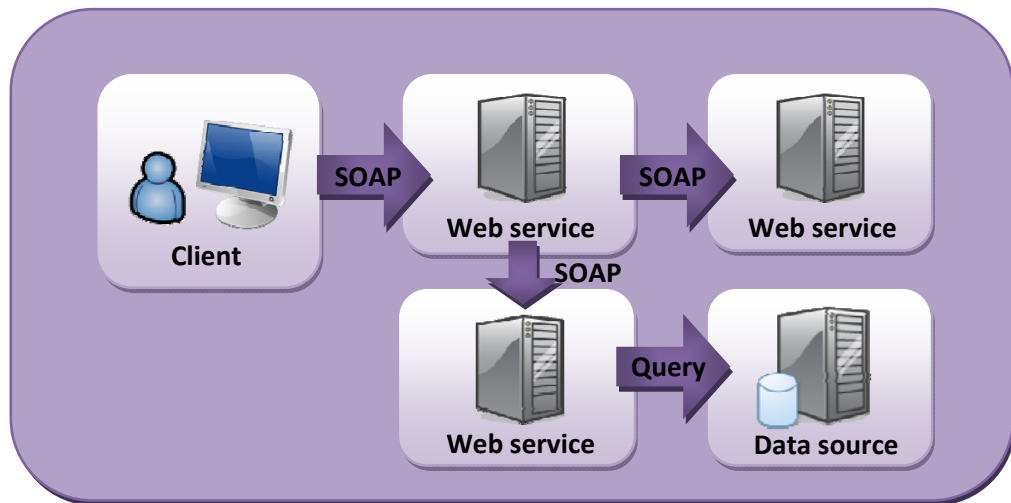


Figure 2: Web services interaction

However, the flexibility and extensibility of web services has involved a new interaction model and development approach for the construction of software. As we can see in Figure 2, the client (which might be software itself) sends a request to a web service. This web service is in turn the client of other web services. Interaction between them is done through the SOAP protocol. In this new scenario, each web service is responsible of doing a small task, so following the SOA principles, a web service can be developed by combining a set of small pieces of software.

## 3 Research in SOA

---

### 3.1 Timeline of the research in SOA

One can assume that SOA paradigm began with the appearance of the Distributed Computing, which consists of a set of computers that communicates each other through a network to achieve a common goal.

Distributed Computing became a branch for itself in Computer Science Research in the 70s. During this period, the appearance of the Remote Procedure Calls (RPC) in the 1976 started a new way of software development through remote invocations. RPC allowed a developer to call a subroutine placed in another address space (commonly on another computer) without the need for coding the details of the remote interaction.

In 1991, CORBA was standardized. CORBA is a technology that allows software components written in multiple computer languages to interact between each other.

By the other hand, in 1998, XML 1.0 became a W3C recommendation. XML is a meta-language that provides several benefits: simplicity, technological independence, human and computer readable, etc. A set of advantages that CORBA didn't fulfill at all. With the increasing use of XML in multiple areas, the idea to include this language as a basis for a new remote interaction technology emerged. Here started the born of the web services standards that are currently used nowadays.

SOAP, which stands for 'Simple Object Access Protocol', was designed in 1998. This is the standard used for web services requests and responses. In 2000 WSDL 1.0 was introduced by IBM, Microsoft and Ariba to describe Web Services for their SOAP toolkit. UDDI, the registry to allocate the different web services, was defined also in 2000. These are the current basic standards for web services.

Nevertheless, there's a lot of research and work to be done, from adding semantics to web services to handle transactions in Composite web service, adding QoS to the lifecycle of web service and so forth.



A good work summarizing the current research areas in SOA Systems was presented by Papazoglou et al. in [3], defining a Research Road Map of SOA Systems with the state of the art of each of these topics.

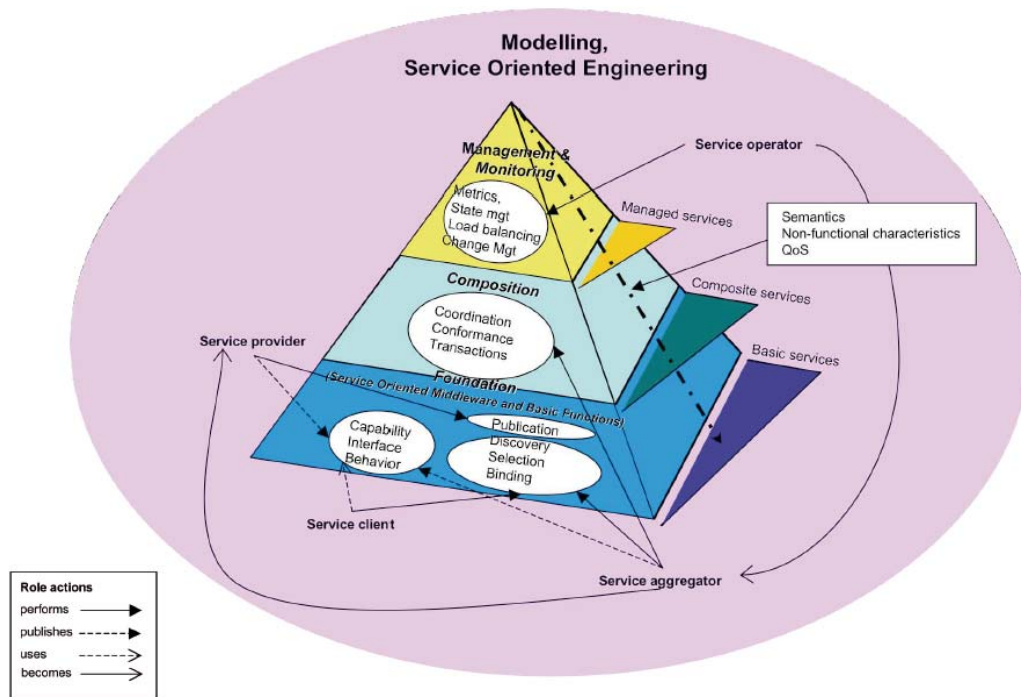


Figure 3: Research areas in the Services Research Roadmap defined in [3]

## 3.2 Current Research in SOA.

At the present time, SOA has gained a lot of attention for several research groups, for both in the academia and in the industry, whose contributions have been published in a significant number of new journals and conferences that have appeared within the scope of SOA systems. European research platforms, networks and projects have emerged in order to enhance the SOA research.

### 3.2.1 European Research Platforms, Networks & Projects:

**Nessi:** NESSI is the European Technology Platform dedicated to Software and Services. Its name stands for the Networked European Software and Services Initiative.

<http://www.nessi-europe.com>

**S-cube:** the Software Services and Systems Network, will establish an integrated, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, thereby helping shape the software-service based Internet which is the backbone of our future interactive society.

<http://www.s-cube-network.eu/>

**BEinGRID:** Business Experiments in GRID, is the European Union's largest integrated project funded by the Information Society Technologies (IST) research, part of the EU's sixth research Framework Programme (FP6).

<http://www.beingrid.eu/>

## 3.3 Conferences and Journals:

Since 2002, several SOA specific conferences have appeared. Additionally, most of Internet related conferences and software engineering conferences include SOA Systems in their program. Furthermore, there have also emerged new journals, making the SOA related topics a notorious research area.

### 3.3.1 SOA specific conferences:

**ICSOC:** International Joint Conference on Service Oriented Computing (ICSOC) is one of the most important conferences in Web Services.

**ICWS:** The IEEE International Conference on Web Services (ICWS) has been a prime international forum for both researchers and industry practitioners to exchange the latest fundamental advances in the state of the art and practice of Web services.

**SCC:** The International Conference on Services Computing ( SCC) is a conference that aims to bridge the gap between Services Computing and Business models with an emerging suite of ground-breaking technology that includes service-oriented architecture, business process integration and management, grid/utility/autonomic computing, and Web 2.0

**ECOWS:** The European Conference on Web Services (ECOWS) is the premier conference for both researchers and practitioners to exchange the latest advances in the state of the art and practices of Web Services.

**ServiceWave:** is an European conference organized by Nessi group. The conference of 2009 has been merged with ICSOC.

Conference	Organized by	2009 edition	Submissions (2008)	Acceptance (2008)	Acceptance rate
<b>ICSOC</b> (International Conference on Service Oriented Computing)	IEEE	7 <sup>th</sup>	151 papers	32 full and 20 short papers	20%
<b>ICWS</b> (International Conference on Web Services)	IEEE	7 <sup>th</sup>	?	?	16%
<b>SCC</b> (International Conference on Services Computing)	IEEE	6 <sup>th</sup>	?	?	18%
<b>ECOWS</b> (European Conference on Web Services)	IEEE	7 <sup>th</sup>	75 papers	23 full papers	30%
<b>Service Wave</b>	Nessi Group	2 <sup>nd</sup>	102 papers from 28 countries	28 papers	27%

Table 1: SOA specific conferences

### 3.3.2 Other conferences that include SOA:

**CAISE:** The Conference on Advanced Information Systems (CAISE) is one of the leading Information Systems conferences these days.

**ICSE:** International Conference on Software Engineering (ICSE) is one of the largest annual Software Engineering conferences with a big historical background in this area.

**ESEC/FSE:** A joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE). This conference of Software Engineering is celebrated alternatively in Europe and in USA.

**WWW:** International World Wide Web Conference (WWW) is an annual conference that since 1994 handles topics related to the World Wide Web. The huge number of submissions it receives every year makes it one of the attractive conferences in the area.

**ICEBE:** International Conference on e-Business Engineering (ICEBE) is one of the largest conferences on e-business engineering.

**ICWE:** International Conference on Web Engineering (ICWE) is a conference which aims on Web Engineering and Web-based applications.

Conference	Organized by	2009 edition	Submissions (2008)	Acceptance (2008)	Acceptance rate
<b>CAISE</b> (Conference on Advanced Information Systems)	-	21 <sup>st</sup>	?	?	?
<b>ICSE</b> (International Conference on Software Engineering)	IEEE / ACM	31 <sup>st</sup>	371	56	15%
<b>ESEC/ FSE</b> (European Software Engineering Conference / SIGSOFT Symposium on the Foundations of Software Engineering)	ACM	7 <sup>th</sup>	152	31	20%
<b>WWW</b> (International World Wide Web Conference)	IW3C2	18 <sup>th</sup>	880	103	12%
<b>ICEBE</b> (IEEE INTERNATIONAL CONFERENCE ON E-BUSINESS ENGINEERING)	IEEE	5 <sup>th</sup>	251	46	18%
<b>ICWE</b> (International Conference on Web Engineering)	IEEE	9 <sup>th</sup>	?	?	?

Table 2: General conferences that include SOA.

### 3.3.3 Journals:

**IEEE Internet Computing:** is a Journal published by the IEEE Computer Society that covers all internet related technologies. Its publication is done every two months.

**International Journal of Web Services Research (IJWSR):** is a web service specific journal published by the Idea Group Publishing (IGP) every four months.

**International Journal of Cooperative Information Systems (IJCIS):** is a journal focused on services and published by the World Scientific Journals every four months.

Journal	Published by	Periodicity
<b>IEEE Internet Computing</b>	IEEE	Bi-monthly
<b>IJWSR</b> (International Journal of Web Services Research)	IGP	quarterly
<b>IJCIS</b> (International Journal of Cooperative Information Systems)	World Scientific	quarterly

Table 3: Journals where SOA is a central topic.

# Systematic Review

# Definition

## 4 Research method for the systematic review.

### 4.1 Motivation

To develop a review in an accurate and objective manner, it is necessary to use a precise and rigorous methodology. The methodology chosen is significant in order to ensure the correctness of the State of the Art. Reviews following those rules are called systematic reviews, which are more accurate and reliable than traditional ones. The importance of the methodology used is critical, since unless a literature review is thorough and fair, it is of little scientific value.

For such a purpose, this work uses the systematic review guideline proposed by B. Kichenham [4]. This guideline has been derived from three existing guidelines used by medical researchers and adapted to reflect the specific problems of software engineering research.

### 4.2 The review Process

One of the key features of systematic reviews is the definition of a review protocol that specifies the methods that will be used to perform the review. Following the guideline [4], we should undertake this thesis accordingly to the following steps:

#### **1. Planning the review**

1. Identification of the need for a review
2. Development of a review protocol

#### **2. Conducting the review**

1. Identification of research
2. Selection of primary studies
3. Study quality assessment
4. Data extraction & monitoring
5. Data synthesis

#### **3. Reporting the review**

Although these stages are listed and presented in a sequential mode, it's important to clarify that the development of this stages involves iteration.

# 5 Planning the review

---

The planning of the review consists of 2 basic steps:

1. Identification of the need for a review
2. Development of a review protocol

## 5.1 Identification of the need for a review

Prior to undertaking a systematic review, we should ensure that a systematic review is necessary. As specified in the guideline, the first step is to search for existing systematic reviews of the given subject.

There's no procedure defined in [4] in order to search for these systematic reviews in an accurate manner. Nevertheless, we present here the procedure that has been followed for retrieving existing reviews. To increase the number of results, we will not focus just on systematic reviews, but on reviews and states of the art regardless the methodology followed for developing them.

We have searched the reviews in the following databases:

- ISI Web of Knowledge (<http://www.accesowok.fecyt.es/wos/>)
- IEEE Xplore (<http://ieeexplore.ieee.org>)
- ACM (<http://portal.acm.org>)
- Springer (<http://www.springerlink.com>)
- Google Scholar (<http://scholar.google.com>)

The following keywords have been used. In order to reduce the noise of the results, the following terms have been applied just to the title of the papers. We consider that, because of the nature of this kind of papers, they should have a relevant name in the title itself.

- 1) (QoS **OR** "Quality Of Service") **AND** Review
- 2) (QoS **OR** "Quality Of Service") **AND** State Of the Art
- 3) (SLA **OR** "Service Level Agreement") **AND** Review
- 4) (SLA **OR** "Service Level Agreement") **AND** State Of the Art
- 5) (QM **OR** "Quality Model") **AND** services **AND** Review
- 6) (QM **OR** "Quality Model") **AND** services **AND** State Of the Art

The results of the queries are shown in **Table 4**. As we can see, despite having some results from databases, none of them have a significant value for our search. This noise<sup>1</sup> is because the given results were related to other topics from different areas using the same acronyms for other concepts.

---

<sup>1</sup> Noise: in the bibliographic context, noise is defined as the set of results which are not related with the area or topic the user is interested on obtain.

	Google Scholar		Springer		ACM		IEEE Xplore		ISI Web of Knowledge	
	Results	Good	Results	Good	Results	Good	Results	Good	Results	Good
(1)	30	-	0	-	0	-	3	-	4	-
(2)	16	-	0	-	0	-	1	-	1	-
(3)	8	-	1	-	0	-	0	-	4	-
(4)	5	-	0	-	1	-	0	-	3	-
(5)	10	-	0	-	0	-	0	-	2	-
(6)	0	-	0	-	0	-	0	-	0	-

Table 4

Despite not having any valid document in the results, it is important to mention that there are indeed some reviews and states of the art of SOA Systems covering different Service Oriented topics, which includes the one of our interest [3][5]. Although they present significant information of the current state of QoS and SLAs, the limitations of covering the wider subject of all SOA related topics, makes necessary the development of a review strictly to the Quality attributes of web services.

We conclude that no review or State of the Art has been reported to the significant listed databases, and the published States of the art or reviews for SOA Systems are too wide to cover them deeply. To fill this gap, we need a more extended research to cover densely the current State of the Art of QoS related topics. Therefore, the development of a systematic review is required.

## 5.2 Development of a Review Protocol

A review protocol specifies the methods that will be used to undertake a specific systematic review. A pre-defined protocol is necessary to reduce the possibility researcher bias.

A review protocol is composed of the following elements:

- The research question that the review is intended to answer
- The strategy that will be used to search for primary studies
- Study selection criteria and procedures
- Data extraction strategy
- Synthesis of the extracted data
- Project timetable






### 5.2.1 The research question

The first step of the review is the development of the research question. During the development of this work, we will focus on those aspects of Quality of Service that might be useful in order to resolve/answer the following question that we propose:

**Is there a framework (or a set of them) which supports the discovery, selection, adaption and monitoring of web services based on the Quality of Services?**

### 5.2.2 The search strategy

To perform the systematic review, we have identified the following databases to search in:

<b>ISI Web of Knowledge</b> <a href="http://www.accesowok.fecyt.es/wos/">http://www.accesowok.fecyt.es/wos/</a>	
<b>Description:</b> ISI Web of Knowledge has a data source of 8.500 journals concerning to science and investigation. The data it stores are: most cited authors, institutions, rankings by country and journal. Its main characteristics are: Research Fronts (hot research topics) and Science Watch (comments from the scientific community). It's being updated every two months.	
<b>IEEE Xplore</b> <a href="http://ieeexplore.ieee.org">http://ieeexplore.ieee.org</a>	
<b>Description:</b> IEEE Xplore is a database produced by IEEE which includes the full publications of IEEE (Institute of Electric and Electronic Engineers) and IET (Institution of Engineering and Technology). It contains journals and proceedings from both institutions since 1988.	
<b>ACM</b> <a href="http://portal.acm.org">http://portal.acm.org</a>	
<b>Description:</b> It has complete access to the publications of ACM (Association for Computing Machinery), for both journals and proceedings.	
<b>Springer</b> <a href="http://www.springerlink.com">http://www.springerlink.com</a>	
<b>Description:</b> Database of journals and books published by Springer-Verlag and other editors, such as Kluwer. It includes 500 multidisciplinary journals, and also 1800 monographs of the collection Lecture Notes in Computer Science (LNCS), all of them specialized in computer science.	
<b>Google Scholar</b> <a href="http://scholar.google.com">http://scholar.google.com</a>	
<b>Description:</b> Google Scholar provides a simple way to broadly search for scholarly literature. Search across many disciplines and sources: peer-reviewed papers, theses, books, abstracts and articles, from academic publishers, professional societies, preprint repositories, universities and other scholarly organizations.	



After performing some searches, we have noticed that these databases retrieve very similar results but presenting them in a different order. The criteria used for each database to rank the results are critical to ensure the correctness of including the most relevant works and reduce the bias of our research.

After evaluating different options we have decided to use ISI Web of Knowledge and Google Scholar. The first is used because, as the same ISI Web of Knowledge states, all entries come from high quality resources carefully selected through an evaluation process. As they state, their objective is to include only the most influential, relevant and credible information available. The use of Google scholar is to embrace all those relevant works that are not included in ISI Web of Knowledge database.

As a consequence of our first searches, we have identified the existence of noise that could interfere on the quality of the query results. This is particularly critical in 'QoS' and 'Quality of Service' since these terms are commonly used in telecommunications regarding to the data transmission in a network. In order to reduce that noise, we can:

- a) Include some widely used SOA specific terms, such as 'web service' to exclude those non SOA-related topics.
- b) Exclude some specific terms which are only used in those 'noisy' subjects.

Depending on the quality of the results we should use strategy (a) or (b).

All queries will be performed on the abstract of the document. If the given database has not the option to search into the abstract (i.e. Google scholar) the search will be performed on the whole text of the paper.

To decide the exact query, we have tested them in the given databases. After some iterations checking the results of the queries, we have identified those ones retrieving the best results. Therefore, we conclude that the following queries will be used to obtain the primary studies of our work:

- 1) (QoS **OR** "Quality Of Service") **AND** (webservice? **OR** "web service?")
- 2) (SLA **OR** "Service Level Agreement") **AND** (webservice? **OR** "web service?")
- 3) (QM **OR** "Quality Model") **AND** (webservice? **OR** "web service?")

Another important issue that we have to take into account is that most of the databases order the 'relevance' results computing the number of citations the paper has. Hence, there might be an important bias as a paper which has been published in an old stage might have accumulated more citations than a newer but more relevant one. Furthermore, a very recent paper may have very few or even no citations, despite its contribution could be very significant for the research community.

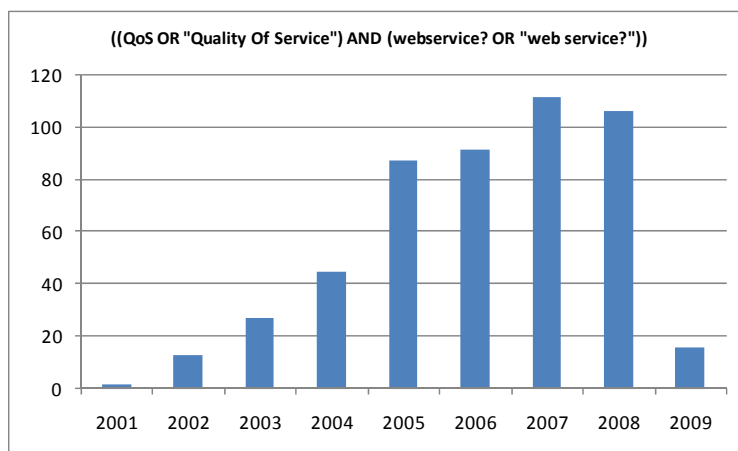
We have taken the following strategy in order to reduce the possible bias given by the citations:

We have searched the queries (1), (2) and (3) in one of the most representative databases, ISI Web of Knowledge. We have performed the search for each one of the years from 2001 to 2009. The number of papers selected for each year for the systematic review has been weighted with the number of papers published in that year, with the exception of 2009 where all papers have been chosen. The reason of choosing all papers from 2009 is because of the youth of

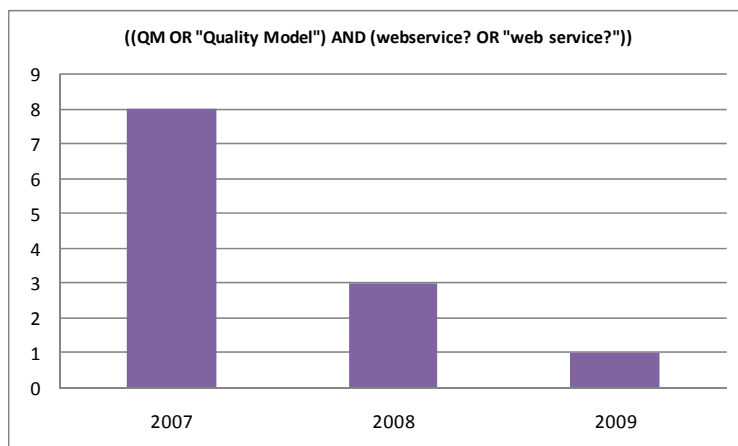
these documents and consequently none of them has citations; therefore the only criteria possible would be the reputation of the publisher (either for congress or journal). By the other hand, as the number of papers published in the first months of 2009 doesn't represent a big extra effort for the systematic review, we consider that the inclusion of these works are an important aspect of our work.

As we can see in the results shown in the following charts, the interest on this area has been increasing either for QoS in the last decade. This is an important factor also for our search policy since we will retrieve more documents from recent years than from elder ones.

As we mentioned in the introduction, we considered focusing deeply on these topics and exclude SLA in the scope of this Master Thesis.



Quality of Service has been a subject of study which has been increasing year after year from 2001 to the peak at 2007.



Concerning quality models, this is a very recent area as there are results only since 2007.

## 5.3 Study selection criteria

Study selection criteria are intended to identify those primary studies that provide direct evidence about the research question. In order to reduce the likelihood of bias, selection criteria should be decided during the protocol definition.

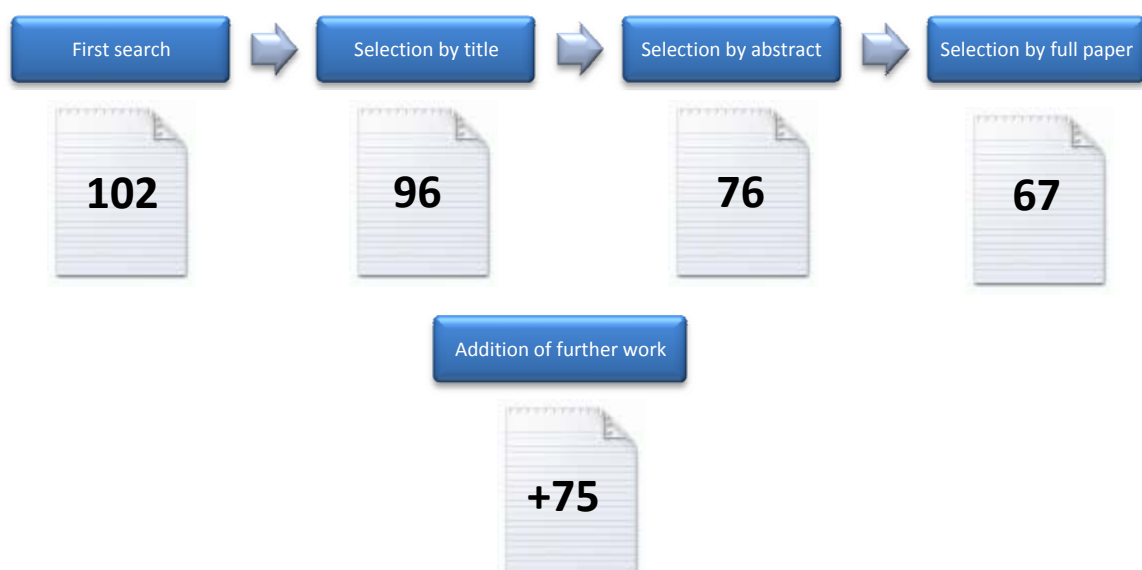
After retrieving the results weighted by year, we define the selection criteria in the following steps:

1. **Selection by title:** The objective of this first filter is to identify and remove the noise of the results. After this selection, documents whose scope is not related with Quality of Services are removed.
2. **Selection by abstract:** At this stage, we discard all works that although being related with Quality of Services are definitely out of the scope. This can be due to the fact that Quality of Service is not the primary contribution of the paper
3. **Selection by full paper:** At this point, we remove by the full text all papers which their contribution are not relevant or present poor results.
4. **Addition of further work:** During the process of the systematic review, other works might be included. This process is performed through obtaining relevant citations of the papers and through further work of the researchers.

From our searches we have retrieved 102 papers weighted by year of publication. From them, we have discarded 6 documents by title resulting in 96 papers. 20 papers were excluded by abstract; resulting in 76 papers to evaluate by full paper. At this point, just 9 papers were discarded by full paper, the main reasons were:

- The language of the paper was not English.
- The paper was not accessible.
- The paper presented was too similar to another already presented by the same author.

Some works might have been evolved from these papers, so the search of this future work has also been taken into account during the process of conducting the review.



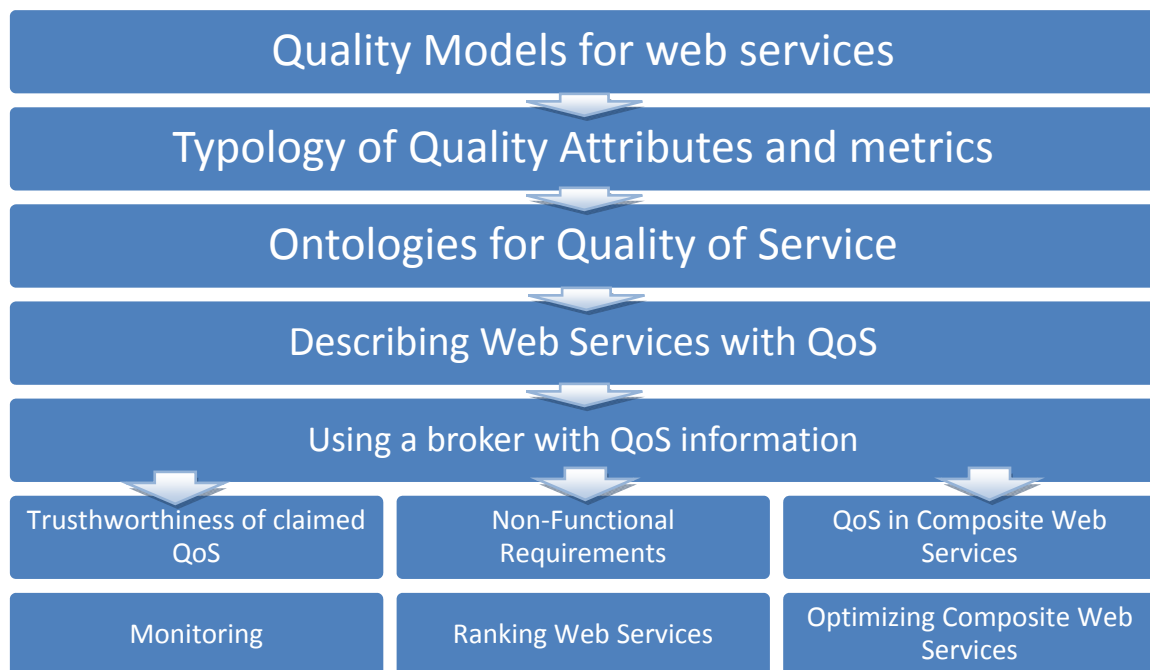
# Systematic Review Results

## 6 Introduction

---

Given the criteria and protocols defined in the previous chapter, the synthesis of the retrieved documents is presented. Here we will describe the State of the Art of the different subtopics related to QoS, from the basis of Quality Models to upper layers such as ranking or monitoring.

As shown in the following diagram, we present a ‘roadmap’ identifying the dependencies between these chapters, in order to provide to the reader a helpful structure of this work.



## 7 Quality Models for web services

---

A Software Quality model is a structured set of Quality Characteristics of Software.

There exist several Quality models for Software Systems, one of the most relevant was established in ISO9126[6], classifying the software quality in a structured set of characteristics and sub-characteristics as follows:

- **Functionality:** Suitability, Accuracy, Interoperability, Compliance, Security
- **Reliability:** Maturity, Recoverability, Fault Tolerance
- **Usability:** Learnability, Understandability, Operability
- **Efficiency:** Time Behaviour, Resource Behaviour
- **Maintainability:** Stability, Analyzability, Changeability, Testability
- **Portability:** Installability, Replaceability, Adaptability, Conformance

However, we have to mention, that not all of these software quality attributes presented in [6] are applicable to Web Services. For instance, installability can obviously not be applied to a web service. In this sense, those Quality Characteristics related to web services are known as Web Service Quality Characteristics or also Web Service Quality Factors.

Most of these quality sub-characteristics/sub-factors are also known in the literature of web services as Quality Attributes. In IEEE610[7] a quality attribute is defined as “A feature or characteristic that affects an item’s quality.”

In a nutshell, the goal of quality Models is to group all quality attributes into a hierarchy of quality characteristics.

The sum of all these quality characteristics and attributes applied to a Web Service is defined as QoS. As stated in ISO 8402 [8], QoS is “the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs”.

A Quality Attribute is, nevertheless, not a quantitative measurement. We can’t state, for instance, that the time behavior of a web service is 40 ms. Instead, we talk about Maximum response time, Latency, Average Execution Time, beyond others. These kind of sub-attributes are known as Quality Metrics. As defined in [9] “a quality metric is a quantitative measurement of the degree to which an item possesses a given quality attribute”.

For instance, from the quality attribute Availability we might be interested on several metrics:

- **Mean Time Between Failure:** Average time between failures of the service
- **Average availability:** The % of time the service is available
- **Downtime per year:** The time the service has been unavailable in a year.

A quality metric is also defined in the book of Kan [10] as follows: “Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project”.

Additionally, in the same book, Software metrics are classified into these three categories:

- **Product metrics** describe the characteristics of the product
- **Process metrics** describe the software development and maintenance.
- **Project metrics** describe the project characteristics and execution.

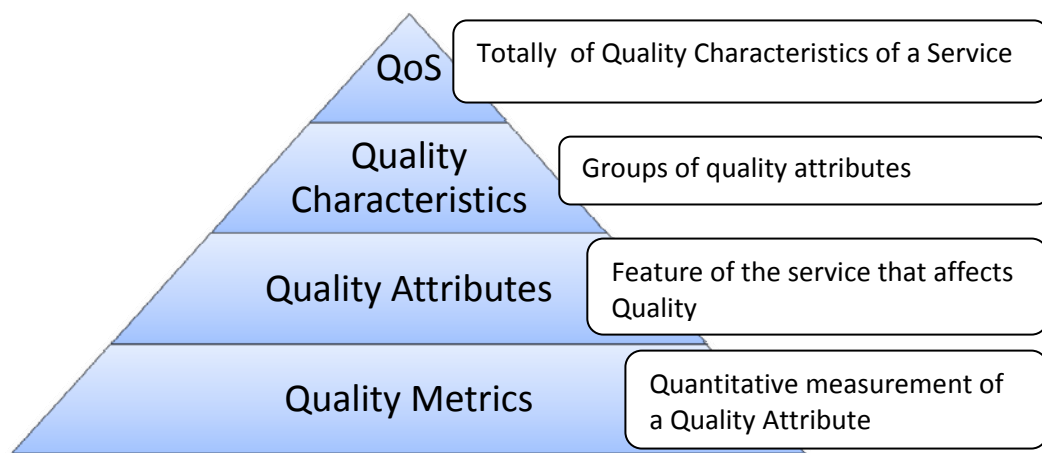


Figure 4: Hierarchy of Quality concepts

Despite the clear theoretical difference between these concepts, the line differentiating each of them can be sometimes blur, which involves on some contributions to use them arbitrarily.

Since 2005 and based on ISO9126, OASIS has been developing a Quality Model for Web Services, named WSQM, consisting of three parts:

- **Quality Factors:** Describes the characteristics, subcharacteristics and attributes of Quality for Web Services
- **Quality Activities:** Are the activities performed during the lifecycle of a web service regarding to their QoS.
- **Quality Associates:** Are the persons or organizations that are involved in Quality Activities

Although WSQM is the set of the three submodels, in this review we have focused on the Quality Factors, which defines the set of quality characteristics and attributes for web services. In their last draft written in November 2008, they defined the Quality Factors model as follows:

- **Business Value Quality:** a quality factor which can be referred to assess in the consistency degree of business purpose
  - **Service cost:** Price, Penalty/Compensation, Billing.
  - **Service suitability:** Business suitability, Usability.
  - **Service Aftereffect:** Business effect, Return On Investment, Consumer Satisfaction.
  - **Service Brand Value:** Recognition, Reputation.
- **Service Level Measurement Quality:** defines quantitative attributes which could be measured while Web services are using.
  - **Performance:** Response time, Maximum throughput.
  - **Stability:** Availability, Accessibility, Successability.
- **Suitability for Standards:** the capability of which a Web service is communicable with the other Web service on a different system or a platform
  - **Conformability:** Message Exchange Conformability, Service Definition Conformability, Service Search and Registration Conformability.
  - **Interoperability:** Services Basic Interoperability, Services Security Interoperability
- **Business Process Quality:** performance indicators needed to represent functionality for collaboration among two or more Web Services
  - **Reliable Messaging**
  - **Transaction Processing Capability:** Short-Term Transaction, Long-Term Transaction
  - **Collaborability:** Orchestration, Choreography
- **Manageability Quality:** Manageability Quality refers to an index of ability to consistently manage Web services
  - **Management Information Offerability**
  - **Observability:** State, Operational Status, Metric Information, Message Exchange Pattern
  - **Controllability:** Control, Configuration
- **Security Quality:** is the degree of ability that can provide stable and reliable services by protecting the services and messages from unauthorized access, forgery, and destruction

- **Confidentiality:** Transport Level Data Confidentiality, Message Level Data Confidentiality
- **Integrity:** Transport Level Data Integrity, Message Level Data Integrity
- **Authentication:** Transport Level User Authentication, Message Level Data Authentication
- **Access Control:** Transport Level Access Control, Message Level Access Control
- **Non-Repudiation:** Message Level Non-Repudiation.
- **Availability:** Transport Level Availability, Message Level Availability
- **Traceability:** Transport Level Audit, Message Level Audit Trace
- **Privacy**
- **Distributed Authorization**

Despite the completeness of these models, we have to point out that most of the works regarding to QoS of web services don't use these Quality Models. In my opinion, this fact can be because there's still not a widely accepted Quality Model for web services. Due to the fact that ISO9126 is not specifically designed for web services whereas the OASIS proposals, although being a very complete model, is still a working draft not widely known by the community.

As a consequence, most of the contributions focus only on a small set of typical quality attributes for web services. The most significant are Response Time, Availability, Cost and Reliability.

Some other works, however, propose their own Quality Model as a basis for their contribution. For instance, in [11] Wang et al. proposed the following Quality Model as a starting point for their contribution.

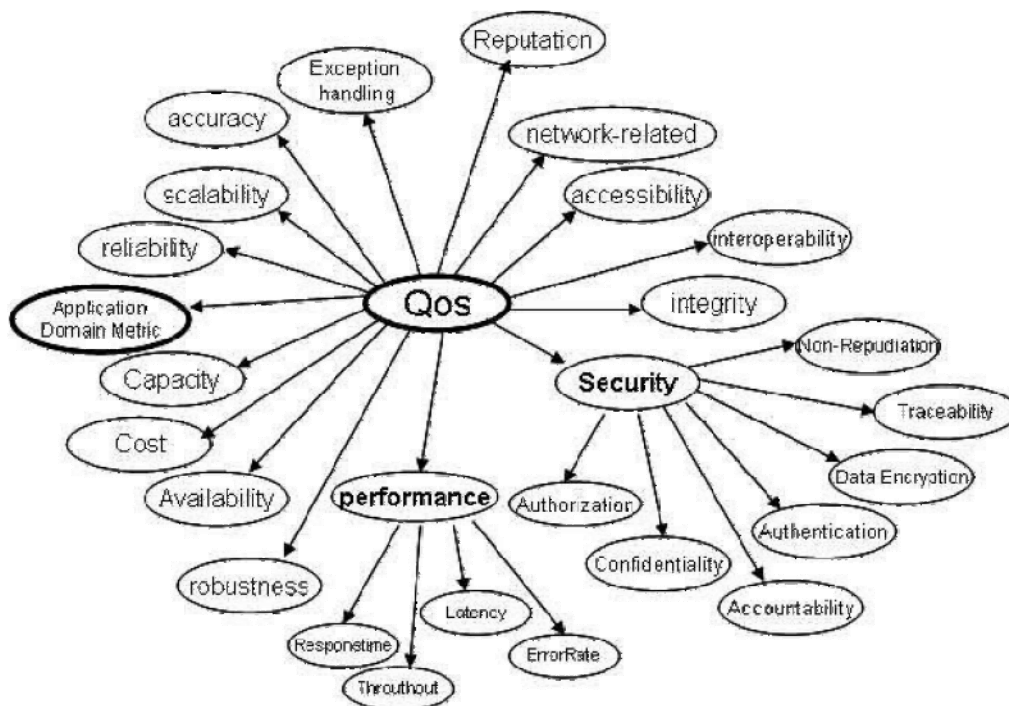


Figure 5: Quality model used in Wang et al. contribution [11]



## 8 Typology of Quality attributes and metrics

---

Apart from the given taxonomies, Quality attributes and metrics can be also classified from different perspectives.

From a domain perspective. Quality attributes are classified from domain-independent or domain-specific attributes.

- Domain-independent attributes: are those that can be applied to any kind of web service. Examples of Domain-independent Quality Attributes are Response Time, Availability and Cost.
- Domain-specific attributes: are those that can only be applied to a certain(s) domain(s). For instant, in a weather forecasting domain, a Quality Attribute could be forecast accuracy.

From a measurable perspective, as stated in [12], Quality metrics can be classified through the method the data can be obtained. In this sense, 3 categories have been identified:

- Provider-advertised metrics: Metrics which their values are obtained by the provider's advertisement. One clear example would be the cost of the service.
- Consumer-rated metrics: Metrics which their values are given from the users' opinions. For instance, the Service reputation obtained from an average of customers' opinions.
- Observable metrics: Metrics which their values are obtained through monitoring or testing. A clear example is Response Time or Availability.

Another kind of classification is from the fact that some Quality Metrics can be obtained as an aggregation function of other quality metrics. For instance, Average Response Time is the average of a set of Current Response Times in different time intervals. These quality metrics are named derived or calculated metrics, whereas the others are named basic or atomic metrics.

## 9 Ontologies for Quality Of Service

---

An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts.

As explained in the prior chapter, a quality model defines concrete metrics, attributes and characteristics to be used in web services. But quality metrics, attributes, and characteristics have, in turn, associated information. That is, metrics might have a domain and a unit, and we might be interested to know if their value change or not on run-time. Quality attributes might have a predefined importance in a given SOA System, we might like to put restrictions on their quality metrics, or advertise our service with quality information, etc.

Building a common ontology with this information is crucial if we want to share the Quality Information of a Web Service between Service Providers, Service Clients, Service Registries, Brokers, etc.

By the other hand, the importance of ontologies in web services has been increasing because of the efforts on the development of the next generation of web services, called Semantic Web Services (SWS), which is closely related to the next wave of the web, the semantic web. Semantic web services are those ones that are not only machine-readable but also machine-understandable, facilitating then the following aspects:

- Automatic web service discovery
- Automatic web service invocation
- Automatic web service composition

Focusing on these points, there are various ontologies for web services, however not all of them takes into account the QoS, and therefore some extensions of these ontologies have been proposed.

One of the most widely known ontology for web services is OWL-S. As defined in [13] “OWL-S is an ontology, within the OWL-based framework of the Semantic Web, for describing Web services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints.”

OWL-S is divided in 3 big groups, it provides information of what the service does through the ServiceProfile, how to invoke the service through the ServiceGrounding and how this service works through the Service Model.

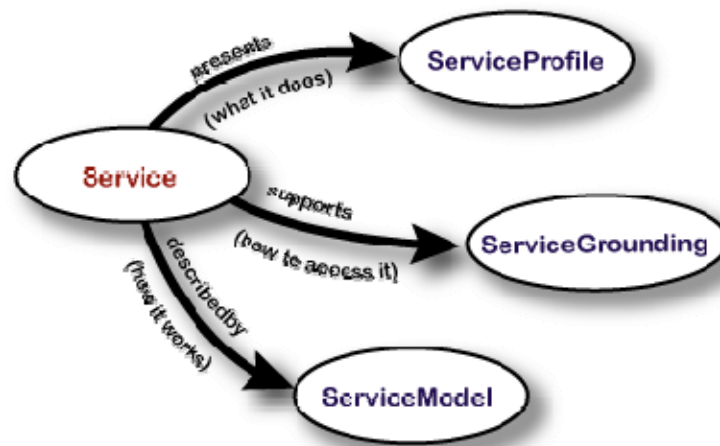


Figure 6: Top Level of OWL-S ontology

OWL-S was defined in 2003 as an evolution of its predecessor DAML-S written in 2001. OWL-S is currently in its version 1.1 and it was included in the list of acknowledgment Member Submissions of W3C.[14]

However, one of the weak points of both OWL-S and DAML-S is their lack of a detailed set of classes, properties and constraints to represent QoS descriptions [15]

In order to resolve this problem, several contributions have been proposed to extend the current these models. One of the first developed extensions was proposed in [16] by Zhou et al. who presented an extension of DAML-S ontology, named DAML-QoS. Notice that, although when the paper was written OWL-S was already defined, the better tool support for DAML-S was a key point in order to decide to extend the DAML-S ontology. This contribution proposed a layered ontology consisting of the following 3 layers:

- **QoS profile layer:** designed for matchmaking purpose between Service Providers and Service Requesters
- **QoS property definition layer:** for defining the property and elaborating the property's domain and range constraints.
- **QoS metric layer:** for metrics definition and measurements.

QoS profile layer	
QoSProfile $\sqsubseteq$ T ProviderQoS $\sqsubseteq$ QoSProfile InquiryQoS $\sqsubseteq$ QoSProfile TemplateQoS $\sqsubseteq$ QoSProfile	<b>ProviderQoS:</b> Advertisement ontology provided by the service provider <b>InquiryQoS:</b> Service requester's inquiry ontology for QoS matchmaking <b>TemplateQoS:</b> is stored by any user for further usage or modification.
QoS property definition layer	
QoSProfile $\sqsubseteq$ T QoSCore $\sqsubseteq$ QoSProfile QoSInput $\sqsubseteq$ QoSProfile QoSOutput $\sqsubseteq$ QoSProfile QoSPrecondition $\sqsubseteq$ QoSProfile QoSEffect $\sqsubseteq$ QoSProfile	<b>QoSCore:</b> QoS Property defined with independence of input-output <b>QoSInput:</b> QoS Property related to the input (i.e: inputBitRate) <b>QoSOutput:</b> QoS Property related to the output(i.e: outputBitRate) <b>QoSPrecondition:</b> Precondition of the QoS required by the Service <b>QoSEffect:</b> Expected effect on the QoS that results from the execution of the service
QoS metric Layer	
Metric $\sqsubseteq$ T AtomicMetric $\sqsubseteq$ Metric ComplexMetric $\sqsubseteq$ Metric	The Metric class has related properties: unit, value, metricName.

Table 5: Layers of DAML-QoS proposed by Zhou et al. [16]

Using this ontology, a service provider could advertise web services with QoS information just using DL syntax. For instance, to advertise a web service with a response time inferior to 20 seconds and a cost of no more than 1 dollar, the following statement could be written (notice that the units used are milliseconds and cents):

**Advert** = QoSProfile  $\sqcap$   
 $(\leq 100 \text{cost:CostUSCentMetric}) \sqcap$   
 $(\leq 20000 \text{responseTime:RespMSMetric})$

The service client could, in turn, define a request with QoS information to retrieve all web services whose response time is no more than 40 seconds and its cost is less than 5 dollars, as it follows:

**Inquiry** = QoSProfile  $\sqcap$   
 $(\leq 500 \text{cost:CostUSCentMetric}) \sqcap$   
 $(\leq 40000 \text{responseTime:RespMSMetric})$

Since the extension of OWL, there have been several contributions and proposals for extensions. Although some of them represent in my opinion an excellent contribution, none have been proposed to be included in the standard.

In [17], Giallonardo et al. presented a 3-layer ontology in OWL named onQos for describing QoS. In its approach three layers were identified:

- **Upper ontology:** describes the QoS ontological language. It provides “the words” we need in order to provide the most appropriate information for formulating and for answering the QoS queries.
- **Middle ontology:** defines the standard vocabulary of the ontology, such as, for example QoS parameters, QoS metrics and QoS scales.
- **Low ontology:** defines the concepts, the properties and the constraints for a specific domain.

In [18], Kritikos et al. developed another OWL-S extension ontology, named OWL-Q. This ontology is separated into several facets. The rationale is that each facet concentrates on a particular part of QoS.

- **Connecting facet:** connects OWL-S with OWL-Q and provides high-level concepts appropriate for defining QoS advertisements and demands.
- **Basic facet:** Describes the most important components of QoS demands and QoS advertisements
- **QoS Metric facet:** is used to define any kind of metric
- **Function, Measurement Directive and Schedule facets:** The function Facet describes all the appropriate concepts of metric functions. The Measurement Facet describes all the concepts for measurements. A Schedule is used to compute the frequency of the computation of a Complex Metric.
- **Unit facet:** Describes the unit of a QoS metric.
- **QoSValueType facet:** Defines the types of values a QoS metric can take.

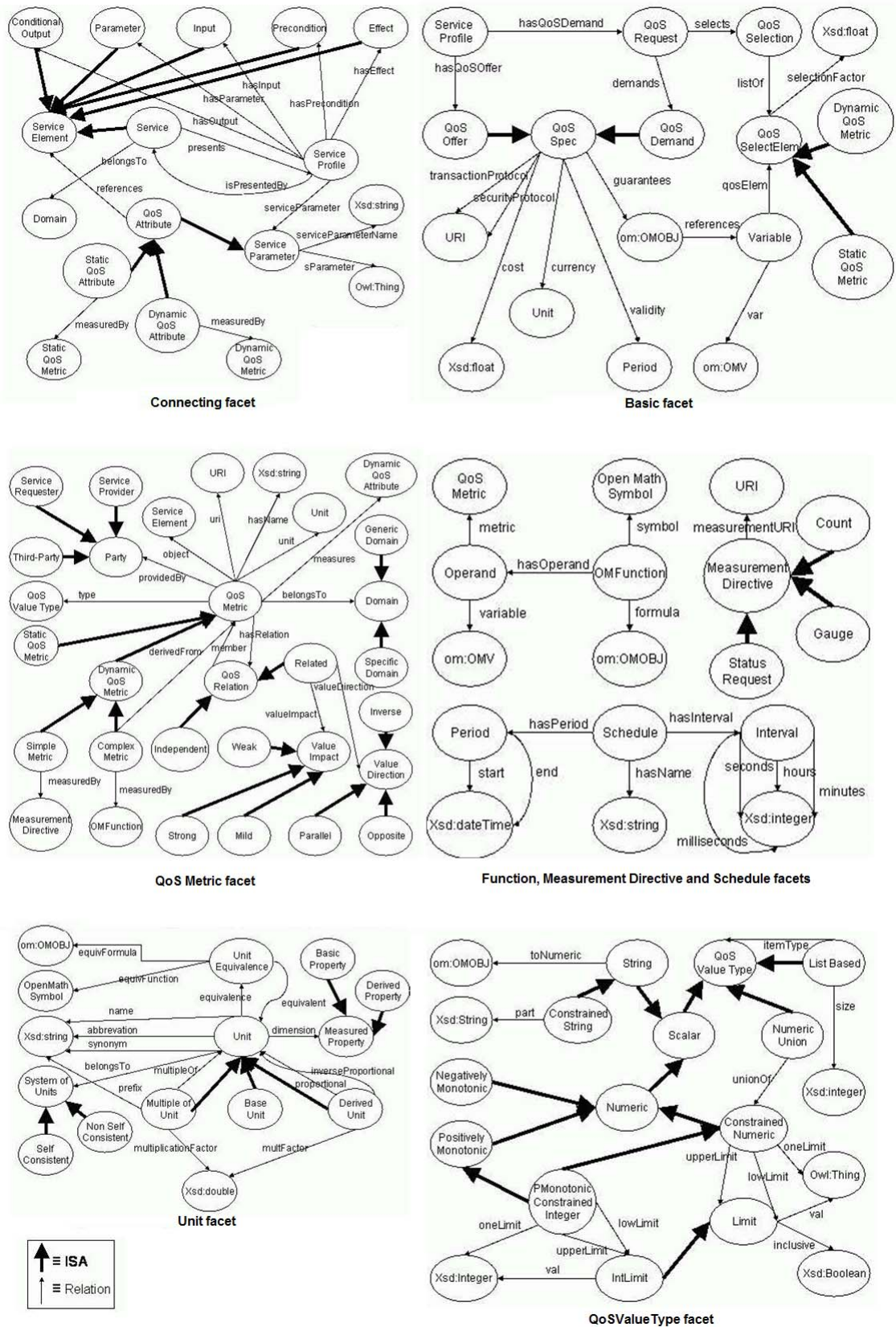


Figure 7: Facets of the Ontology proposed by Kritikos et al. [18]

Recently, Zhang et al. proposed in [19] another OWL-S extension. Notice that the presented work is not only focused on QoS, but on several aspects of web services. However, it provided a QoS ontology for web services, which were defined with the following quality attributes: Latency, Reliability, Accuracy, Scalability, Availability, Capacity and Cost.

Another widely known ontology, is WSMO (Web Service Modeling Ontology)[20], this proposal is in the list of the acknowledged Member Submissions to W3C. The most important concepts of WSMO are:

- **Goals:** representations of an objective for which fulfillment is sought through the execution of a Web service.
- **Ontologies:** Provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse.
- **Mediators:** Connectors between components of WSMO elements. Provides interoperability between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining Web services (process level).
- **Web Services:** Semantic description of Web Services. May include functional (Capability) and usage (Interface) descriptions.

WSMO is defined in MOF and includes QoS Information in its ontology. Nevertheless, these quality aspects are part of the non-functional information of a Web service description which are statically defined as: Accuracy, Availability, Financial, Network-related QoS, Performance, Reliability, Robustness, Scalability, Transactional and Trust.

Because of the lack of expressivity and flexibility of WSMO for describing Quality aspects, Wang et al. proposed in [11] an extension of WSMO, named WSMO-QoS, which extends the existing WSMO model by defining the QoS as a subclass of a nonFunctionalProperty.

```
Class QoS sub-Class nonFunctionalProperties
hasMetricName type string
hasValueType type valueType
hasMetricValue type value
hasMeasurementUnit type Unit
hasValueDefinition type logicalExpression
    multiplicity = single-valued
isDynamic type boolean
isOptional type boolean
hasTendency type {small, large, given}
isGroup type boolean
hasWeight type string
```



Not all contributions are extensions of these standards. One of the most recent and relevant contributions with independence of OWL-S or WSMO was presented by Jureta et al. in [21], where they proposed a new Quality Ontology named Quality-Value-Dependency-Priority (QVDP) model. This QVPD consists of 4 distinct submodels:

- **Q (Quality Characteristic submodel):** Describes the concepts of quality dimensions  $q$ , and quality characteristics  $\bar{q}$ . (Notice that a quality dimension  $q$  is, in fact, a quality metric, whereas a quality characteristic  $\bar{q}$  is the set of distinct/aggregate quality dimensions).
- **V (Quality Value submodel):** Defines how quality is to be measured
- **D (Quality Dependency submodel):** Defines interdependencies between values of distinct quality dimensions. That is, how the behavior of one dimension can affect to another's one.
- **P (Quality Priority submodel):** Establishes priorities between Quality dimensions. This submodel is used for the following reason: when increasing a value of a quality dimension affects negatively to another (this interdependency is given by the Dependency Model), we can establish which dimension has a higher priority to be optimized.

As the authors state, the novel points of their contributions are the Dependency submodel and Priority submodel, which could be used for web service optimization.

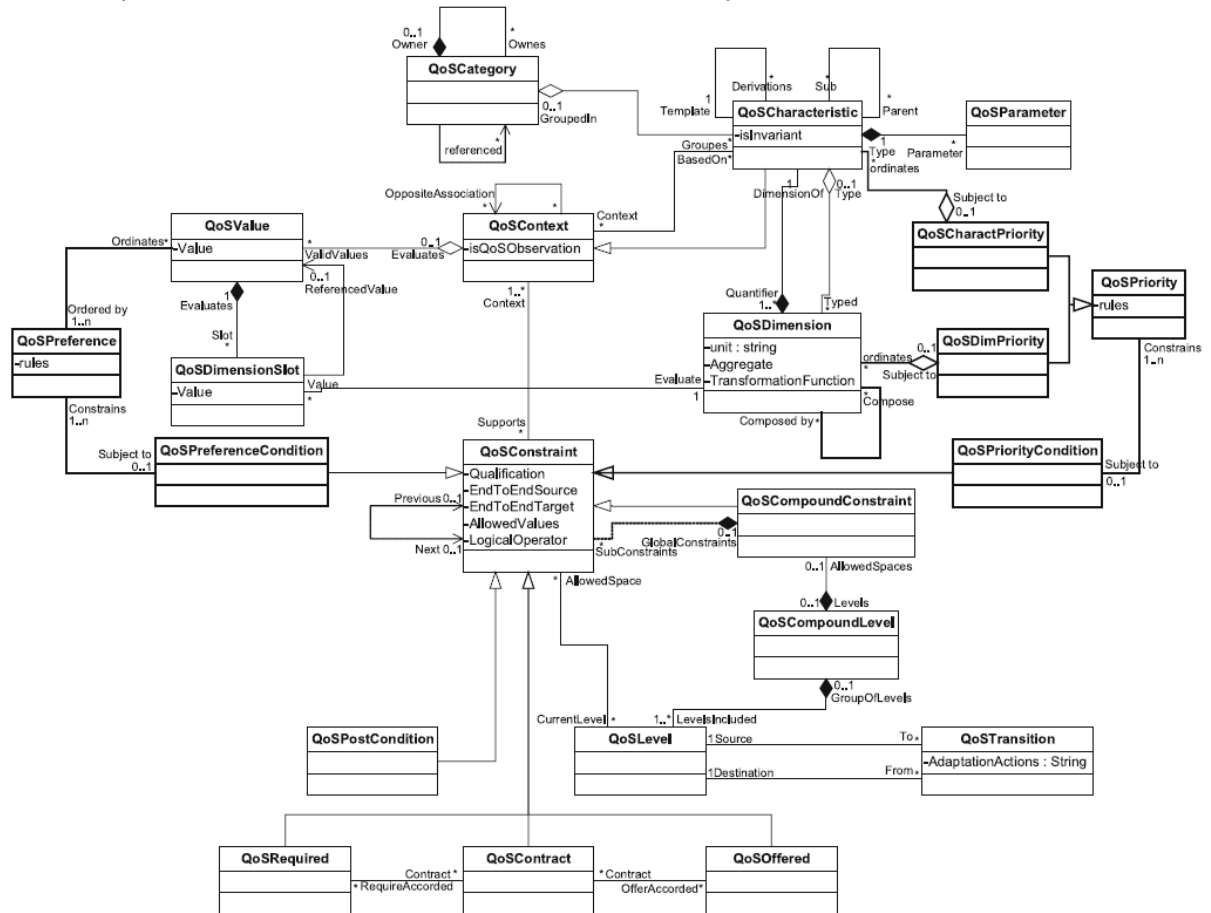


Figure 8: QoS metamodel with the proposed extensions in QVDP[21]



Finally, we present here a table of the different proposals. For each one, we present the language they have used to be described, the ontology that they extend (if any) and the impact of the contribution.

Ontology	Defined in	Extends	Impact
<b>DAML-QoS[16]</b>	DAML	DAML-S	Basis for other approaches in OWL.
<b>onQoS[17]</b>	OWL	-	-
<b>OWL-Q [18]</b>	OWL	OWL-S	-
<b>Zhang et al.[19]</b>	OWL	OWL-S	-
<b>WSMO[20]</b>	MOF	-	Acknowledged Member Submission to W3C
<b>WSMO-QoS [11]</b>	MOF	WSMO	-
<b>QVDP [21]</b>	UML	-	-

Table 6: Ontologies with QoS information

# 10 Describing web services with QoS

---

Functionality in Web services are defined in WSDL, an XML standard which specifies the operations that the web service can perform. The information provided in the WSDL covers all the data needed to invoke the web service itself but lacks of specifying the Quality of Service.

By the other hand, those services may be registered in a UDDI. A UDDI is a library of web services which are used to discover them; subsequently the client can select the one which suits him most. Nevertheless, discovering Web services are founded on using keyword-based search techniques, which may not return suitable results to clients' requirements since they might not reach the desired QoS.

In order to fulfill these problems there are several proposals to extend either UDDI or WSDL standards with QoS information. There are also proposals of adding a new role, a broker, to fill this gap.

Notice that this information might not represent the real Quality values for the set of attributes which may vary on run time, but the one claimed by the provider instead. Therefore, the provider's trustworthiness is another important part to take into account.

## 10.1 WSDL extension with QoS

WSDL (Web Services Description Language) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.<sup>2</sup>

In a WSDL document the functionality of a service is specified as it defines the methods of the service and how they are invoked. Nevertheless, it lacks of support for specifying non-functional properties such as QoS. In order to resolve this problem, several proposals of extending WSDL with Quality of Service information has been presented [22][23] [24]

Several works needs as a basis for their contributions a WSDL extension with QoS. Most of these contributions use a simple WSDL extension. For instance, in [24] Kang proposes the use of annotations, which is currently supported by WSDL standard, to describe WSDL with QoS information. An annotation is similar to a comment, and is used in WSDL mainly to embed the documentation.

As semantic web services are increasing on popularity, a new kind of annotations has emerged: SAWSDL (Semantic Annotation for WSDL and XML Schema). SAWSDL is an extension of WSDL that provides semantic annotations through the use of ontologies[25]. This WSDL extension is a W3C Recommendation since 2007[26].

Using SAWSDL, a service provider can advertise his web service linking its components with an ontology written in OWL. However, because of the novelty of SAWSDL, most of approaches in QoS extensions are still focused in WSDL and provide their own way to express this QoS.

---

<sup>2</sup> Definition from <http://www.w3.org/TR/wsdl>

In [22] the author presents an extension of WSDL with QoS data using model-driven techniques.

From the standard WSDL XML Schema, the author introduces the corresponding WSDL metamodel written in MOF. Such a transformation is performed using the XMI (XML Metadata Interchange), which is a language used in Model Driven Development to transform models in XML and viceversa.

From the WSDL metamodel, another transformation is performed to get a QoS-enabled WSDL metamodel (Q-WSDL). The resulting Q-WSDL metamodel is an extension of the original. This extension has been developed in a way that instances of WSDL metamodel are also compliant with Q-WSDL. Finally, another transformation is performed to get the Q-WSDL XML Schema. All this transformations are outlined in Figure 9.

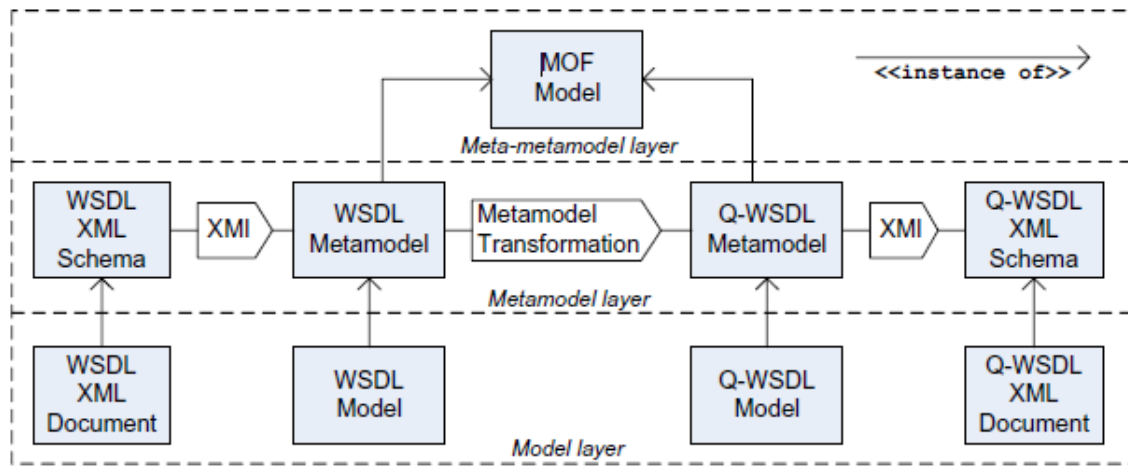


Figure 9: WSDL transformation to Q-WSDL

The resulting Q-WSDL from these transformations is an extended WSDL with several Quality Attributes. As we can see in Figure 9, some of these quality characteristics applies to the entire service (i.e. Availability and reliability) whereas some others applies to the operations of the service (i.e. Operation Latency, operation demand). Other elements of the service have also their own quality characteristics, port has network quality characteristics, and messages have Message Encryption quality characteristics.

In [23], the author presents WSDL-S, an extension of WSDL with semantic concepts defined in ontologies. It suggest to define Classes and properties in the domain model, and then map them into XML-complexTypes and XML-elements respectively from the WSDL. This approach is similar in how SAWSDL works in the way that it establishes an interrelation between WSDL entities and Ontological entities.

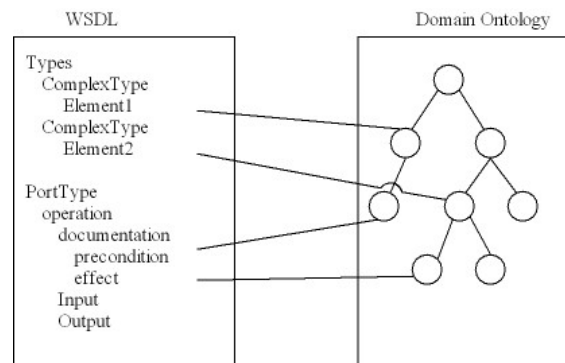


Figure 10: mapping

But one of the most prominent contributions, in my opinion, comes from the OASIS open group. The same group who is developing the quality model WSQM, is also working in a WSDL extension with QoS named WSQDL ( Web Service Quality Description Language)[27]. Whereas WSQM is a working draft, as stated in the oasis-openg.org, WSQDL is an approved document [28].

WSQDL is a WSDL extension able to express the quality items of WSDM. In order to provide a way to represent all quality elements, in WSQDL, the quality attributes represented in WSQM were divided into item types by binding them into groups of similar nature:

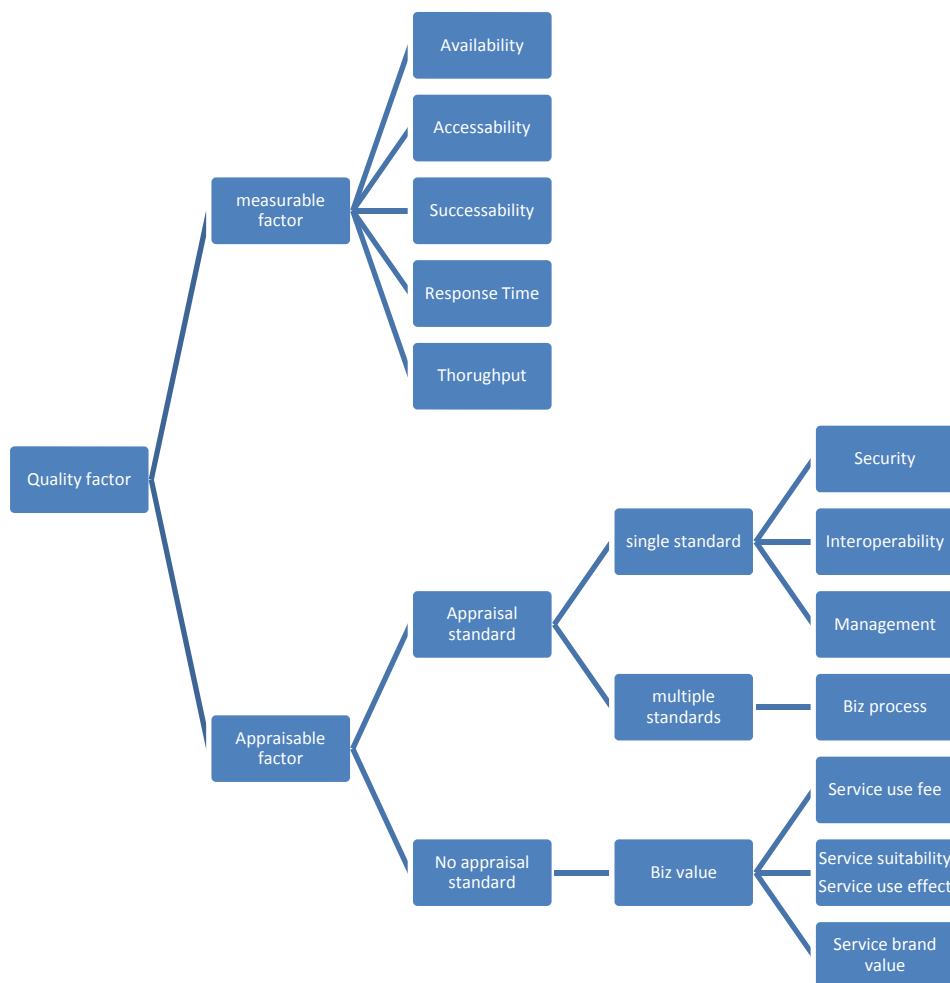


Figure 11: Quality attributes of WSQDL

Using this structure, WSQDL expresses a way to define a Web Service as a complex Type in the WSDL itself. This type has in turn quality factors, and these factors are divided in the presented groups. A complete WSQDL schema is provided in the document with an exhaustive example. We present here an extract of this WSQDL instance example, with just response time and service cost.

```
<QualityFactor>
  <MeasureFactor>
    <ResponseTime>
      <EnvVariables>
        <Variable>
          <VarName>number of CPU</VarName>
          <VarValue>4</VarValue>
        </Variable>
      </EnvVariables>
      <MetricValue>
        <Range>0.12-0.17</Range>
        <Type>float</Type>
        <Unit>second</Unit>
      </MetricValue>
    </ResponseTime>
  </MeasureFactor>
  <BizValueFactor>
    <ServiceCost>
      <ServicePrice>
        <Price unit="won">100000</Price>
        <ContractDoc>
          <ContractDocNumber>CN20034324</ContractDocNumber>
        </ContractDoc>
        <PayMethod>quarterly</PayMethod>
      </ServicePrice>
    </ServiceCost>
  </BizValueFactor>
</EvalFactor>
<Security>
  <Property name="confidentiality">
    <SubProperty name="message level confidentiality">
      <Function name="XML ENC Handler" severity="1">
        <Conformity>
          <Specification>
            <Name>WS-I BSP</Name>
            <Version>1.0</Version>
          </Specification>
          <RuleCategoryComformity>
            <RuleCategory name="Security Header">
              <SatisfiedRules>R3212 R3206 R3210</SatisfiedRules>
            </RuleCategory>
            <RuleCategory name="Timestamp">
              <SatisfiedRuleRatio>0.98</SatisfiedRuleRatio>
            </RuleCategory>
          </RuleCategoryComformity>
          <TotalConformity>
            <TotalSatisfiedRuleNumber>70</TotalSatisfiedRuleNumber>
          </TotalConformity>
        </Conformity>
        <Description/>
      </Function>
    </SubProperty>
  </Property>
</Security>
</EvalFactor>
</QualityFactor>
```

## 10.2 UDDI extension with QoS

UDDI ( Universal Description, Discovery and Integration) is an XML-based registry which defines a set of data structures to describe Web services for the purpose of advertisement and discovery. Service providers can publish their web services to an UDDI whereas the clients can discover them through different approaches.

As part of UDDI, we have tModels. A tModel is a data structure representing a service type (a generic representation of a registered service) in the UDDI registry. Each business registered with UDDI categorizes all of its Web services according to a defined list of service types. Businesses can search the registry's listed service types to find service providers. The tModel is an abstraction for a technical specification of a service type; it organizes the service type's information and makes it accessible in the registry database. Another UDDI data structure, the bindingTemplate organizes information for specific instances of service types.<sup>3</sup>

Each tModel should contain an overviewURL, which references a document that describes the tModel and its use in more detail. Here we show a simplified UML Model for UDDI Information Model:

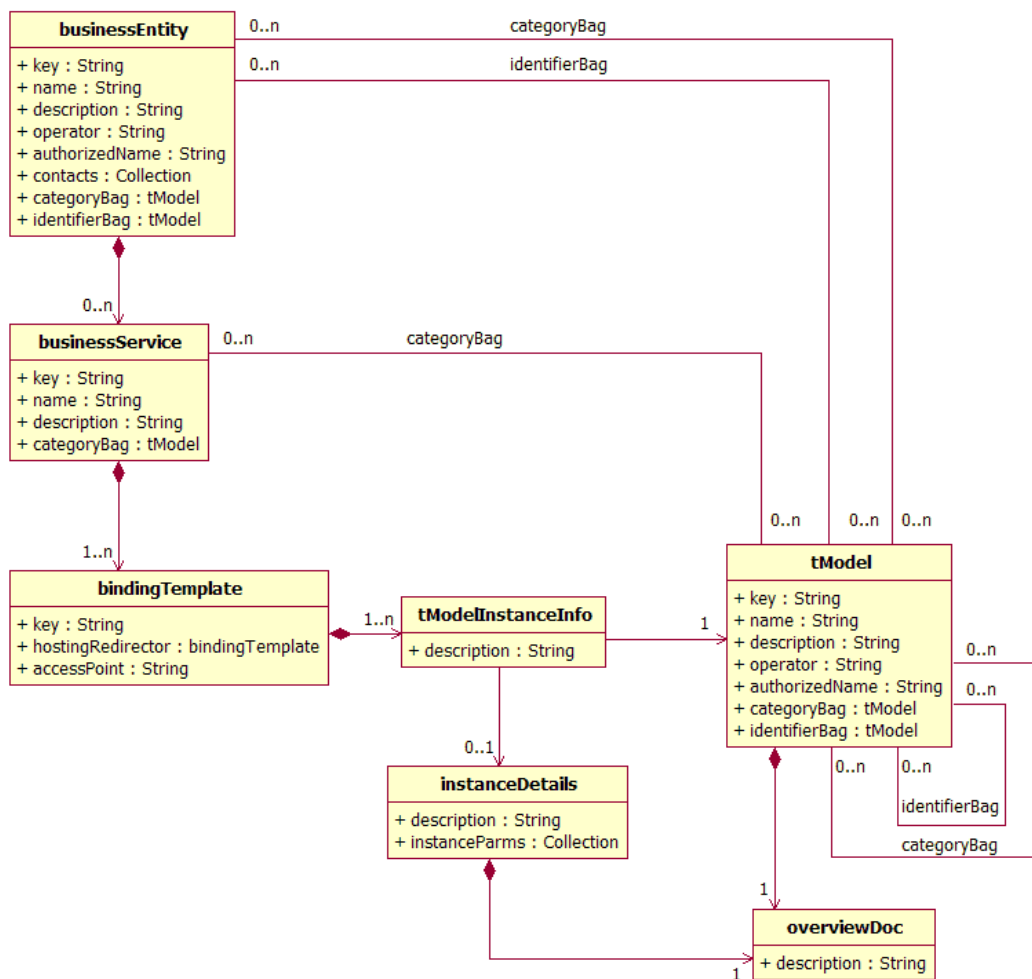


Figure 12: UDDI metamodel

<sup>3</sup> Definition from [http://searchsoa.techtarget.com/sDefinition/0,,sid26\\_gci805760,00.html](http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci805760,00.html)

One important criteria for service discovery is service QoS. However, UDDI standard lacks of expressing Quality information. Hence, there are several proposals to adapt UDDI standard with this new information.

There are some proposals which emphasize the use of tModels to express QoS [29] [30] [31], since it is one of the straightest ways to define quality attributes in this kind of web service registry.

In [30] the authors proposed four methods to store QoS data using tModels. These approaches were presented to OASIS as a contribution to be standardized, although it didn't emerged. The conclusion of the discussion can be checked at [32].

In [31], Xu et al. proposed another extension of UDDI with QoS for web service advertisement and discovery using tModels.

Chi-Chun Lo et al. [29] developed a study of some existing approaches to represent QoS in UDDI in 2008, and presented three new approaches. This study evaluated several proposals including the ones presented in [30] and created three different new proposals to model QoS using tModels.

Here we present four examples of how this extension can be performed from the contributions which provided these examples.

- **Type based approach [29]:** This approach allows multiple QoS values to be stored in one QoS tModel.

```
<tModel tModelKey="uuid:8DD3381C0-06E6-11DC-BC9E-AB0FC98B91F7">
  <name>Cost_sequence_Service</name>
  <categoryBag>
    <keyedReference keyName="types" keyValue="categorization"
      tModelKey="uuid:C1ACF26D-9672-4404-9D70-39b756E62AB4"/>
    <keyedReference keyName="types" keyValue="checked"
      tModelKey="uuid:C1ACF26D-9672-4404-9D70-39b756E62AB4"/>
    <keyedReference keyName="types" keyValue="cacheable"
      tModelKey="uuid:C1ACF26D-9672-4404-9D70-39b756E62AB4"/>
    <keyedReference keyName="QoSValue" keyValue="120"
      tModelKey="uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
  </categoryBag>
</tModel>
```

- **Keyword based approach [29]:** The second proposed approach is to use keyword category system in the UDDI, which uses keyNames to represent different category systems.

```
<tModel tModelKey="uuid:DCD71480-06E6-11DC-BC9E-80DEEE405E59">
  <name>Cost_sequence_Service</name>
  <categoryBag>
    <keyedReference keyName="QoS" keyValue="Cost_Sequence"
      tModelKey="uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
    <keyedReference keyName="QoSValue" keyValue="120"
      tModelKey="uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
  </categoryBag>
</tModel>
```

- **Ontological approach** [29]: This approach is based on transforming ontological elements into tModels, using the patterns from Naumenko et al. [33].

```
<tModel tModelKey="uuid:2287B8A0-F4CB-11DB-BCB5-C11AF03DDF0D">
  <name>Cost_sequence</name>
  <categoryBag>
    <keyedReference keyName="type relation" keyValue="UnmeasurableQoS"
      tModelKey="uuid:2E74D900-ED5E-11DB-AEA8-D51050A91E68"/>
    <keyedReference keyName="aggregatedUnmeasurableQualities relation"
      keyValue="Cost_SearchFlight"
      tModelKey="uuid:2E74D900-ED5E-11DB-AEA8-D51050A91E68"/>
    <keyedReference keyName="aggregatedUnmeasurableQualities relation"
      keyValue="Cost_SelectFlight"
      tModelKey="uuid:CDEAA310-F4C6-11DB-BCB5-9D0C2BE60799"/>
    <keyedReference keyName="hasCompositeType relation"
      keyValue="Sequence_Cost"
      tModelKey="uuid:76650F00-F20E-11DB-AEA8-C92DAF269FE8"/>
    <keyedReference keyName="hasUnit relation" keyValue="NTdollars"
      tModelKey="uuid:6E2B3830-ED7E-11DB-AEA8-CD3AF091E984"/>
    <keyedReference keyName="QoSValue relation" keyValue="120"
      tModelKey="uuid:C5E91160-F4B7-11DB-BCB5-E03DE8A7D016"/>
  </categoryBag>
</tModel>
```

- **Xu et al. approach** [31]: In this approach, the quality metric name is placed in the keyName attribute of the tModel whereas its value is placed in the keyValue.

```
<tModel tModelKey="somecompany.com:
StockQuoteService:PrimaryBinding:QoSInformation">
  <name>QoS Information for Stock Quote Service</name>
  <overviewDoc>
    <overviewURL>
      http://<URL describing schema of QoS attributes>
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:QoS:Price"
      keyName="Price Per Transaction" keyValue="0.01" />
    <keyedReference tModelKey="uddi:uddi.org:QoS:ResponseTime"
      keyName="Average ResponseTime" keyValue="0.05" />
    <keyedReference tModelKey="uddi:uddi.org:QoS:Availability"
      keyName="Availability" keyValue="99.99" />
    <keyedReference tModelKey="uddi:uddi.org:QoS:Throughput"
      keyName="Throughput" keyValue="500" />
  </categoryBag>
</tModel>
```



Another extension of UDDI is UDDIe[34]. This extension is one of the most popular in the literature since it has been used in several frameworks for web service selection [35][36][37].

UDDIe is compliant with UDDI and includes several new features, one of them is QoS, since it is able to provide quality information by adding the quality attributes as a property of the web service in the propertyBag.

An example of a service search using UDDIe is shown bellow. Each requested metric of the web service will be a property in the property bag, which contains the following attributes:

- **propertyName:** the name of the metric
- **propertyValue:** the threshold value for the metric.
- **propertyType:** The type of the value in propertyValue (i.e: number, boolean,... )
- **propertyFindQualifier:** it establishes the operand for the search (i.e: Greater than, less than, equals,..)

```
<find_service businessKey="*****" generic="2.0" xmlns="urn:uddi-org:api_v2">
  <name>*****</name>
  <categoryBag>
    <keyedReference tModelKey="*****" keyName="*****" keyValue="*****" />
  </categoryBag>
  <tModelBag>
    <tModelKey>*****</tModelKey>
  </tModelBag>
  <propertyBag>
    <property>
      <propertyFindQualifier>*****</propertyFindQualifier>
      <propertyName>*****</propertyName>
      <propertyType>*****</propertyType>
      <propertyValue>*****</propertyValue>
    </property>
  </propertyBag>
</find_service>
```

UDDIe has been implemented under the GPL license, and the tool is available on the web, which has facilitated its use in several research works.

# 11 Using a broker with QoS information

---

Another approach to store and manage QoS information is the use of a broker as an intermediary between the client and the service provider. In this approach the broker is responsible to manage and provide to the client the QoS for web services. For that purpose, the broker might use an extended WSDL or store the QoS in an internal component of the framework. It may also retrieve the list of web services from the UDDI or implement its own web service registry.

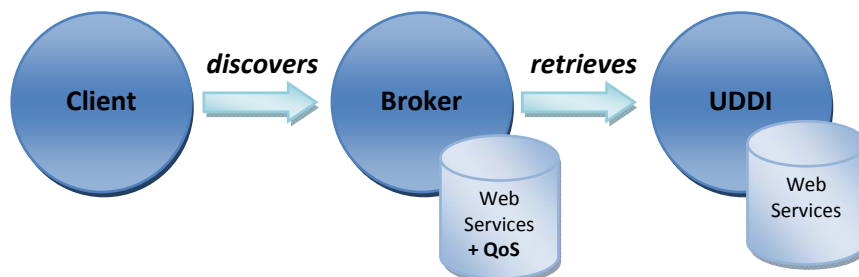
This kind of approach is the most used in the literature of web service discovery. This is because of the flexibility that a broker provides, as a broker can include several components for an enhanced web service discovery.

Most of the brokers proposals include, apart from a web service registry with QoS information, a monitor to obtain those dynamic quality attributes. That is, the broker would be responsible of monitoring the registered services to get and store those dynamic attributes of the web services, whereas static quality attributes such as price should be stored declaratively.

Another component of the broker is the module that implements the ranking algorithm for web service selection taking into account the QoS of the different web services. Obviously, the service user should perform the discovery over the broker instead of using the UDDI directly.

In addition, brokers can be specialized in web service compositions, which include several modules apart from the mentioned above. These kinds of brokers are further explained in the Web service composition chapter.

In [38][39] the authors propose the use of this approach with a broker implementing their own web service registry, replacing therefore the UDDI with QoS information for web service selection. They also include a monitoring tool in order to obtain the real QoS of the web services.



In [36], the authors use UDDI as a web service registry. Additionally it adds other components such as QoS Negotiator, QoS verifier, Admission manager, beyond other components, which are used in posterior stages.

In [40] this approach is used but using an external UDDI, that is, a UDDI that don't belong to the same framework and therefore providing more flexibility to the architecture. In this approach, the presented tool is able to collect the list of Web Services from this UDDI and gather the quality metrics through monitoring.

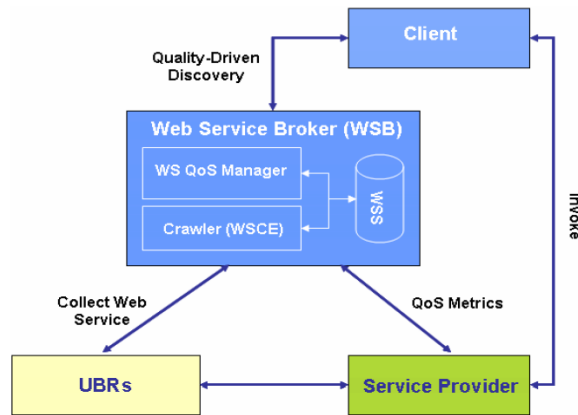


Figure 13: Brokering architecture as presented in [40]

Some works, however, argue that these approaches suffer from scalability limitations. Since brokers only considers those web services which have been registered in their own service registry. Once a service provider would publish a new web service, he should register in several brokers. In order to solve this problem, Serhani et al. presents in [41] a cooperative approach in which the brokers shares the QoS of the web services they manage. Three interaction patterns are studied: random, round robin and the proposed 'cooperative brokers'. The authors conclude that the configuration with the best results is the cooperative policy.

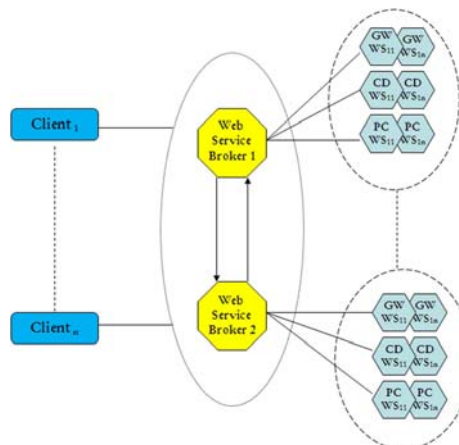


Figure 14: Cooperative approach as presented in [41]

Approach	Web services obtention	UDDI relationship	Monitoring use
<b>Yu et al. [38]</b>	Registered services	Replaces UDDI	Yes
<b>Liu et al. [39]</b>	Registered services	Replaces UDDI	Yes
<b>Serhani et al. [41]</b>	cooperative	Replaces UDDI	Yes
<b>Cardellini et al. [42]</b>	Registered services in UDDI	Uses UDDI	Yes
<b>Badidi et al. [36]</b>	Registered services in UDDIe	Uses UDDIe	Yes

**Table 7: Broker architectures for QoS description**

## 12 Trustworthiness of claimed QoS

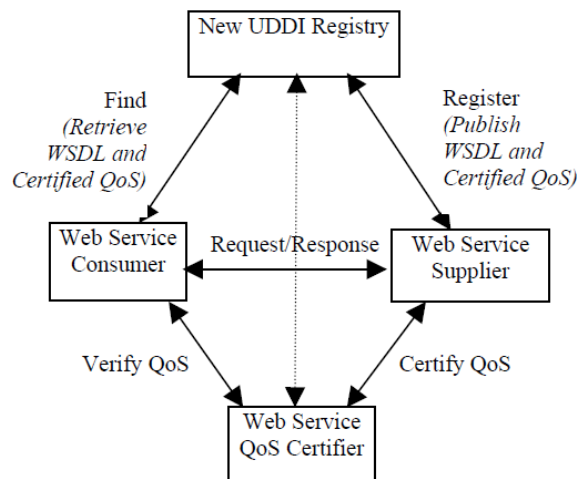
---

Once the Quality of Service is defined by the service provider, there's an imperative need to check that the QoS claimed is consistent with the real Quality of the Service. This is especially significant for those dynamic quality attributes which values may vary on run time.

In order to achieve this point, monitoring or testing is needed in order to compare the obtained values with the claimed ones.

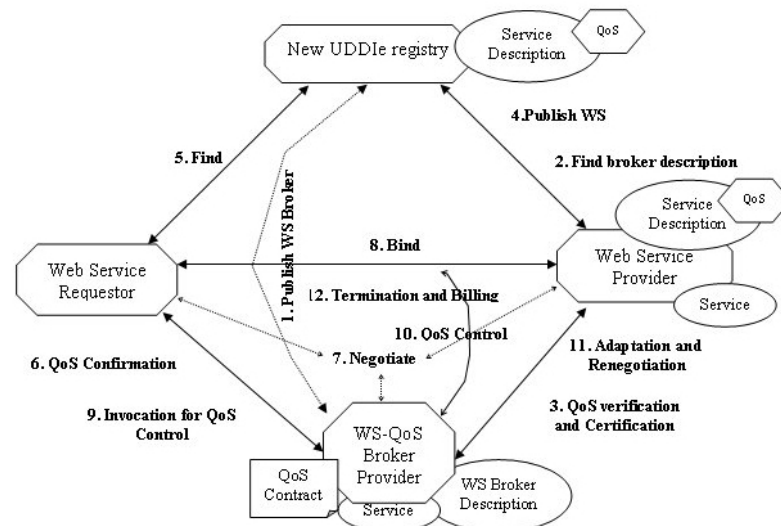
One of the first and most significant works in this area comes from Ran, S. [43]. His main contribution was the addition of a new actor in the web service discovery. This new actor is known as the Web Service QoS Certifier and is responsible to validate that the quality of Service advertised by the provider is the actual Quality of Service. To do so, the following steps are performed:

1. The service provider asks for a certificate to the QoS Certifier for its web service.
2. The QoS Certifier validates the QoS of the web service by monitoring and sends (or not) the certificate to the provider.
3. The service provider publishes its service in the UDDI with the certificate.
4. The service user searches in the UDDI a web service with an specific QoS.
5. The service user chooses one of the resulting web services from the query and validates the authenticity of its certificate through consulting to the QoS certifier.
6. The QoS Certifier monitors the web Service while he is in its registry in case the QoS change.



Based on this work, several authors have proposed frameworks and tools following this structure.

Serhani et al. presented in [35] a broker-based architecture for web service discovery. Among other responsibilities, the broker is used to test the service, compare the obtained values to the claimed ones, and provide a certificate to the service provider (which can be Gold web service, Silver web service or Bronze web service). Finally the service provider can publish his web service with the certificate which demonstrates its trustworthiness. At this point, the broker tests and monitors the web service in order to maintain the certificate.



**Figure 16: Architecture of the broker by Serhani et al. [35]**

Other approaches focus on the experience of the service users instead of using a third-party certifier. In this approach the user is responsible to monitor the web service and provide a ranking of trustworthiness to a reputation manager.

This approach is followed in [31], where the discovery of web services is performed by a broker, named Discovery Agent, which combines the results of an extended UDDI with the Reputation Scores obtained from the Reputation Manager. While the user is consuming the web service, he provides ratings to the Reputation Manager. The ratings are values between 1 to 10, where 10 means extreme satisfaction and 1 no satisfaction at all.

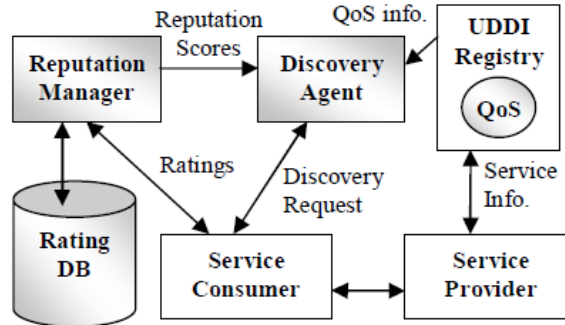


Figure 17: Xu et al. framework presented in [31]

The final reputation score  $U$  is calculated as follows:

$$U = \sum_{i=1}^N S_i \lambda^{d_i}$$

Where

- $N$  is the number of ratings for the service,
- $S_i$  is the  $i^{\text{th}}$  service rating,
- $\lambda$  is the inclusion factor,  $0 < \lambda < 1$ , it's used to adjust the impact value of recent ratings
- $d_i$  is the age of the  $i^{\text{th}}$  service rating in days.

A weighted an adjusted reputation is afterwards multiplied with the values in the ranking of web services.

Another approach is a Learning-Based Trust Model presented in [44]. The author uses a Trust Model expressed as  $T(s1, s2)$ , where  $s2$  is a web service using  $s1$ . This trust model  $T(s1, s2)$  is a combination of three sub functions named  $QTM()$ ,  $MTM()$  and  $DH()$ .

- **$QTM(s1, s2)$**  is the function for evaluating the QoS of the service  $s1$  based on the requirements of the service  $s2$  accordingly to historically gathered data.
- **$MTM(s1, s2)$**  is the function for evaluating the QoS of the service  $s1$  based on the requirements of the service  $s2$  accordingly to the experience of other clients of  $s1$ .
- **$DH(s1)$**  is a function that reward the reputation of  $s1$  if it provides feedback of its experience with its providers. This function is added in order to motivate to give feedbacks.

The Trust Model equation is defined as:

$$T(s1, s2) = w_1^{s2} * QTM^n(s1, s2) + w_2^{s2} * MTM^n(s1, s2) + w_3^{s2} * DH^n(s1)$$

Where  $w_i^{s2}$  represents the weight assigned by the user for each function.

# 13 Monitoring

---

In order to obtain the real quality data of attributes whose values may change on run-time, a monitoring system is needed.

Monitoring can be applied on web services during different stages of a SOA System lifecycle. During the Service discovery, monitoring is useful to obtain the real QoS for a reliable Web Service Selection. Once the service has been selected, monitoring can be used in order to ensure that the service is still fulfilling the promised QoS. Particularly, in Self-healing SOA Systems, when this requirement violation happens, the SOA system might change the failing web service by another one.

Obviously, monitor systems retrieve data only from basic metrics, since derived ones are calculated from the previous ones through a predefined rule. These monitoring systems can be divided in two large groups: Passive Monitoring and Testing.

- **Passive monitoring system** (or for simplicity, monitoring): is a system with the responsibility of sniffing the interaction between Service providers and Service Clients and obtaining therefore the Quality Of the Service minimizing the interaction with the monitored system.
- **Testing system**: is a system responsible for querying requests to a Service Provider and obtaining the responses to ultimately evaluate the QoS.

## 13.1 Passive Monitoring

Passive monitoring is an approach for obtaining the real values of the QoS of the different run-time attributes of a service in a non-intrusive manner. That is, the monitor is just an observer of the interaction between the client and the provider. As a consequence, in a passive monitoring, the monitor doesn't have the control of what requests may be performed, neither how or when.

We have to note that there already exist several commercial monitors for web services. Most of them are components of a SOA manager product. Basically, these monitors focus on monitoring a small set of metrics, commonly, throughput, availability and response time. The data retrieved is usually presented to the user in a statistical chart and some of them also include a module that trigger an alarm if a given threshold value is exceeded.

However, these monitors are specially designed for the product they have been included in, and therefore they lack of the interoperability and extensibility needed for other uses. By the other hand, and despite the existence of final monitoring software, there are still research lines to be explored and solved.

One of the key issues in order to include a passive monitoring system in SOA is about the allocation of the monitor. Accordingly to [45] there are four kind of configurations.

- **Configuration 1:** The monitor, the web service client and the web service belong to the same system.
- **Configuration 2:** The monitor is placed in the web service client
- **Configuration 3:** The monitor is placed in the web service provider
- **Configuration 4:** The monitor, the web service client and the web service belong to distinct systems.

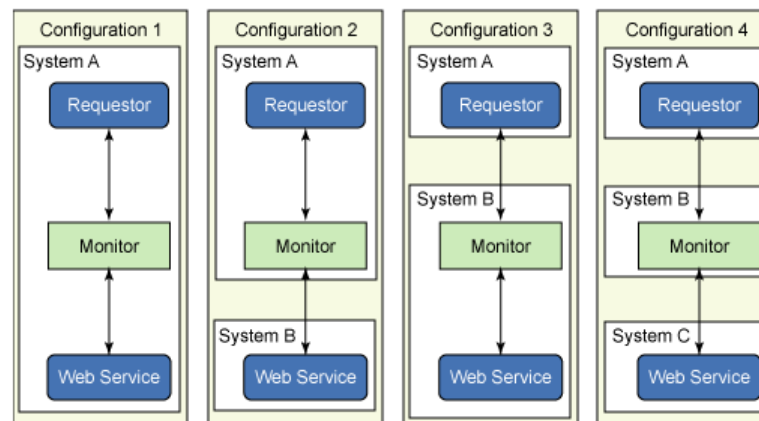


Figure 18: Configurations of monitor allocations as presented in [45]

In [45] the monitor developed by IBM used the Man in the Middle approach, which is the configuration 4 of the figure 18. The process is as follows:

1. The web service client performs the queries to the monitor instead of the web service directly
2. The monitor registers the request and forwards the message to the web service.
3. The web service replies to its sender (which is the monitor).
4. The monitor registers the response and forwards the message to the service client.

This technique is pretty simple, and has gained special attention since the current popularization of Enterprise Service Bus (ESB)[46]. ESB is an emerging architecture for integrating enterprise applications using a message-oriented, event-driven paradigm. It acts as a broker in messaging able to provide, beyond other features, a monitoring system.

Other approaches that follow the Man in the Middle style is by means of the same broker used to discover the web service. This broker may also act as an intermediary for the web service calls in order to obtain the QoS data. This approach is used in Badidi et al.[36]. The service consumer, performs the discovery, negotiation and use of the web service through the web service broker.

Regarding to other configurations, most of popular Web service engines, such as Apache Axis2 and Glassfish, include an execution chain. The execution chain is responsible to manage input and output messages independently of the Business logic, its features include security, error handling, and so forth. These executions chains are implemented in a flexible manner, so they are able to include a monitor tool for the input and output messages.



In [15], Zhou et al. proposed a novel approach to develop monitors in the execution chains of service engines. Based on their previous work on DAML-QoS Ontology [16], the authors proposed the use of the Ontology for the development of a tool that automatically generates monitor as handlers for the Axis engine. These handlers are then located in the handler-chain of Axis, so there's no need to modify the code either for the service client or provider.

By the other hand, there are various works that focuses not only on monitoring but also into correctly identify the source of a given malfunction. When a Service fails, for instance the response time is too high, it might be because of several reasons: the Network bandwidth is low, the Service Provider execution time is high, etc. This is an important area since detecting the real cause of the failure is crucial in Self-Healing SOA Systems.

In [47] Halima et al. proposed the use of discrimination algorithms in order to infer the real cause of the failure of the system, so a proper reconfiguration of the Self-Healing SOA System can be performed.

Also Benharref et al. stated in [48] that in order to track a fault into its originating Web Service, the observation of all, or a significant subset, of web services is required. In this contribution, the multi-observer architectures are shown.

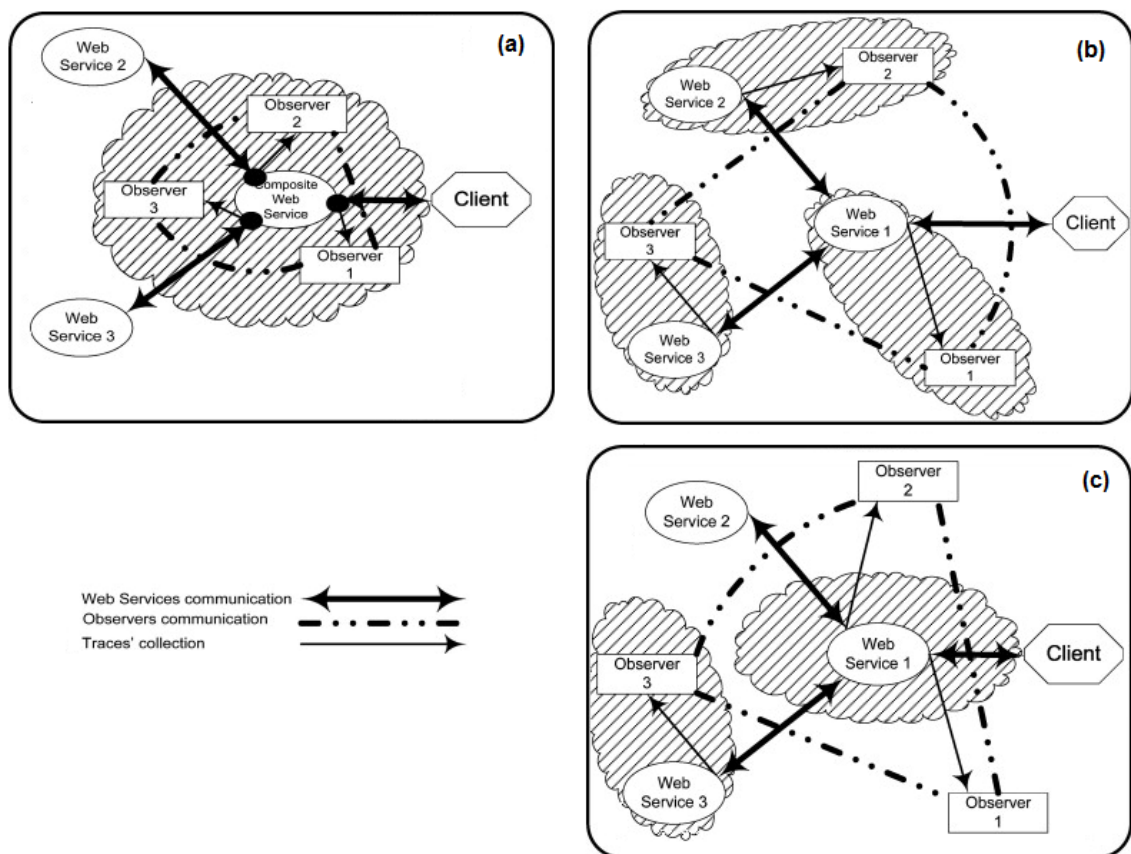


Figure 19: Multiobserver architecture presented in [48]

- In (a) the monitors, named observers, are placed in the client side, the Composite Web Service.
- In (b) the monitors are placed in the server sides, the basic Web Services.
- In (c) a hybrid solution is shown.

Another matter of subject is about the quantity of information monitored. If the web services consumed by the client are third-party services, and the monitor is place on that server side, we have to trust that no manipulation has been performed. By the other hand, if the monitors are placed in the client side, only the client who invokes the web service is aware of the real QoS, which might be suitable for self-healing SOA Systems but not for web service Discovery.

One possible solution is based on the mentioned broker which is the intermediary between several service users and the web services. Another approach is that the broker may test the web services so the user is able to perform the web service calls directly.

There are several solutions and approaches to resolve this problem, but one of the most important was presented in [49] by Jurca et al. In their approach they presented a collaborative QoS notifications by web service clients.

After invoking the service, the client reports the monitored QoS information to a Reputation Mechanism, which combines all the retrieved data from several clients and compares them with the advertised QoS in the Directory where it has been published (currently standard in UDDI). Furthermore it adds payment mechanisms to ensure the trustworthiness of all the parts. The clients are rewarded if they report QoS data and the provider is charged with a penalty payment if the QoS they have advertised is far from the real QoS. This technique is used for trustworthiness QoS.

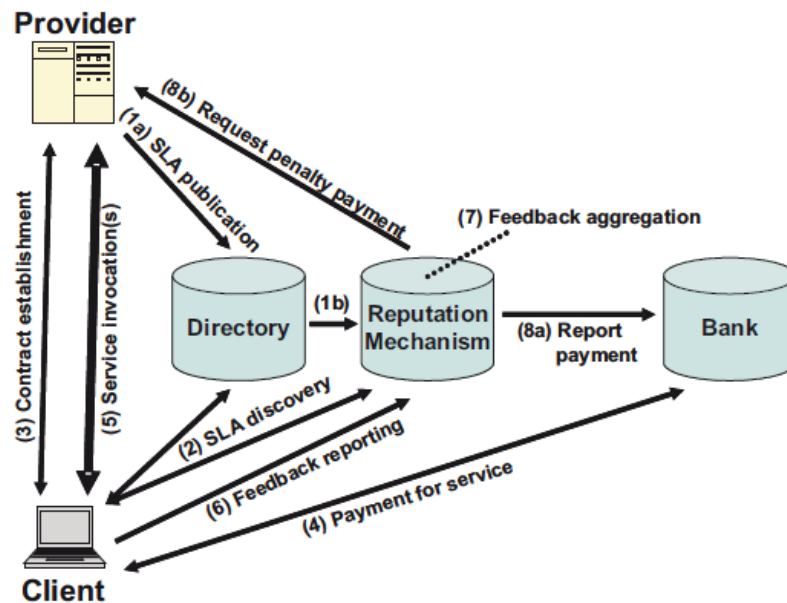


Figure 20: Collaborative monitoring architecture by Jurca et al. [49]

Most of approaches, are however extremely technological dependent, since they rely on the web service engines, programming languages, etc.

In order to add a technological independent layer, Zeng et al. provided in [12] a way to perform monitoring through the use of Event-Condition-Action(ECA) rules. ECA rules are used because as stated by the authors, they frees users from the low-level details of procedural logic. An ECA rule fulfills the following pattern:

Event(eventPattern)[condition] | expression

For instance, an ECA rule could be:

Event(E1::e)[e.a2 > 12 ] | (MC1.serviceID == e.serviceID) MC1.m2 := f1(e)

Which means that when an instance of  $E_1$ , denoted as  $e$ , occurs, if  $e.a_2 > 12$ , then the event is delivered to the Monitor Context  $MC_1$  which has the same serviced, with a metric value computed by  $f_1(e)$ . The advantage of using ECA rules, as stated by the authors, is to provide a high-level language for performing monitoring.

The Nessi Group designed a metamodel of monitoring in [5]. Where they identified all the common parts of a monitoring system. Briefly, as we can see in the figure 21, the monitoring socket is the mechanism to obtain the monitoring data. This data is afterwards checked with a Monitoring Rule (This approach is used to identify possible violations of the Service Level Agreement). These rules are applied to metrics, which in turn express measures of some Service Property ( Quality attribute).

This Service Properties can refer both to an entire service (e.g. Mean Time Between Failure) or to the operations offered by the service itself ( e.g., the operation X has to feature some transactional property).

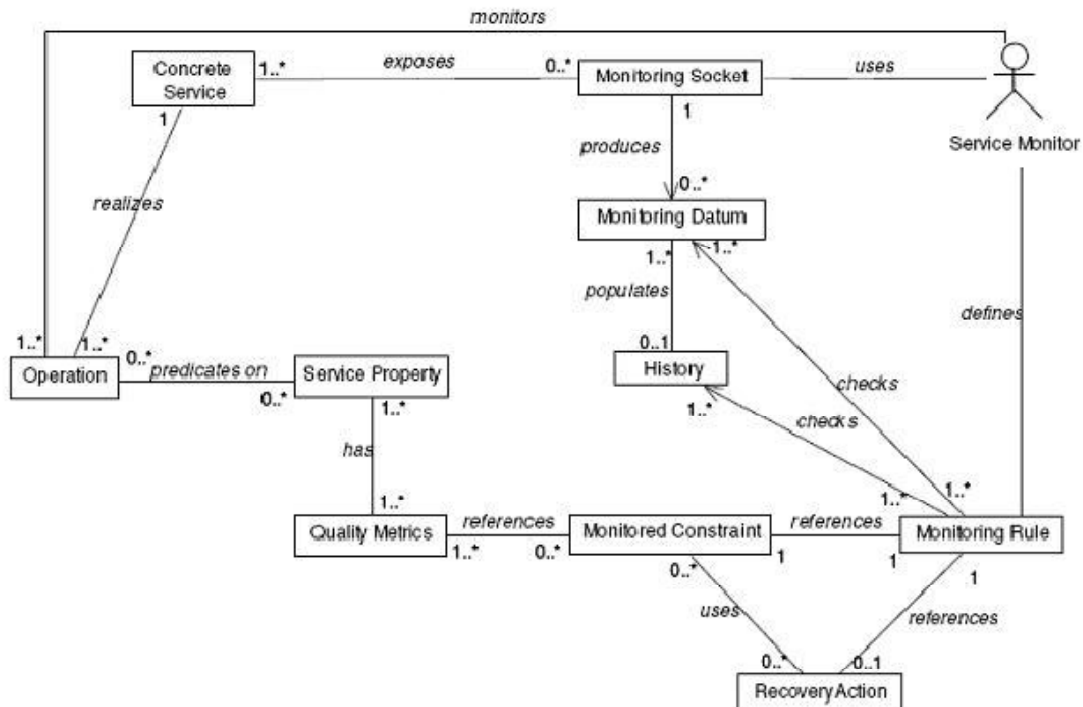


Figure 21: Monitoring metamodel presented in [5]

## 13.2 Testing

Testing systems are those ones which are responsible to perform queries to services in a systematic manner, and retrieve the QoS.

The advantages of testing over monitoring are several:

- The QoS of the web service can be defined before its publication.
- If the number of the users' requests is low, there can be more reliable information through testing.
- It can predict the failure of the system before the users themselves. Providing an excellent way to develop self-healing SOA systems.
- There's absolutely control over the requests.
  - It can determine the capacities of the web service under heavy load circumstances.
  - It can force error conditions in the system.

However, there are also some disadvantages to take into account:

- It relies only on considered requests. So some failures can not be detected if the testing operations have not been defined accurately. (i.e. the web service may reach an infinite loop or a deadlock in a concrete condition).
- If the use of the service implies a cost, and there's no special agreement with the provider, it can be excessively expensive and impracticable.

One of the key issues on testing is how these tests can be developed. Automatic generation of tests is a crucial feature for the success of this technology. Accordingly to Hielscher et al. [50] there are two kind of test case generation procedures:

- **Testing constituent services instances:** The development of the test cases is done through code generation from service descriptions such as WSDL
- **Testing service compositions:** The construction of the test cases is done from composition specifications such as BPEL.

There already exist some commercial tools that provide a helpful support for testing. One of the most well-known is soapUI, a free tool developed by eviware with nearly 1 million downloads. This tool is able to obtain from a WSDL the SOAP requests and add assertions in order to build the test cases.

However, soapUI is unable to build in some cases fully operable SOAP requests, this is because of the fact that some request parameters might have some flexibility such as they might be optional or its cardinality might be greater than. Therefore, the requests are not ready for execution and some manual code writing is needed.

In this aspect, there are some contributions that use soapUI combined with their own components to provide a fully-automated testing framework. Bartolini et al. presented in [51] a tool which focused on fill this soapUI blanks for automatic test case generations.

However, as the same authors explained, it lacked of interdependencies between the operation calls. That is, in a e-commerce web service, we should call the operation addToCart one or more times before calling the operation payOrder.

To solve this problem, Bertolino et al. proposed in [52] a new approach taking in consideration the order in which the operations should be called. To do so they introduced heuristic methods to determine the dependencies between the given operations of a WSDL file. They focus on the parameters and results defined in these operations, and if a parameter of an operation Op1 is the result of a given operation Op2, then Op2 must be called before Op1, since the output of Op2 is part of the input of Op1.

To determine the matching between these parameters and results they used several heuristics, with the lasting result of a directed graphs of the operations.

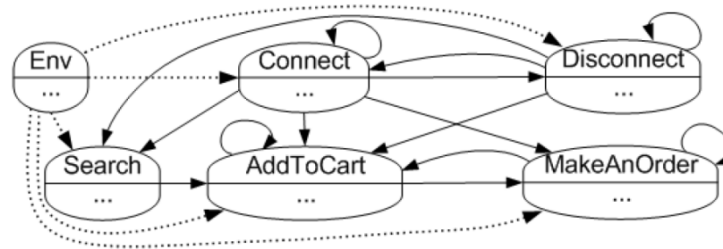


Figure 22: Dependency graph between operations [52]

Also Bertolino et al. presented before in [53] an approach based on the Service Level Agreement and service definitions. In their approach, they develop an empty web service client from the WSDL (there are currently several tools that can perform a transformation from WSDL to a programming code). To fill the empty skeleton, the authors used a wsCodeBuilder which fills the testing code from an SLA standard specification (WS-Agreement) and a WSDD file, which is a deployment file for Axis2.

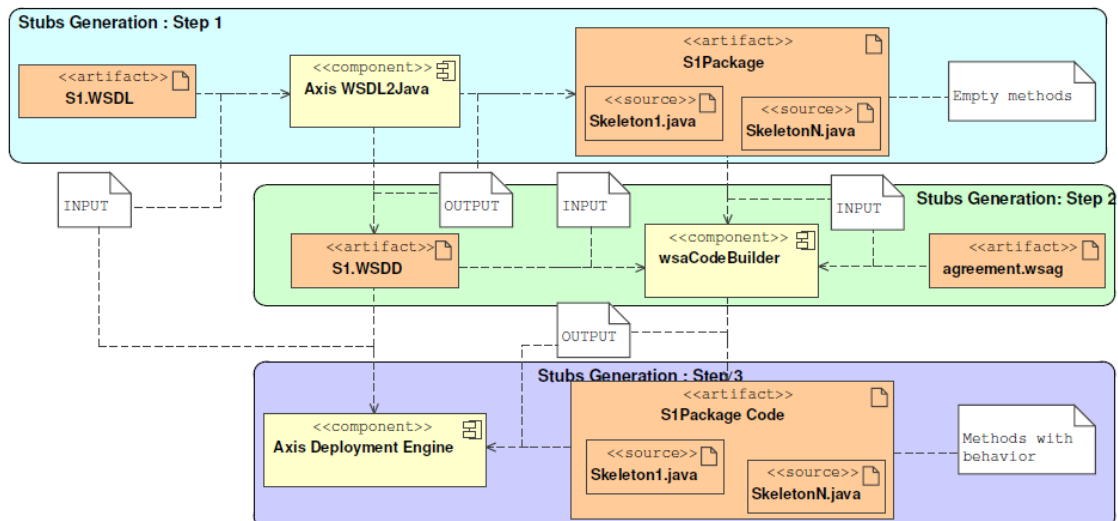


Figure 23: Test generator architecture presented in [53]

Testing can be used in order to determine a failure of a web service component in a composite web service before the service fails on a real request by a user. Although most of works regarding to self-healing SOA Systems are based on monitoring, some authors consider testing an essential issue because it prevents the violation of Service Level Agreements between providers and service users.

In [50] Hielscher et al. introduced a framework, named PROSA, for self-healing SOA Systems using testing techniques. Furthermore, its framework is able to be combined with monitoring techniques, distinguishing then between proactive and reactive adaptations.

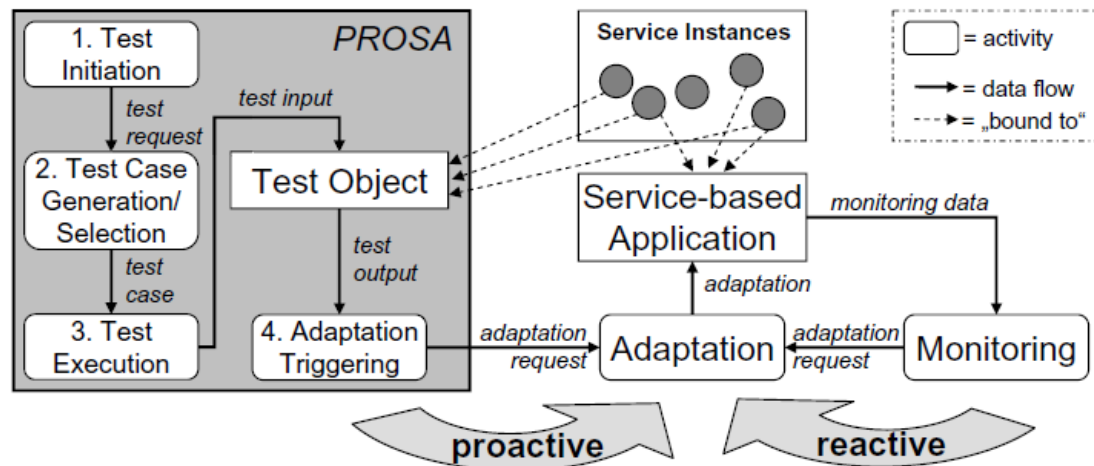


Figure 24: PROSA Framework as presented in [50]

### 13.3 Simulation

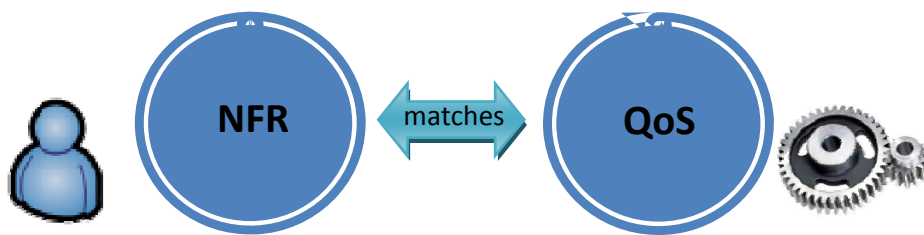
Simulation is an approach to forecast the QoS of a web service without interacting with it but by modeling a hypothetically situation and estimate the expected QoS. Simulations can be useful to determine the QoS of a web service in extremely heavy conditions. Although it's not truly a way to retrieve the QoS precisely, it can be used to predict the QoS in certain conditions when testing is not possible.

This approach has been used in [54] to estimate the QoS in heavy load conditions. The tool they present, named sPac, tests the web service in low conditions to obtain the QoS. This information is afterwards provided to the simulation model, which is responsible to reproduce the behavior of the web service in heavy conditions. The tool they present is focused in Time related metrics.

# 14 Non-Functional requirements for web service selection

Non-Functional requirements should be expressed by the user for selecting the web service that better fulfills them. To do so, these non-functional requirements are matched with the QoS of web services. Most of the literature expresses non-functional requirements as entities that can be directly mapped to quality metrics.

For instance, a NFR about availability can be expressed as: the average availability of the service should be > 99.5%. Since the average availability is a quality metric of the service, the comparison between NFR and QoS is trivial.



However, how these requirements are taken in consideration is a matter of subject.

Some approaches of web service selection, considers the fact of weighting each non-functional requirements. That is, a non-functional requirement with a higher weigh is more important to a less-weighted one. This is due to the fact that the degree of satisfaction of the user shall be better fulfilled if those more important Non-functional requirements are satisfied, rather than the less-important ones. How these weights are applied on the web service selection algorithms is further explained in the next chapter of Web service selection.

Most of the approaches consider simply just putting a weight for each quality metric or attribute, without any interrelation between them. However, there's a curious approach in [55], where they consider the use of what they named a Composition Requirement Tree (CRT). A CRT is a weighted AND-OR tree whose leaf nodes contains quality attributes and edges are represented with weights.

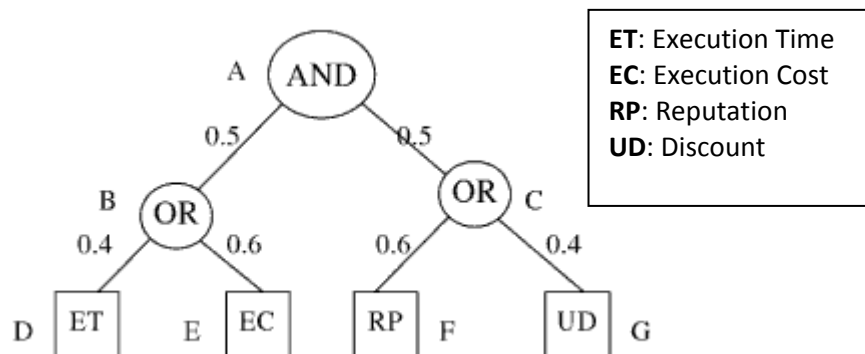


Figure 25: Composition Requirement Tree

As we can see in Figure 25, the user wants to maximize the profit in the same grade (1) ET or EC and (2) RP or UD.

Other approaches consider the use of fuzzy Non-functional requirements. That is, instead of specifying unequivocally the degree of satisfaction for a quality metric (i.e.: Cost per execution must be below 1 €), they use just fuzzy concepts instead. For example, fuzzy non-functional requirements could be that the web service must be very fast, quite secure and cheap.

To do so, some approaches have been proposed. In [56] Lin et al. use a Moderated Fuzzy Discovery Method (MFDM). MFDM consists in two parts:

- **MFDM Framework:** A set of components which includes a Fuzzy Classifier, Fuzzy Engine, UDDI/OWL and Fuzzy Moderator
- **Similarity Aggregation Method (SAM):** algorithm for resolving conflicts emerged from different opinions.

Using this technique, each participant represents his fuzzy perception on one specific criterion with a trapezoidal fuzzy number. Using SAM, the set of opinions are merged to obtain the fuzzy values.

Other techniques, such as the ones proposed in [57][58], use the Intuitionistic Fuzzy Sets (IFS). An IFS is a mathematical concept introduced by Atanassov in 1986 [59]. Intuitionistic Fuzzy Sets are sets whose elements have degrees of membership. For instance, a Web Service with a response time of 10 ms belongs to the sets of 'very fast', 'fast', 'normal', 'slow' and 'very slow' in different degrees. Using this technique, the user can specify his requirements in a fuzzy mode.

An Intuitionistic Fuzzy set A is a set of elements of the following form:

$$A = \{(x, u_A(x), v_A(x)) \mid x \in X\}$$

where

$u_A(x)$ : represents the degree of membership to the element A.  $u_A: X \rightarrow [0,1]$ .

$v_A(x)$ : represents the degree of non-membership to the element A.  $v_A: X \rightarrow [0,1]$ .

There is also a degree of indeterminacy (hesitation margin):  $\pi_A(x)$ .

$$u_A(x) + v_A(x) + \pi_A(x) = 1$$

All this information is needed to transform a fuzzy linguistic term such as 'very good' to an Intuitionistic Fuzzy Number (IFN). For example, an IFN,  $A = [0.5, 0.4]$ , represents that its membership  $u=0.5$ , non-membership  $v = 0.4$  and the hesitation margin  $\pi= 0.1$ .

In [57] they use the IFS not only for specifying the performance required for each of the quality metrics but also for specifying the weights (the importance) of these attributes in a fuzzy manner. After applying an algorithm within the scope of IFS, a ranking is reached.



Linguistic terms	IFNs
Very unimportant (VU)	$[0.1, 0.9 - \pi]$
Unimportant (U)	$[0.3, 0.7 - \pi]$
Medium (M)	$[0.5, 0.5 - \pi]$
Important (I)	$[0.7, 0.3 - \pi]$
Very important (VI)	$[0.9, 0.1 - \pi]$
I do not know (N)	$[0.0, 0.0]$

Table 8: Intuitionistic Fuzzy Numbers for weighting the quality attributes

Linguistic terms	IFNs
Very poor (VP)	$[0.1, 0.9 - \pi]$
Poor (P)	$[0.3, 0.7 - \pi]$
Fair (F)	$[0.5, 0.5 - \pi]$
Good (G)	$[0.7, 0.3 - \pi]$
Very good (VG)	$[0.9, 0.1 - \pi]$
I do not know (N)	$[0.0, 0.0]$

Table 9: Intuitionistic Fuzzy Numbers for the performance of the quality attributes.

# 15 Ranking web services with QoS for web service Selection

---

One of the key issues for adding QoS information to the web services is, in fact, to provide to the service client a way to compare and finally choose the web service that best fulfills his Non functional requirements from those ones which offers the same functionality.

Most of the literature agrees that as web services and SOA systems are increasing in number and popularity, the amount of web services performing the same functionality would also increase in a large number. In this sense, a ranking system that shows the results ordered by the user's requirements helps to determine the best web service for the user. Such as, for instance, a web searcher ranks the web pages accordingly to the user's query.

So, once QoS have been defined for each web service through one of the approaches explained in the previous chapters. Those services should be ranked when a client performs a search with QoS requirements.

## 15.1 Ranking algorithms

Algorithms for ranking web services with QoS information has been a subject of interest for several researchers [13][23][39][40] [60].

One of the challenges in this subject is how can we compare two metrics whose values have completely distinct ranges and units. For instance, how can we combine response time, cost, availability, etc. in a unique value so that the web services can be ranked accordingly to the user's requirements.

The root of this problem precedes the existence of SOA. Therefore, the solutions applied to web services are based on the most important contributions in this area, which is called the Multiple Attribute Decision Making (MADM)[61].

Most of the contributions, builds a matrix of QoS attributes [13][39][60] in this matrix each row represents a web service, whereas each column represents a single quality metric.

$$E = \begin{bmatrix} q_{1,1} & q_{1,2} & \dots & q_{1,j} \\ q_{2,1} & q_{2,2} & \dots & q_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ q_{i,1} & q_{i,2} & \dots & q_{i,j} \end{bmatrix}$$

In order to be able to compare values from different ranges, the data stored in the matrix is normalized from 0 to 1 so different quality metrics can be compared. There are several ways to perform this normalization. In [61], the Linear and Vector normalization are explained, other kinds of normalizations such as Max-Min normalization are also used, as in [60].

Notice that the normalization is done differently if the metric must be minimized or maximized. That is, a user might be interested on minimize the cost of the web service but maximize its reliability. For the correctness of the proposed algorithms, the normalization has to take into account this factor.

- **Linear normalization:** In maximization, simply divides the values of each different quality metrics by its maximum value. For minimization, the same formula is applied to  $\frac{1}{q}$ , which results into the minimum divided by the value of the quality metric.

Minimizing	Maximizing
$Norm(q_{i,j}) = \frac{q_{i,j_{min}}}{q_{i,j}}$	$Norm(q_{i,j}) = \frac{q_{i,j}}{q_{i,j_{max}}}$

- **Vector normalization:** In maximization, this normalization divides each of the values of the quality metrics by its norm. For minimization, the same formula is applied to  $\frac{1}{q}$

Minimizing	Maximizing
$Norm(q_{i,j}) = \frac{1}{q_{i,j} \sqrt{\sum_{i=1}^m \frac{1}{q_{i,j}^2}}}$	$Norm(q_{i,j}) = \frac{q_{i,j}}{\sqrt{\sum_{i=1}^m q_{i,j}^2}}$

- **Max-Min normalization:** Max-min normalization subtracts the minimum value of an attribute from each value of the attribute and then divides the difference by the range of the attribute.

Minimizing	Maximizing
$Norm(q_{i,j}) = \frac{q_{i,j_{max}} - q_{i,j}}{q_{i,j_{max}} - q_{i,j_{min}}}$	$Norm(q_{i,j}) = \frac{q_{i,j} - q_{i,j_{min}}}{q_{i,j_{max}} - q_{i,j_{min}}}$

Since some attributes might be more relevant for the client than other, the client might be able to assign weights to each quality metric. This is simply done by specifying a weight for each quality metric and multiplying the values of these quality metrics for the weight. In order to maintain the range between [0,1], the weights are also defined between [0,1]

Finally, services are ranked accordingly to the sum of their normalized QoS values.

$$E = \begin{bmatrix} q_{1,1} & q_{1,2} & \dots & q_{1,j} \\ q_{2,1} & q_{2,2} & \dots & q_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ q_{i,1} & q_{i,2} & \dots & q_{i,j} \end{bmatrix} \xrightarrow{h_{i,j} = w_j \left[ \frac{q_{i,j}}{\max(N(j))} \right]} E' = \begin{bmatrix} h_{1,1} & h_{1,2} & \dots & h_{1,j} \\ h_{2,1} & h_{2,2} & \dots & h_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ h_{i,1} & h_{i,2} & \dots & h_{i,j} \end{bmatrix}$$

$$WSRF(ws_i) = \sum_{i=1}^N h_{i,j}$$

**Example of ranking values for web services using a weighted linear normalization, presented in [13]**

Some other works includes the use of user requirements to determine the best Web Services. Instead of summing the different normalized metrics of the Web Services, they calculate the difference between the user's requirements and the QoS of the Web Service [11] [60]

One common algorithm is calculating the Euclidian Distance between Service QoS and user's non-functional requirements once these values have been normalized. [60].

The Euclidian Distance between two vectors  $t_i = (t_{i1}, \dots, t_{ik})$  and  $t_j = (t_{j1}, \dots, t_{jk})$  is given by the following formula:

$$\text{dis}(t_i, t_j) = \sqrt{\sum_{h=1}^k (t_{ih} - t_{jh})^2}$$

In [23] another approach is presented without using a normalized matrix to rank web services accordingly to their QoS. One of the points of this approach is that they do not use a unique QoS value for a Quality Attribute but a set of values from different kind of sources.

Let  $A_i$  denote  $i^{\text{th}}$  attribute of a service  $S$ , and  $U$  the user of  $S$ .

- $E(A_i)$  represents the quality of  $A_i$  that  $U$  expects
- $P(A_i)$  is the actual quality of  $A_i$  perceived by  $U$
- $R(A_i)$  is the quality rating that  $U$  give to  $A_i$ .
- $D(A_i)$  is the advertised quality rating provided by the Service Provider

From all this information, the algorithm extracts a unique  $Q(A_i)$  which represents the overall rating of the Quality Attribute.

$$Q(A_i) = f(k, D(A_i)) + \sum_{j=1}^k w_j * R_j(A_i)$$

Where:

- $k$  represents the number of users of that service.
- $f$  is a function (e.g.,  $\frac{1}{k} D(A_i)$ ), as more users, the smaller role of  $D(A_i)$
- $w_j$  represents a weight for that Quality attribute stated by the user  $j$ .

The overall rating for the Service is then calculated with the following formula:

$$QS(S) = \sum_{i=1}^k w_i * Q(A_i)$$

Another approach without using normalized matrix is presented in [37] by Tabein et al. In this approach they consider the comparison of the different web services with the same functionality just based on a given Quality attribute, obtaining then a set of ranked web services named WSP. Using two or more WSPs the ranking is performed. This work was extended in [62] using a learning automata. That is, it takes into account previous scores of services by applying the concepts in Stochastic Learning Automata. Each of the services is compared to its new sorting rank; the services with the same position are rewarded whereas the other ones receive a penalty.

## 15.2 Ranking through Fuzzy requirements

Some works focus on ranking web services through using fuzzy sets of QoS instead of the actual values of their Quality Attributes. This is, the user may ask for a fast web service, instead of a web service with a response time less than 0.5s.

In [63], Wu et al. presents a Bayesian Network model in order to evaluate a service with fuzzy QoS. The proposed model is a naïve Bayesian network in which the root node denotes the service capability whereas the leaf nodes denote users' requirements for each QoS property. In order to perform the evaluation, as a first step, a discretization of continuous variables must be performed, afterwards, the model is able to calculate the probabilities of the service capabilities, from poor, moderate, good to excellent. For instance, when a service consumer is searching for books plane ticket services and his availability is high, the resulting conditional probability functions might be:

P(SC = "poor" | OP="plane", AV="high")  
P(SC="moderate" | OP="plane", AV="high")  
P(SC="good" | OP="plane", AV="high")  
P(SC="excellent" | OP="plane", AV="high")

# 16 QoS in composite web services

---

A composite service is a software service implemented through the composition of software services<sup>4</sup>. This kind of web services, instead of being developed from scratch, uses and compose other web services in order to provide a new and more complex service to final user.

The building and execution of this web services can be carried out in the following 4 phases [55]:

- 1) The individual tasks of the composite service are identified
- 2) The suitable web services for each task are discovered
- 3) The optimal composition of these web services is identified
- 4) The execution phase executes the assigned web services.

The QoS of a web service may be strongly affected by the QoS of the various Web services it uses. There are several studies which analyze how service components constraints the QoS of their consumer: how we can calculate the QoS of a service from its components, identification of bottlenecks, etc.

In order to resolve this problem, the use of service composition languages is mandatory. Most of these languages have been based on workflows, and although they are not intended just to be focused on QoS, they play an important role in the evaluation of QoS in SOA Systems. The use of workflows is reasonable since a business task can be wrapped with a Web service interface.

At the moment, there have been several efforts on enhancing workflow systems to support Quality of Service management, as well as methods to compute and predict QoS. One of the advantages of integrating QoS models with workflows systems is a better capability to support self-adaptive SOA systems.

## 16.1 Brief history of Service Composition Languages

In 2001, IBM introduced WSFL (Web Services Flow Language)[64], a language to represent the workflow of composite services. The representation of their models can be in both graphical or XML mode.

The basis of this language is the use of a directed-graph where:

- each node represents a web service
- each directed edge between nodes *A* and *B* indicates that *A* uses the service *B*.

By the same time, Microsoft presented XLANG, an extension of WSDL to provide both the model of an orchestration of services as well as collaboration contracts between orchestrations [65]

After releasing their respective languages, IBM and Microsoft decided to combine them into a new language, BPEL4WS. In April 2003, BPEL4WS 1.1 was standardized through the OASIS

---

<sup>4</sup> Definition from the Nexof Glossary: <http://www.nexof-ra.eu/?q=node/187>

group. Soon later, in 2004, the name BPEL4WS was changed for WS-BPEL in order to fulfill the convention of starting with WS- all web service related components [66].

One of the advantages of WS-BPEL is that is an executable language. That is, using a WS-BPEL engine, a service developer can have its web service working with the WS-BPEL model.

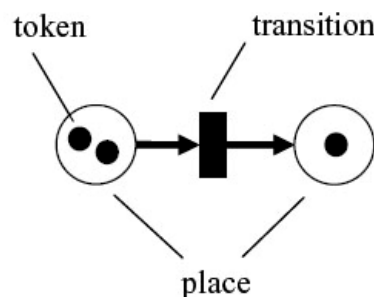
```
<?xml version="1.0" encoding="UTF-8"?>
<process
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:print="http://www.eclipse.org/tppt/choreography/2004/engine/Print"

  <!--Hello World - BPEL program -->

  <import importType="http://schemas.xmlsoap.org/wsdl/"
    location="../../test_bucket/service_libraries/tppt_EnginePrinterPort.wsdl"
    namespace="http://www.eclipse.org/tppt/choreography/2004/engine/Print" />
  <partnerLinks>
    <partnerLink name="printService"
      partnerLinkType="print:printLink"
      partnerRole="printService"/>
  </partnerLinks>
  <variables>
    <variable name="hello_world"
      messageType="print:PrintMessage" />
  </variables>
  <assign>
    <copy>
      <from><literal>Hello World</literal></from>
      <to>.value</to>
    </copy>
  </assign>
  <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />
</process>
```

**WS-BPEL example: "Hello world"**

Other languages used in the literature for representing Composite web services are not technological solutions but mathematical modeling languages for the description of discrete distributed systems. One of the most used in this area are Petri Nets[67]. A Petri Net is a directed bipartite graph which has three basic components: places (where tokens reside), transitions and directed arcs.



**Figure 26: Petri Net components**

In the Web services area, the states of the web service are represented as places and the operations as transitions. Full details of how web service compositions are designed in Petri Nets can be checked at [68].

These languages are the basis for the calculation of the QoS of composite web service from their components. Most of the contributions in this area are based on these standards.

## 16.2 QoS aggregation.

QoS aggregation is the computation of the Quality of Service of a composite Web Service from the QoS provided by its components. A considerable number of contributions have focused in this area in order to get a set of formal mathematic functions that automatically calculates the QoS of composite services.

One of the first and most relevant works, was presented in 2002 with a very significant proposal using WSFL to evaluate the throughput of services [69]. This proposal stated that the edges (representing a web service use) should be labeled with the *relative visit radio*, this label indicates the average number of times a node is visited for a visit of its parent.

As we can see in Figure 27,  $V_{AB}$  would be the number of times node  $B$  is visited per visit to node  $A$ . In the same way,  $A$  requests to  $C$  a number of  $V_{AC}$  requests per each time  $A$  is visited; and so forth.

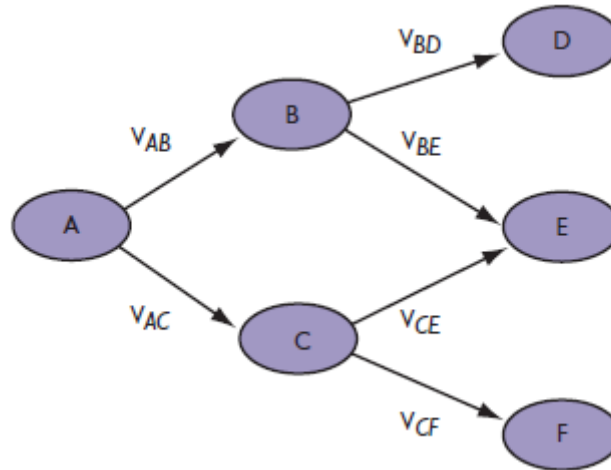


Figure 27

Given the whole graph of Figure 27, the author states that the throughput  $X_A$  ( the number of requests per second that  $A$  is able to handle) must be less than the throughput of any of its components / *relative visit radio*. The equation of the given figure is:

$$X_A \leq \min \left\{ \frac{X_B}{V_{AB}}, \frac{X_C}{V_{AC}}, \frac{X_D}{V_{AB}V_{BD}}, \frac{X_F}{V_{AC}V_{CF}}, \frac{X_E}{V_{AB}V_{BE} + V_{AC}V_{CE}} \right\}$$



This equation is useful to identify which service is the bottleneck of the service composition, and therefore, the service component that we might change in order to increase the throughput of the entire service.

Future works have been focused on other quality attributes such as response time or availability, beyond others. By the other hand, the mentioned approach needs to have in knowledge the rates of use of the different web services, which as they may vary on run time, it has some limitations.

In order to calculate the QoS aggregation of several web service components, we need to know how this services are composed. In [70] four basic workflow patterns were used: sequential, parallel, conditional and loop. Another kind of construct, called discriminator or fault tolerant was introduced by Cardoso [71]. Resulting with the following 5 patterns:

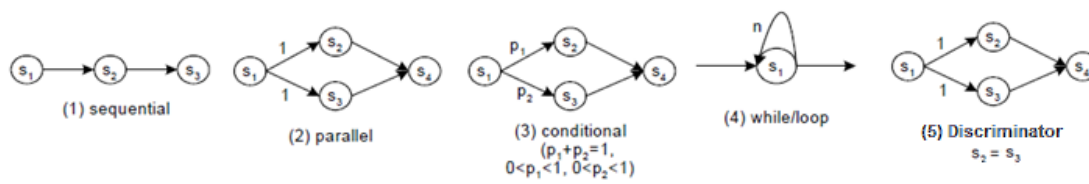


Figure 28

- 1) **Sequential:** a web service is executed after the execution of a predecessor one
- 2) **Parallel:** two or more web services are executed simultaneously.
- 3) **Conditional:** There's a condition which establishes what web service will be called.
- 4) **While/loop:** The same web service is called several times.
- 5) **Discriminator:** Two or more web services performing the same functionality are executed at the same time. Only the response of the faster one is taken.

Since any kind of web service composition can be expressed as a combination of these patterns, the functions to calculate the QoS for each of these patterns is adequate to calculate the QoS of the composite web service.

In [72], the author presents the aggregate functions for computing the QoS of several attributes ( Response Time, Cost, Availability and Reliability). Although it states the existence of 4 kinds of composition flow Models (it doesn't take into account discriminator). The aggregate functions presented are applicable only for sequential mode.

In [73], the same aggregate functions were presented with the same results. Nevertheless, the author also added an aggregate function to calculate the Bandwidth.

Attributes	Aggregate Function
<b>Response Time</b>	$T = \sum_{i=1}^n (T_{si} + T_{ni})$ <p>Where  <math>T_s</math>= Response Time (without network time)  <math>T_n</math>= Network transmission time.</p>
<b>Service Cost</b>	$C = \sum_{i=1}^n C_{si}$ <p>Where  <math>C_s</math>= The cost of the service</p>
<b>Reliability</b>	$R = \prod_{i=1}^n R_i$ <p>Where  <math>R</math>= Reliability ( The probability that a request is correctly responded)</p>
<b>Availability</b>	$A = \prod_{i=1}^n A_i$ <p>Where  <math>A</math>= Availability (The probability that a services is available)</p>
<b>Bandwidth</b>	$B = \min_{1 \leq i \leq m} (B(s_i))$ <p>Where  <math>B</math>= Bandwidth</p>

Some algorithms for Optimizing web services may be interested in converting these products into summations in order to have linear functions. To do so, the following logarithmic function can be used [72]:

$$\log\left(\prod_{i=1}^n X_i\right) = \sum_{i=1}^n \log(X_i)$$

Nevertheless, Hwang et al. identifies in [74] that most of current approaches to calculate QoS aggregation consider QoS attributes as if it were deterministic values. However, because of the nature of web services, it's more realistic to take into account the randomness of these QoS attributes. Therefore it states the need of modeling them in a probabilistic form.

In this new situation, there are some relevant issues to take into account. For instance, taking into account the probabilistic forms of Response time, it's stated that the Average response Time of a Web Service (WS) composed of two Web Services ( $ws_1$ ,  $ws_2$ ) in a parallel mode is not the maximum of the average response time between  $ws_1$  and  $ws_2$ , as it would be in a deterministic value. Furthermore, it can not be calculated only with this given information.

In order to solve this problem, the paper defines a Probabilistic Model of WS-QoS based on modeling discrete random variables, stating a probability for each one of these values. For

instance, for Response Time, the use of time intervals is proposed to convert the continuous function into a discrete function.

**Example with response Time.**

$f_x(0) = 0$	Probability that response time is 0 .
$f_x(1) = 0,2$	Probability that response time is in (0,1]
$f_x(4) = 0,6$	Probability that response time is in (1, 4]
$f_x(7) = 0,2$	Probability that response time is in (4, 7]

In the same paper, how to calculate from two random variables their addition, multiplication, maximum, minimum is presented. So it can operate with them to get the different Quality of Services in the different model flows.

However, most of approaches are based on an abstract flow model. Despite of making those approaches technologically independent, they lack of providing a solution based on a standard technological support. Recently, in [75], Mukherjee et al. presented an approach using WS-BPEL standard to compute the QoS of a composite web service. Hence, providing a tool for computing QoS for web services described in WS-BPEL. However, their approach only focused on three quality attributes: reliability, response time and cost.

The Qos aggregation is used as a basis for optimizing composite web services.

In Table 10 we present the contribution given by each paper in QoS aggregation.

	[72]	[73]	[74] ( <i>using probabilistic models</i> )	[75]
<b>Response Time</b>	sequential	sequential	sequential, parallel, conditional, loop, discriminator	using WS-BPEL
<b>Service Cost</b>	sequential	sequential	sequential, parallel, conditional, loop, discriminator	using WS-BPEL
<b>Reliability</b>	sequential	sequential	sequential, parallel, conditional, loop, discriminator	using WS-BPEL
<b>Availability</b>	sequential	sequential	-	-
<b>Bandwidth</b>	-	sequential	-	-
<b>Fidelity</b>	-	-	sequential, parallel, conditional, loop, discriminator	-

**Table 10**

# 17 Optimizing composite web services.

---

The selection algorithm which is constrained by more than two dependent parameters is a known NP-complete problem. This is formally demonstrated in [76] which reveals that the Multiple Constrained Path selection (MCP) problem, which is a known NP-complete problem, can be mapped as a concrete case of the QoS-assured Service Composition (QSC) problem, which is needed for the optimization of the QoS of Composite Web Services.

Once there are a large number of web services available to use in a composite web service, the performance of selecting those web services can be affected. Therefore, the use of efficient algorithms for composition is needed.

Most of approaches use algorithms that don't ensure the best web service composition in terms of QoS but is probabilistically one of the bests. This is because of the fact that Self-adapting SOA Systems have gained a lot of interests, and since adaption must be performed on run-time, the interval of time spent for retrieving the solution is critical.

Optimization of web service compositions is closely related to Web service Selection. However in this new scenario, more than one web service has to be taken into account, constructing new paths of interaction between them, and how this new interactions affects the QoS of the composite web service.

As in QoS aggregation, the need of using a Service Composition Language is imperative. In [77] Xiong et al. used Petri Nets in order to model composite Web Services. For the web service selection with multiple quality attributes they use the Multi-attribute Decision Making (MADM)[61] for comparing elements with separate attributes.

Since these approaches needs of heuristic algorithms two key aspects are in consideration:

- The efficiency of the algorithm
- The closest result to the optimal one

In [78] Berbner et al. proposed a heuristic algorithm that uses backtracking capabilities, accordingly to the authors, the algorithm reaches as close as 99% of the optimal solution spending less than 2% of time of an exact algorithm in their experiments.

In [79] Zhang et al. proposed another heuristic algorithm based on the idea of constructing the convex hull of related points for approximation, whose optimal percentage achieved was above 70%.

In [80] the probabilistic aspects of QoS have been taken into account for their heuristic. Furthermore in this approach the authors advocate not only for QoS of composite services but the QoS of the final application being used by the user. To do so, they add an extra layer has been added in QoS Management

Another possible approach is to apply genetic algorithms for web service composition optimization. Such a work is described in [81]. However, this approach is much slower than integer programming ones, in my opinion this makes genetic algorithm inappropriate for Self-healing SOA Systems, but they might be useful for other purposes such as building a composite web service on design time rather than on execution time.

To do so, the authors provide a genome, which is an array representing the abstract services that are used in the composite web service. The crossover operation is the standard two-points crossover, while the mutation operator randomly selects an abstract service and randomly replaces the corresponding concrete service with another one available. Then the QoS of the composite web service is computed which may have some restrictions in turn.

One of the advantages of this approach is that it can handle not-linear aggregation functions, whereas integer-programming needs to convert those functions into linear functions, using a logarithmic function on the QoS attribute instead of the actual QoS attribute (this is further explained in QoS aggregation chapter).

# 18 Conclusions of the review

---

Going back to the question of the research, and after the development of the synthesis of the literature in this review, we will try to answer the following issue in this chapter.

**Is there a framework (or a set of them) which supports the discovery, selection, adaption and monitoring of web services based on the Quality of Services?**

There is still lack of a consensus to use a unique Quality Model in order to define the basis to describe QoS information in Web Services. Although several approaches have been presented, none of them have had a big impact in the research area. Nevertheless, the recent efforts of the OASIS Group to provide an extensive Quality Model (currently a working draft), may introduce a solution of the problem.

To express the QoS of Web Services, there is not enough in having a common Quality Model, but there is also the need to use a common ontology. Semantic Web Services have gained a lot of attention in the past few years. The most used Web Service ontology is OWL-S, however it lacks of QoS information capabilities. To resolve this problem, some extensions have been proposed. From the contributions presented, OWL-S extensions and WSMO are those ones that have gained more attention in the community and might be used in a recent future.

To describe these web services, there is also the need to extend current standards WSDL and UDDI.

Regarding to WSDL extensions, there is no standard on how to describe the QoS of a Web Service. The most common approach is the use of annotations, which is a concept currently supported by the standard, but there are also other methods that extends the WSDL definition. Although there have been a big number of proposals for extending WSDL files, none of them have gained a distinguishing interest. Again the OASIS Group, and based on their work in Quality Model, may change this status if their proposal of WSDL extension success. However, in their approach, the WSDL file achieves an extremely huge size, which might be a problem for its adoption.

Regarding to UDDI extensions, most of the approaches use tModels in order to define the QoS of Web Services. In this area, we have to mention that, in contrast with the other QoS topics, one technology has started to become known and used, UDDIe. This UDDI extension is available as a technological solution and has already been employed in several works from several researchers.

Since the QoS may vary on run-time we need the use of monitoring tools to ensure the correctness of the described Quality of Service. There already exist commercial tools that implements passive monitoring approaches, however, they lack of extensibility to be included in other QoS frameworks. Regarding to testing, there are tools that partially build test cases; the approaches and prototypes that try to implement a fully-automated test case generator are still on on-going research. Simulation is also a new method that might be useful in some cases.

For the Web Service discovery, there is the need to rank web services accordingly to their QoS and the preferences of the user. Most of approaches are based on mathematical solutions of a wider area, and therefore, it is in a mature state of the research. However there are still

two issues to handle: none of the prototypes developed have been used widely and there is not a benchmark that compares the different ranking algorithms with respect on the interests of the user. By the other way, new challenges may arise when we talk about Fuzzy Requirements.

With reference to Composite Web Services, there are two areas that have gained special attention in the research: QoS aggregation, and optimizing the composition of Web Services.

QoS aggregation is the calculation of the QoS of the Composite Web Service from its components. The functions to do so are widely known and since most of the literature agree to use one common composition language (either Petri Nets or WS-BPEL) it is a mature area. However, new challenges take place if we take into account the probabilistic behavior of Web Services.

Regarding to Web Service Optimization, this is a widely known NP-Complete problem, so most of proposals focus on using heuristic algorithms that get close to the optimal solution. There is still a need of a benchmark of the different proposed algorithms, and to our knowledge, there is still no tool able to perform this composition changes on run-time.

With the goal of developing a framework able to perform composition changes on run-time, and particularly for Self-healing SOA Systems, starts the 2nd part of this Master Thesis. These systems need several components; one of the most important is a reliable Monitoring Tool which is, in fact, also functional for the Web Service discovery.

# Monitoring tool

## 19 Introduction

---

In order to add Quality information in the different stages of the lifecycle of web services, trustworthiness is a key issue. Retrieving the real quality values of dynamic attributes (that is, those ones that may vary on run time) is achievable through a system that uses passive monitoring or testing approaches.

The project of the development of a monitoring tool, named SALMon, was started by David Ameller in 2008, when the main generic specifications and architecture were defined. At this point, SALMon became part of my research work with the aim of extending the work started by David.

With the continuance of the project, the challenges of this new development stage have been:

- Refine the initial architecture.
- Extend the monitoring functionalities.
- Implement and make available the monitoring tool.

With the goal of including the tool in two separate frameworks for:

- Self-Healing SOA Systems
- Web Service discovery with QoS information

As a result of this work, a first prototype of SALMon have been implemented and it is available at: <http://appserv.lsi.upc.es/salmon/>

### 19.1 Self-Healing SOA System (MAESoS)

Adaptability is a key feature of Service-Oriented Architecture (SOA) Systems. Self-Healing SOA systems must evolve themselves in order to ensure their initial requirements as well as to satisfy arising new ones. In this situation it is necessary not only ensuring that the system fulfils its requirements but also that every service satisfies its own requirements, and dynamically adapting the system when some of them cannot be ensured. To do so, there's the need of tools for monitoring and adapting SOA Systems at run time.

The tool specification began with the goal of being included as part of a framework for Self-Healing SOA Systems named MAESoS.



MAESoS is an on-going research framework designed by the research group GESSI (Group of Software Engineering for Information Systems) at UPC, and the Institute for Systems Engineering and Automation (SEA) at the Johannes Kepler University (Linz, Austria). As a member of the UPC team that is contributing in this project we have to mention also Lidia Lopez.

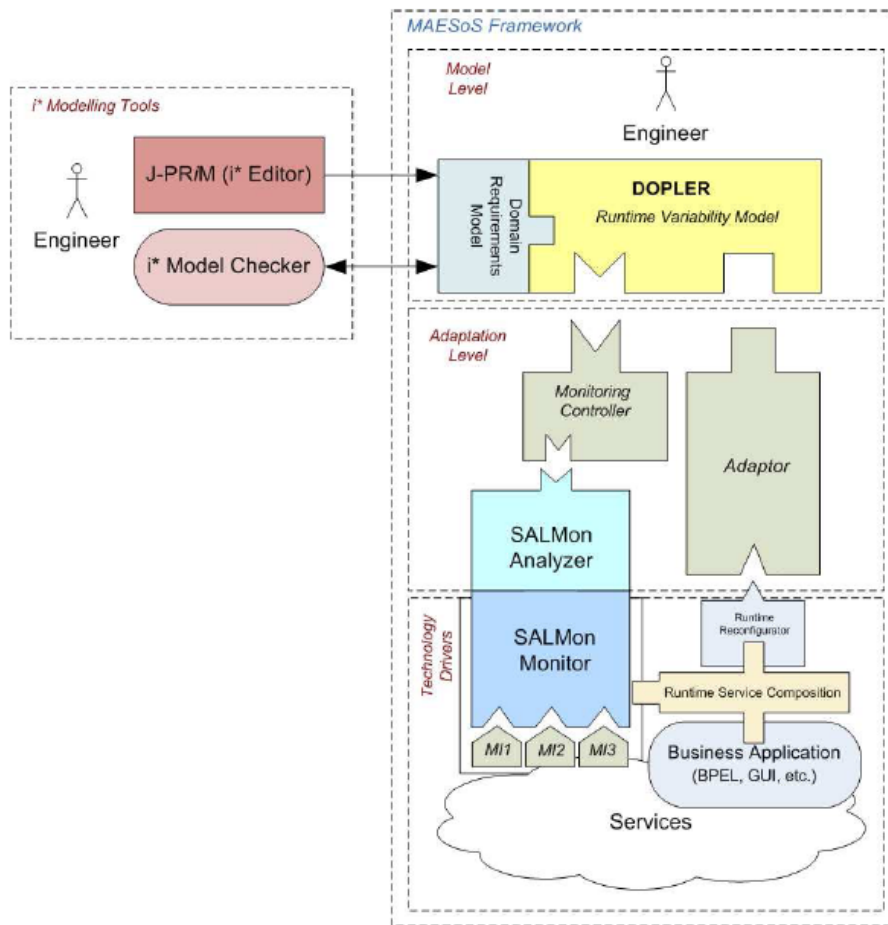


Figure 29: MAESoS architecture

## 19.2 Web Service discovery with QoS information (WeSSQoS)

Although the initial scope of the monitoring project was to include SALMon only in the MAESoS framework, during the development of this Master Thesis, another objective emerged: the inclusion of the monitor as part of a framework for Web Service Discovery with QoS information, named WeSSQoS.

WessQoS is another on-going research framework developed by Oscar Cabrera with contributions of my research and the supervision of Xavier Franch and Jordi Marco, as part of the activities of the GESSI research group and CENIDET research center of Mexico. My work in this aspect has been to adapt the monitor and define with Oscar a common interaction mechanism to provide dynamic quality information to the tool.

As a result of these works, WessQoS is available at <http://appserv.lsi.upc.es/wessqos>

# 20 Requirements

---

Most of requirements were identified during the first SALMon definition carried out by David Ameller. Nevertheless, we present them in order to provide a self-contained document of the tool. By the other way, during the development of the Master Thesis, we added Functional Requirement #5.

## 20.1 Functional requirements

<b>F. Req # 1</b>	<b>The platform shall monitor one or more quality metrics for one or more Web Services.</b>
<b>Description &amp; Rationale:</b>	
<i>In order to obtain the real quality metrics of web services which may vary on run-time, the monitor should be able to retrieve the behavior of its registered web services</i>	
<b>Fit Criteria:</b>	
<i>The monitor is able to monitor a set of quality metrics for any web service.</i>	

<b>F. Req # 2</b>	<b>The platform shall allow monitoring each quality metric in more than one way.</b>
<b>Description &amp; Rationale:</b>	
<i>Since passive monitoring and testing have their own advantages and disadvantages, the monitor should be able to use both of these techniques.</i>	
<b>Fit Criteria:</b>	
<i>The monitor is able to obtain the quality metrics of a web service either through passive monitoring or testing.</i>	

<b>F. Req # 3</b>	<b>The platform shall be able to compute derived quality metrics.</b>
<b>Description &amp; Rationale:</b>	
<i>Basic metrics are monitored whereas derived ones are obtained through a function of a set of metrics. Since some conditions or requirements may apply to derived metrics, the platform should be able to compute derived quality metrics</i>	
<b>Fit Criteria:</b>	
<i>The platform is able to compute a set of derived metrics from the given basic metrics.</i>	

<b>F. Req # 4</b>	<b>The platform shall be able to connect to an external decision system to take automatic actions.</b>
<b>Description &amp; Rationale:</b>	
<i>In order to support Self-adaptive SOA Systems, the platform should be able to connect to an external tool, named decision system, able to switch a failing web service for another one.</i>	
<b>Fit Criteria:</b>	
<i>The platform is able to notify to a decision system when a web service is not fulfilling the expected QoS.</i>	

<b>F. Req # 5</b>	<b>The platform shall be able to connect to an external Web Service discovery system to provide QoS information of the web services.</b>
<b>Description &amp; Rationale:</b>	
<i>In order to support Web Service discovery with QoS information, the platform should be able to connect to the discovery system and provide the monitored and computed quality metrics of the web services</i>	
<b>Fit Criteria:</b>	
<i>The platform is able to provide to a web service discovery system the set of monitored or calculated metrics of web services.</i>	

<b>F. Req # 6</b>	<b>The platform shall be able to write reports about the incidences..</b>
<b>Description &amp; Rationale:</b>	
<i>In order to better manage the quality of web services, the monitor should be able to provide the information about the behavior, concretely on incidences, of the web services.</i>	
<b>Fit Criteria:</b>	
<i>For the given monitored web services, there exists an available report of the incidences of low QoS.</i>	

## 20.2 Non functional requirements

<b>NF. Req # 1</b>	<b>The platform shall be extensible.</b>
<b>Description &amp; Rationale:</b>	
<i>Since it will be a component-based platform, we need enough extensibility capabilities for accepting new components.</i>	
<b>Fit Criteria:</b>	
<i>The platform is able to accept new components.</i>	

<b>NF. Req # 2</b>	<b>The platform shall be developed as a service.</b>
<b>Description &amp; Rationale:</b>	
<i>Developing the platform as a service would facilitate its integration into existing SOA Systems.</i>	
<b>Fit Criteria:</b>	
<i>The platform is developed as a service under the paradigm of SOA</i>	

<b>NF. Req # 3</b>	<b>The monitoring process should not interfere with the performance of the monitored services</b>
<b>Description &amp; Rationale:</b>	
<i>To provide accurate and reliable QoS information, it is necessary that the monitoring process should not interfere with the performance of the monitored services.</i>	
<b>Fit Criteria:</b>	
<i>The performance of the web service is not altered because of the monitoring</i>	

**NF. Req # 4** The platform should be adhered to standards.

**Description & Rationale:**

*In order to be easily incorporated with other frameworks and accepted for the community, the development of the platform should prioritize the use of the standards*

**Fit Criteria:**

*The platform should be compliant with current standards such as WSDL and SOAP. Also the metrics should follow be derived from an standard Quality Model (such as ISO9126)*

**NF. Req # 5** The platform shall be able to work with continuous data flows.

**Description & Rationale:**

*Because of the nature of the information treated, the platform should be able to work with continuous data flows*

**Fit Criteria:**

*The platform is able to obtain and process the data continuously*

**NF. Req # 6** The platform shall be secure.

**Description & Rationale:**

*Since the platform is dealing with private data, there is the need to ensure the privacy of this information*

**Fit Criteria:**

*A user can access certain information only if he is authorized to do so.*

## 21 Attributes and metrics

---

To identify the different quality metrics for the monitoring tool, David Ameller used as a basis the standard Quality Model ISO9126. From this Quality Model, the characteristics able to be monitored are functionality and efficiency related characteristics. Maintainability, portability, usability and reliability are groups of characteristics that can not be monitored, basically because they are types of characteristics that are not supposed to change in time.

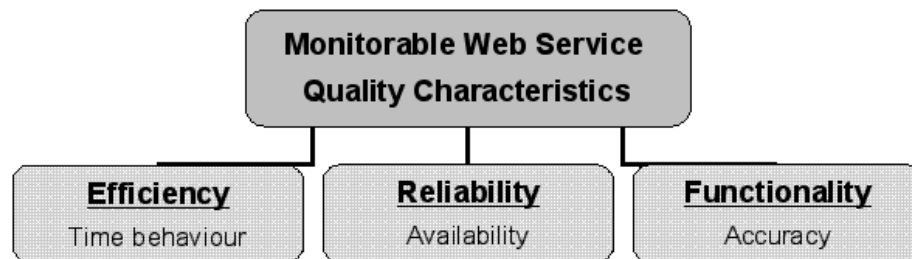


Figure 30: Sub-Quality Model used in SALMon

Each of these characteristics is composed of a set of attributes:

- **Time behavior:** is the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
  - **Response time:** The first attribute is the Response Time. The user will want to monitor his agreed response time .
  - **Execution time:** The second attribute is the Execution Time. It measures the time that a Web Service takes to execute a certain job (a method, a process...).The user will want to monitor his agreed execution time.
- **Availability:** Is the degree to which a system is operable. The user will want to ensure that the service is available in the agreed time.
- **Accuracy:** is the capability of the software product to provide the right or agreed results or effects with the needed degree of precision.

For the new version of SALMon specification, and in order to develop an efficient monitoring technique, we have classified whether if these attributes were applicable to a concrete operation or to the whole web service. The importance of the classification is based on how a single testing invocation can gather different quality attributes.

		Metric	Description
Web Service's attributes	Availability	Current availability	It measures the current availability of a Web Service all the time or in slots of time.
		Accumulative availability time	It measures in percentage how much time a Web Service has been available since it has been monitored for a first time.
		Accumulative unavailability time	It measures in percentage how much time a Web Service has not been available since it has been monitored for a first time.
		Average recovery time	It measures in seconds the average time that a Web Service needs to be available again (It considers unavailability).
	Response time	Current response time	It measures the current response time in milliseconds to access to a Web Service.
		Minimum response time	It measures which is the lowest response time in milliseconds to access to a Web Service.
		Maximum response time	It measures which is the maximum response time in milliseconds to access to a Web Service.
		Average response time	It measures which is the average response time in milliseconds to access to a Web Service.
Operation's attributes	Execution Time	Current execution time	It measures the current execution time in milliseconds that a Web Service takes to execute a job. (response+execution)
		Minimum execution time	It measures which is the minimum execution time in milliseconds that a Web Service takes to execute a job.
		Maximum execution time	It measures which is the maximum execution time in milliseconds that a Web Service takes to execute a job.
		Average execution time	It measures which is the average execution time in milliseconds that a Web Service takes to execute a job.
	Test functionality	Current functionality compliance	It measures the current functionality compliance of a Web Service.
		Parameter Accuracy factor	It divides the number of accepted correct type parameters passed to a Web Service method by the number of expected parameters for this method (1 is better).
		Result Accuracy factor	It divides the number of correct type results returned by a Web Service method by the number of expected results of this method (1 is better).
		Fault factor	It divides the number of faults that a Web Service method had generated by the total times that this method had been executed (1 is worse)

**Table 11: Quality metrics of SALMon**

Each metric has associated information in order to calculate them (i.e: to compute the Average Response Time, we need the time interval). Most of this information was already defined for most of the metrics in the first version except those ones regarding to accuracy.

Accuracy (sometimes named Test functionality) is a challenging quality attribute to measure, and therefore we need some special information to retrieve their metric values. The problem arise when we have to identify whether a response is or not correct regarding to a request.

To ensure if a response of the service is correct, it's necessary to evaluate and check that it fulfills the expected structure for the given request. To do so, we propose the use of patterns so the response can be checked through the given pattern.

This is not a new issue in web services, since WSDL schemas are used in order to check if the structure of both requests and responses are well built. However, it lacks for support to evaluate the content. To achieve this, the use of another pattern system is needed (for instance, Regular Languages).

## 22 Platform Architecture

Our platform is a SOA System itself, and following the SOA paradigm, is composed by the following services: Analyzer, Decision Maker and Monitor. The Measure Instruments as shown in the figure are part of the Monitor service. It also uses a service to store the monitoring data, a service for authentication and authorization. The presented architecture structure has had not important changes since its inception. However, on the contrary, the internal structure of its components has experimented significant modifications.

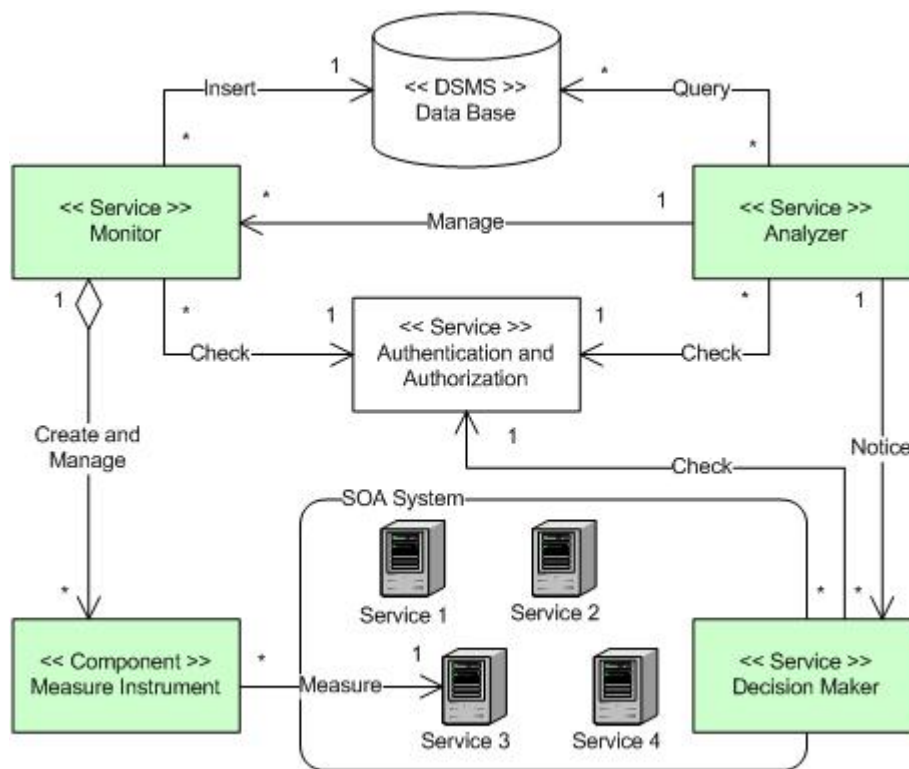


Figure 31: Deployment diagram

- **Monitor service:** It uses Measure Instruments to get the information about QoS. Measure Instruments are components generated by the Monitor to communicate with the services in order to get the monitoring information that is relevant for computing the chosen metrics. This information is stored in a data base and is also rendered to the Analyzer.
- **Analyzer service:** It is responsible of checking for SLA violations in concrete SOA Systems: when a violation is detected, it is notified to the Decision Maker service of the affected SOA System. To attain its goal, the Analyzer manages a Monitor service.
- **Decision Maker service:** It selects the best treatment to solve the incidences detected by the Analyzer in a concrete SOA system. Each Decision maker is related with one (and only one) SOA System.

## 22.1 Monitor

The Monitor service is composed of several Measure Instruments for the same SOA System. Measure Instruments are components used to get all the basic metrics of the selected quality attributes. Derived metrics will be obtained from them. These components are responsible of bringing the measures to the Monitor, which has the responsibility of maintaining this information updated. The update process was in an initial conception defined as an iterative call to each Measure Instrument in different intervals of time, saving the results in a database.

However, in order to add more independence between measure instruments, we redesigned from scratch the measure instruments as independent threads that are created when a new metric must be monitored. Hence, creating a multi-thread platform able to handle those metrics independently. This subject is further explained in the *Implementation details* chapter

Currently, we implemented these measure instruments using the testing approach (although we plan on implementing passive monitoring shortly). Since testing is an intrusive approach, Measure Instruments have the responsibility to minimize the number of interactions performed with the monitored service. To perform the tests, we have implemented the monitor using the following approach:

We implemented all service-related metrics sharing the same time interval between calls and hence the same measure instrument calls. In this new situation, if the user needs to measure the basic metrics current availability and current response time of a service, there is not an extra contact with the web service being monitored.

The same happens in operation-related metrics. However, to monitor the metrics of an operation, further information details are needed. In particular, the name of the operation and at least one valid SOAP message request. To test the accuracy of the operation, we need also to have the knowledge to determine whether if a response message is valid or not. To do so, we use patterns of correct SOAP message responses. If the message response meets the pattern, we say that the response is a valid message, otherwise we say that it's invalid. This approach however, cannot state if the data given in the response is reliable but if it is well-structured and consistent accordingly to the request.

By the other hand, monitors have the particularity that they might be used in other contexts, such as in Web Service Discovery. Since it was not in the initial formation of the tool, the monitor definition has also been extended in order to be able to provide this new usage for a Web Service discovery framework. This has been achieved through adding operations that return the QoS of the web services that are being monitored.



## 22.2 Analyzer

The internal architecture of Analyzer was already defined by David Ameller, and its development has had few changes from its starting point.

The Analyzer manages Monitors and checks for SLA violations in concrete SOA systems. When a violation is detected it is notified to the Decision Maker of the affected SOA system. In general the Analyzer can handle multiple SOA systems using one Monitor and one Decision Maker for each one. The use of Decision Maker services is optional but if they are not used, the SALMon user is limited to monitoring and SLA violation detection.

The SLA can be configured manually with the interface provided by the Analyzer ( Although there are plans to parse SLA standards). We understand SLA as a set of conditions that must be true in some time interval. A condition is composed of the evaluated metric, a relational operator and a value for the comparison (i.e. "Current Response Time < 100ms" is a condition that must be true for the specified service during the specified time interval).

Defining time intervals is important since some conditions are relevant in a specific interval of time or date. For instance, it could be possible that a service is required to be available in a specific timetable, but we could agree that this service can be temporally unavailable in a scheduled time for maintaining purposes.

The Analyzer is also responsible to compute the desired derived metrics from basic metric values stored by the monitor service.

## 22.3 Decision Maker

The Decision Maker service has a repository of treatments and alternative services for a concrete SOA system. It will automatically select and execute the best treatment for the reported incidences.

Because the kind of job of this service and for security reasons, it is preferred to place the service in the concrete SOA system where it is working.

The Decision Maker has the following responsibilities:

- To take actions when something goes wrong in the SOA system.
- To write reports of the incidences with the taken actions.

The development of this component is done by the JKU Team.

## 22.4 Users

There are two kinds of user in SALMon architecture, normal user and administrator user. Note that the kind of user is set for each service, for example one user could be administrator of an Analyzer and a Decision Maker but only with normal access to a Monitor service.

The normal users will be limited to the finality of each service while the administrators have extra functionalities: management of the access to the service, establishment of restrictions in services (e.g. set the maximum number of Measure Instruments in a Monitor), and set the interconnection between services.

The user of the Analyzer service must be authenticated before start working with it, this user must be authorized to use the Monitor services. If the user wants to use Decision Maker services, he/she must also have enough rights in each of the monitored SOA systems. Measure instruments are property of one monitor so they don't need authentication.

To be able to use SALMon the user will need to have at least a normal user level in one Analyzer and in one Monitor. The Decision Maker is optional but if the user has no access to the Decision Maker or the SOA system has no Decision Maker service, it will be limited to monitoring.

## 23 Use Cases

The SALMon has 6 use cases showed in the following figure:

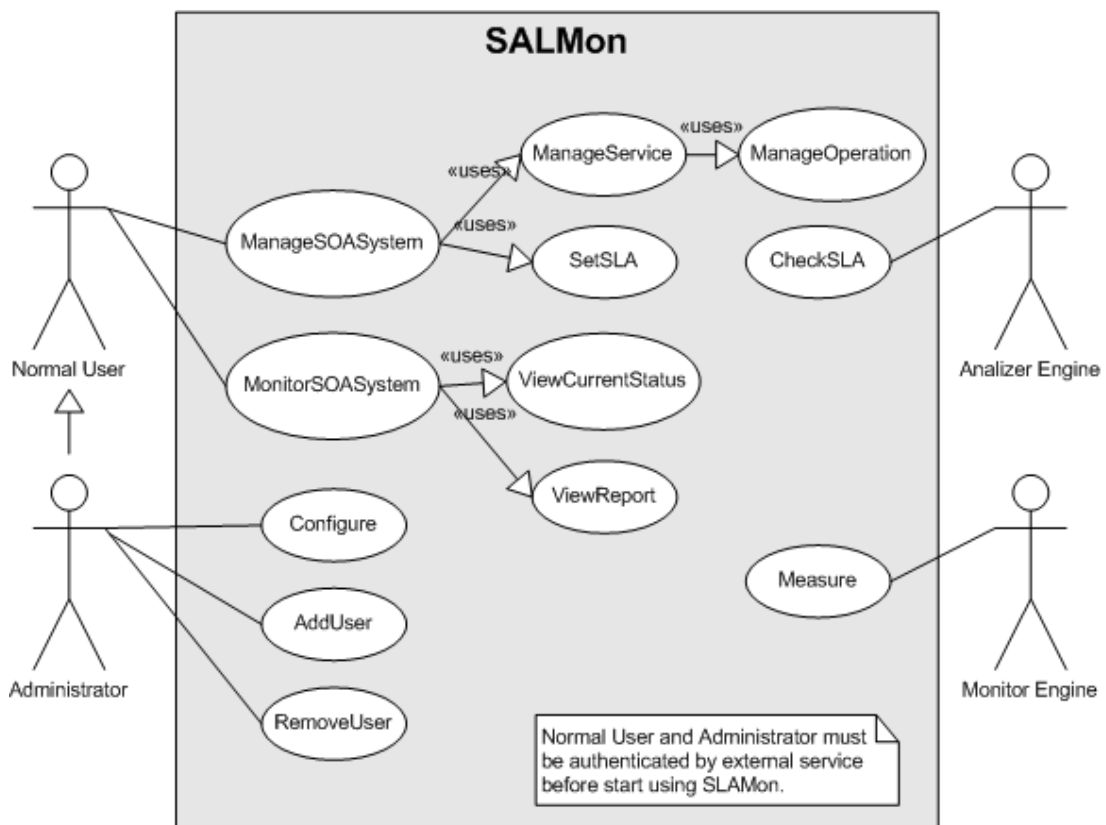


Figure 32: SALMon use cases

Two use cases for the normal users that represent the user interaction with the system, two use cases for administration proposes, and two more used by virtual actors to represent the internal behaviour of the system.

The first two use cases are composed by a group of use cases in order to make the diagram more understandable.

## 24 Sequence diagrams

In this section there are the sequence diagrams for the use cases described in the previous section.

As we can see in the Figure 33, we have the main use case for normal users, which is ManageSOASystem. Note that the login process is not described but it must be done before this use case. In this new version, we have extended ManageSOASystem with ManageOperation methods in order to be able to retrieve operation-related metrics.

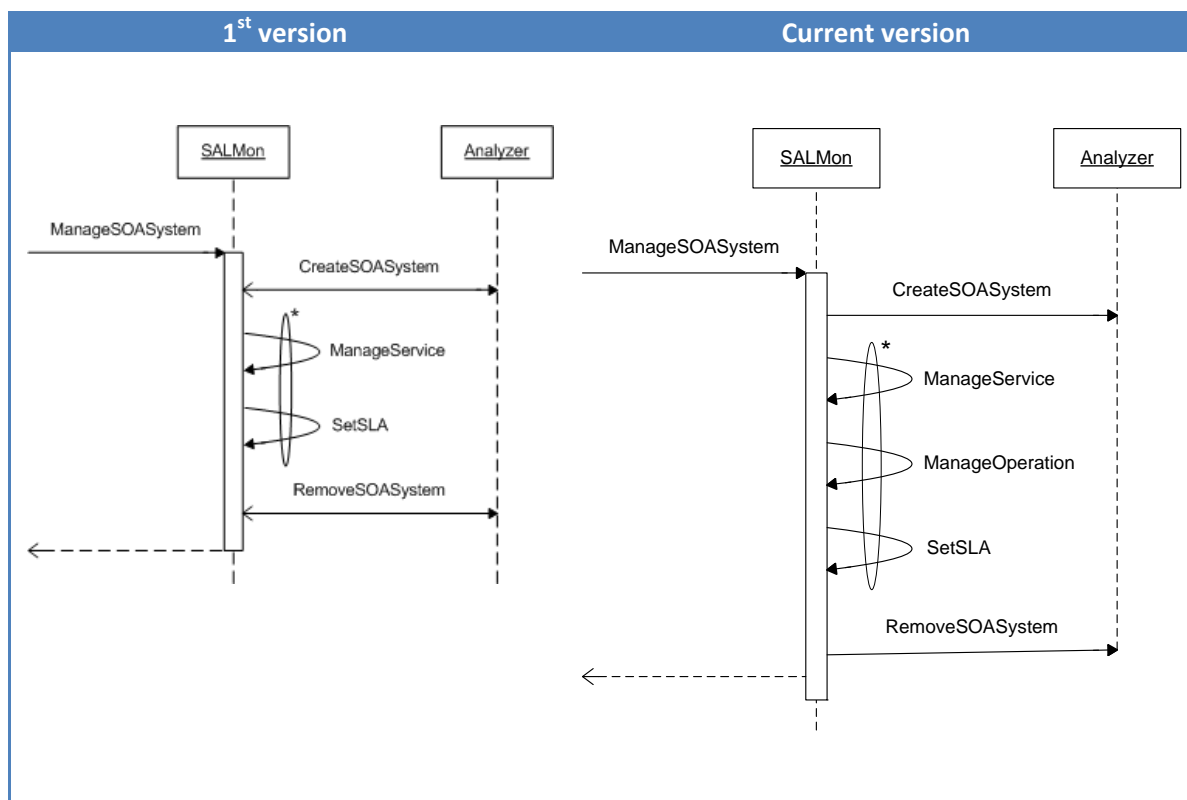
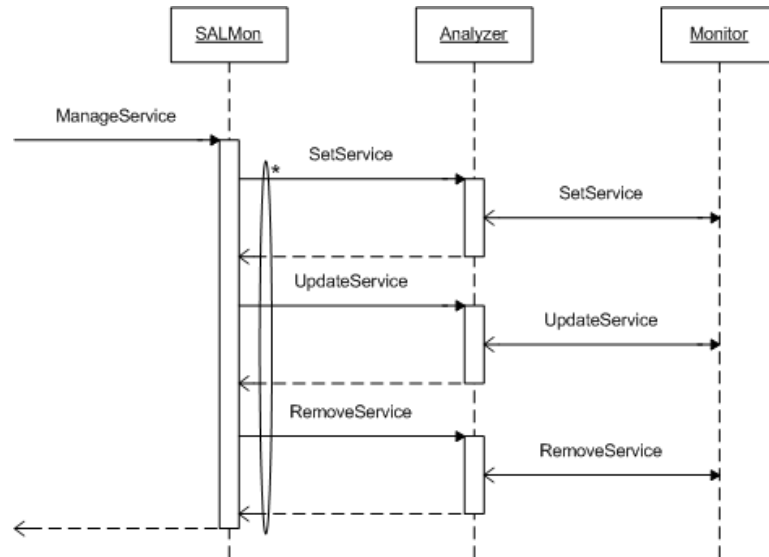


Figure 33: ManageSOASystem sequence diagrams

This use case represents the full life of a SOA system in SALMon system. The definition of a SOA system consists on creating a virtual space where the user can set a group of services and the SLA restrictions for them, after that the user can change the information in the same use case.

ManageService (Figure 34) shows the basic interaction in the system, where the Analyzer is always controlling the rest of SALMon services.

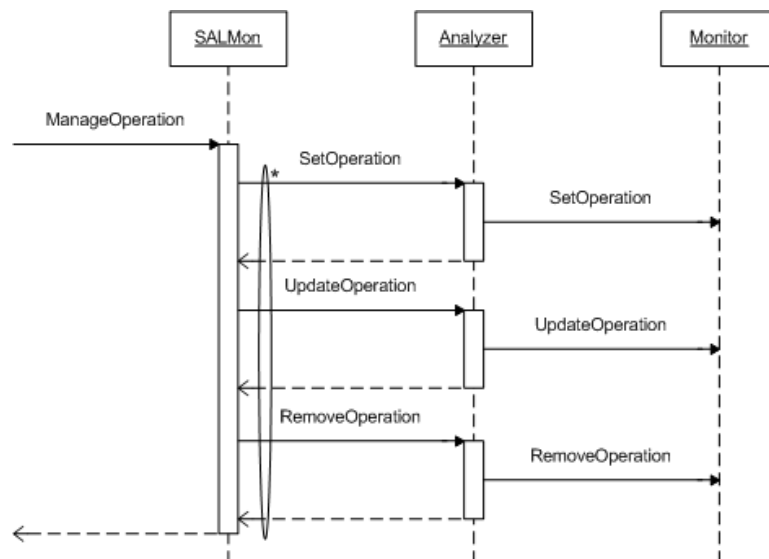
**1<sup>st</sup> and Current version (no changes)**



**Figure 34: ManageService sequence diagram**

ManageOperation (Figure 35) shows the basic interaction in the system to manage the operations of the Web Services. Although we manage the operations through these methods, we consider in a future work the ability to import WSDL files.

**Current version (it was not defined in 1<sup>st</sup> version)**



**Figure 35: ManageOperation sequence diagram**

The SetSLA use case (Figure 36) has two execution ways, using SLA specification or using the individual methods: set, update and remove condition (one call for each SLA property and condition). The second way is more useful when we want to change something in the current SLA.

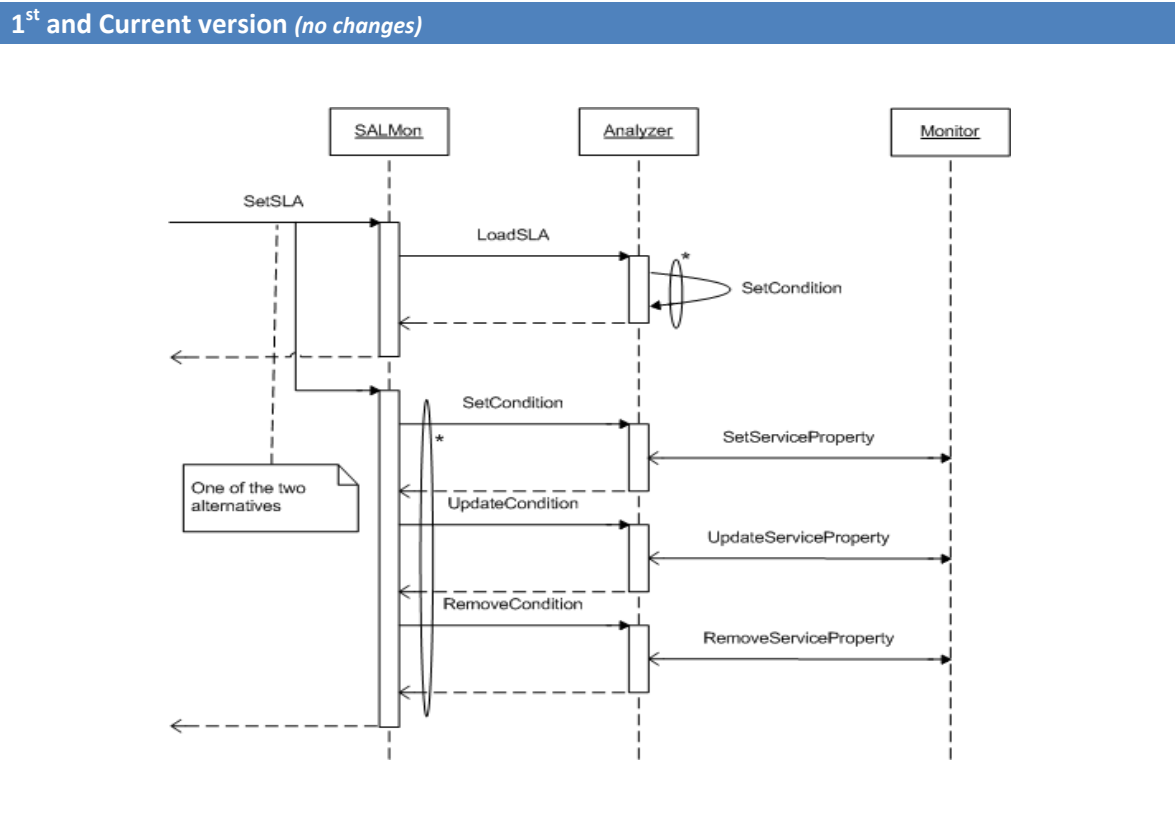


Figure 36: SetSLA sequence diagram

To perform the tests, in the first version, the Monitor called iteratively the different Measure Instruments every X periods of time. In this new version, however, the measure instruments are autonomous components able to perform the measures independently. In this sense, when we need to monitor a new metric, we create a new Measure Instrument which is able to perform the tests (Figure 37).

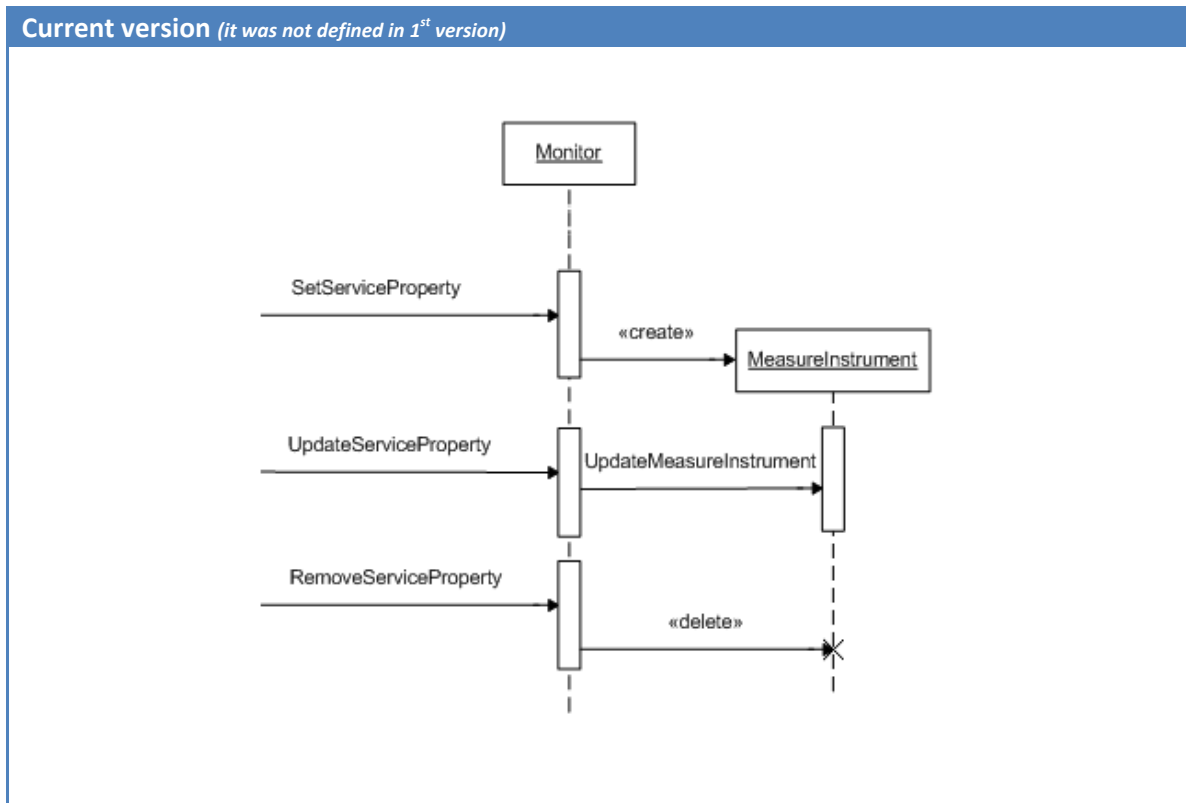


Figure 37: Measure Instruments sequence diagram.

To obtain the different monitored results (Figure 38), we have the initial operations ViewCurrentStatus and ViewReport. Additionally, in the new version, we have added the operation viewQoS.

- **ViewQoS**: to obtain the QoS of the Web Services (method implemented in the monitor)
- **ViewCurrentStatus**: to obtain a log of the conditions (fulfilled or unfulfilled) of the web services (method implemented in the Analyzer)
- **ViewReport**: to obtain a log of the actions performed in case of the unfulfillment of the conditions. ( method implemented in the DecisionMaker)

Currently, ViewQoS is the only one already implemented.

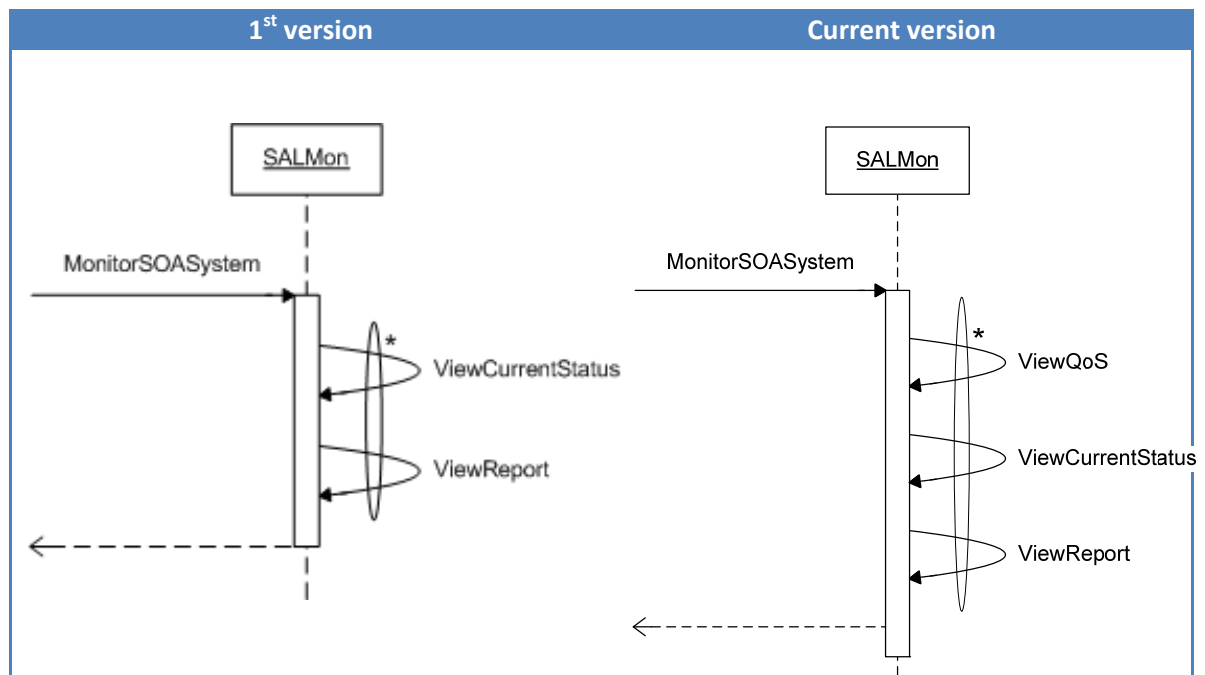


Figure 38: MonitorSOASystem sequence diagram

The ViewQoS sequence diagram (Figure 39) has the operation GetServicesDataFromDomain. This method retrieves the QoS of the web services of a certain domain. This is because of the Web Service Discovery system architecture requirements. Nevertheless, ViewQoS could be extended with other methods to provide the list of QoS using other filters.

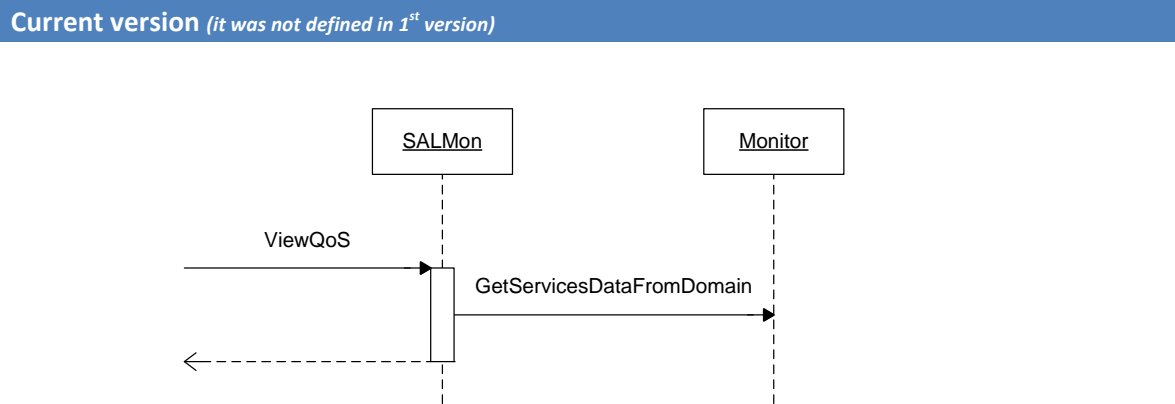


Figure 39: ViewQoS sequence diagram

The ViewCurrentStatus sequence diagram (Figure 40) shows how we will get information of the monitored services. We can get all the monitoring information or just the information of one SOA system or one service.

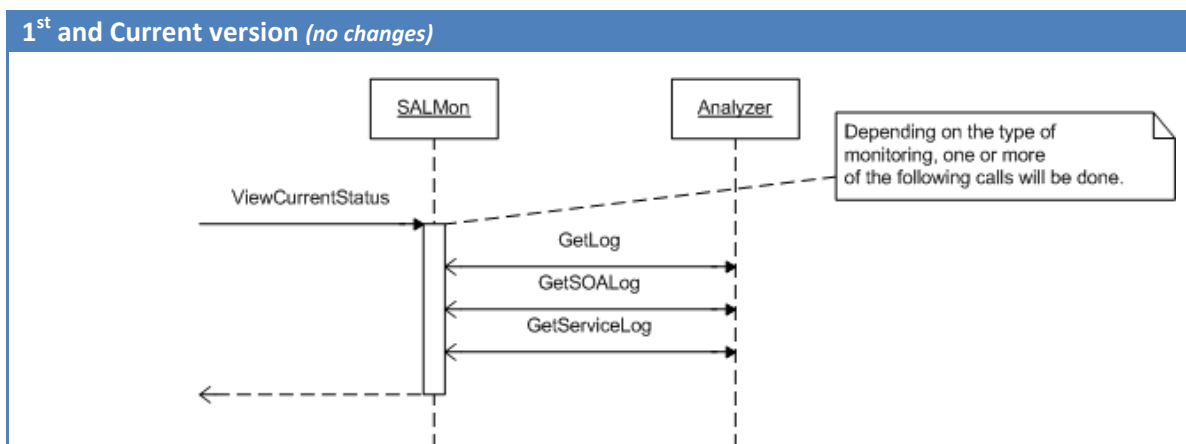


Figure 40: ViewCurrentStatus sequence diagram

The report information (Figure 41) depends on the Decision Maker, so it will only be available when the user had linked it with the Analyzer.

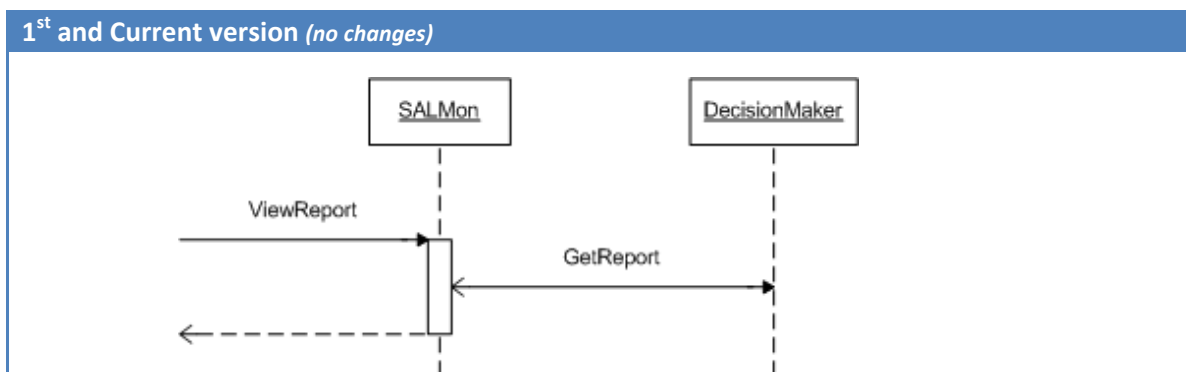


Figure 41: ViewReport sequence diagram



The CheckSLA sequence diagram (Figure 42) represents the Analyzer interaction with the system. The Analyzer will query to the data base and compare the results with the specified SLA for each Web Service. If any problem is detected and the SALMon system is configured to use a Decision Maker a notification will be sent. When a Decision Maker receives the notification it will select and apply the best treatment to resolve the issue.

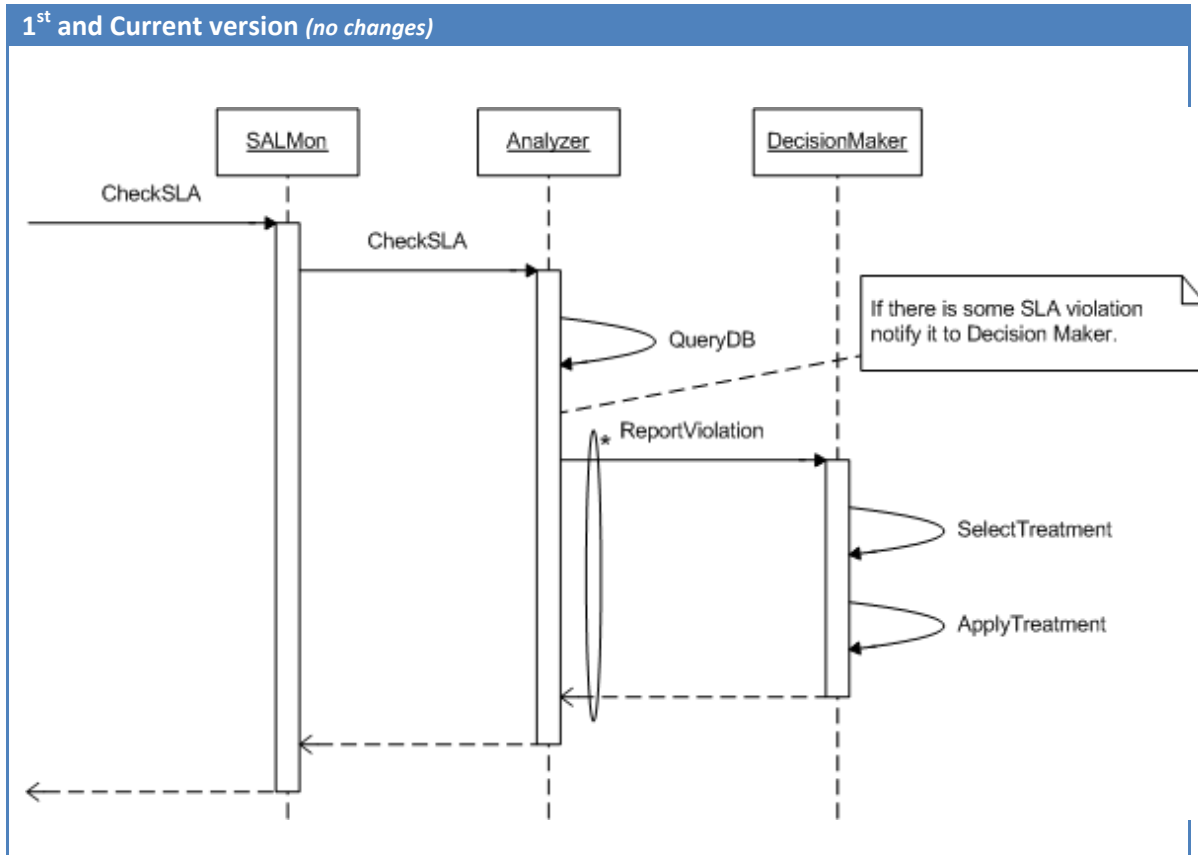


Figure 42: CheckSLA sequence diagram

# 25 Services Interface specification

## 25.1 Initial Specification

The initial Data model designed in the first version was the following:

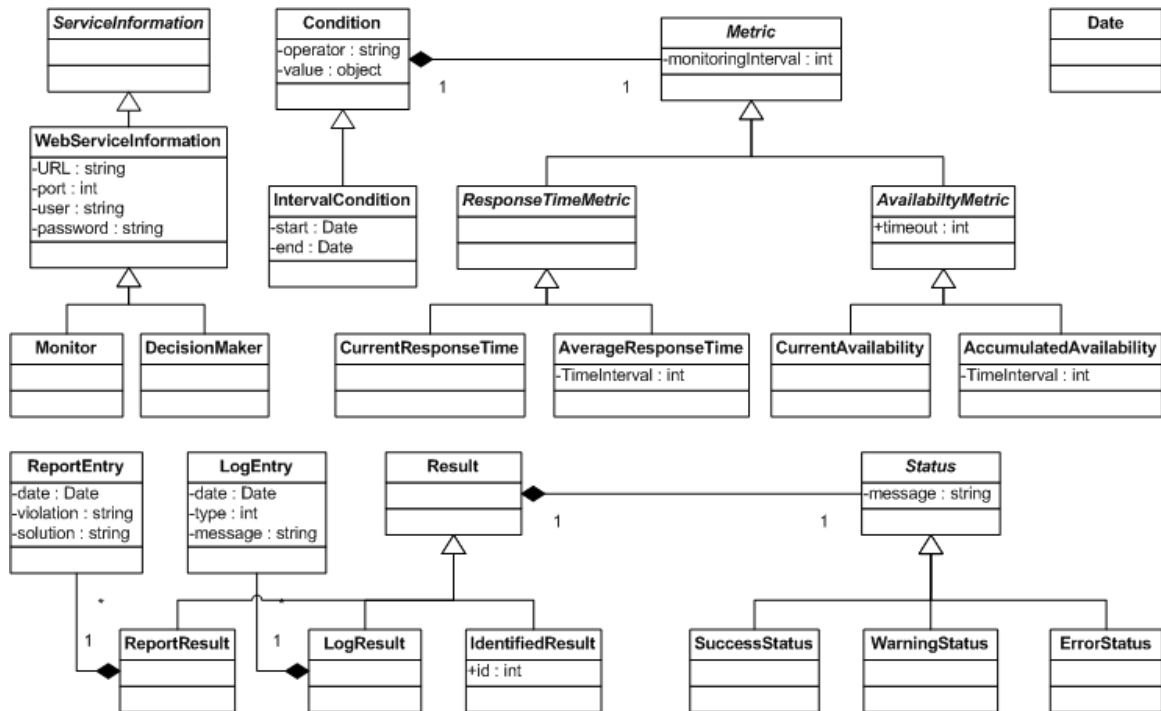


Figure 43: Initial Data Model Specification

As we can see, **WebServiceInformation** is a subclass of **ServiceInformation** providing hence, extensibility to monitor other kinds of services in future releases. Since **Monitor** and **DecisionMaker** are services used in Analyzer, they can be handled as subclasses of them.

Regarding to **Metrics**, here are represented only **Response Time** and **Availability** Metrics, for those derived ones, a new attribute named **TimeInterval** is used. In **Metric** we have the attribute **monitoringInterval** which indicates the period of performing the test. A **Metric** belongs to a **Condition** that establishes the **operator** and the **value** that must be fulfilled. A **Condition** can be for an interval of time, which is the subclass, **IntervalCondition**

Regarding to **Results**, they can be for Reports, Logs, or general results (Identified). Each result has an associated status which can be Success, Warning or Error.

## 25.2 Current Specification

The initial specification didn't cover the information needed to monitor operation-related metrics, which are Execution Time and Accuracy metrics. In order to provide this functionality enhancement we added the **OperationInformation** class as components of **WebServiceInformation**.

The attribute *monitoringInterval* indicates the period of time for each invocation. We believe that this information doesn't belong to the metric (since derived metrics shouldn't have this parameter) and it should belong to the MeasureInstrument. As a consequence, we add the **MeasureInformation** classes, which are classes that provide the necessary information to measure instruments to perform the tests.

**MeasureInformation** classes might be for Web Services or for Operations: **WebServiceMeasureInformation** or **OperationMeasureInformation**.

A **WebServiceInformation** have associated a **WebServiceMonitor**, which is the class that handles the **MeasureInformations**. Each **WebServiceMonitor** can have one **WebServiceMeasureInformation** and several **OperationMeasureInformations**.

From each MeasureInstrument, we obtain the Measurement Values that might be **Measurements** for Accuracy, Execution Time, Response Time or Availability. These **Measurements** have a set of **Values** correspondingly.

In this new scenario, we are able to describe the whole set of metrics. All of them are computed from the **Measurements** that the Measure Instruments perform. And consequently, we have added extensibility to add new kind of derived metrics easily.

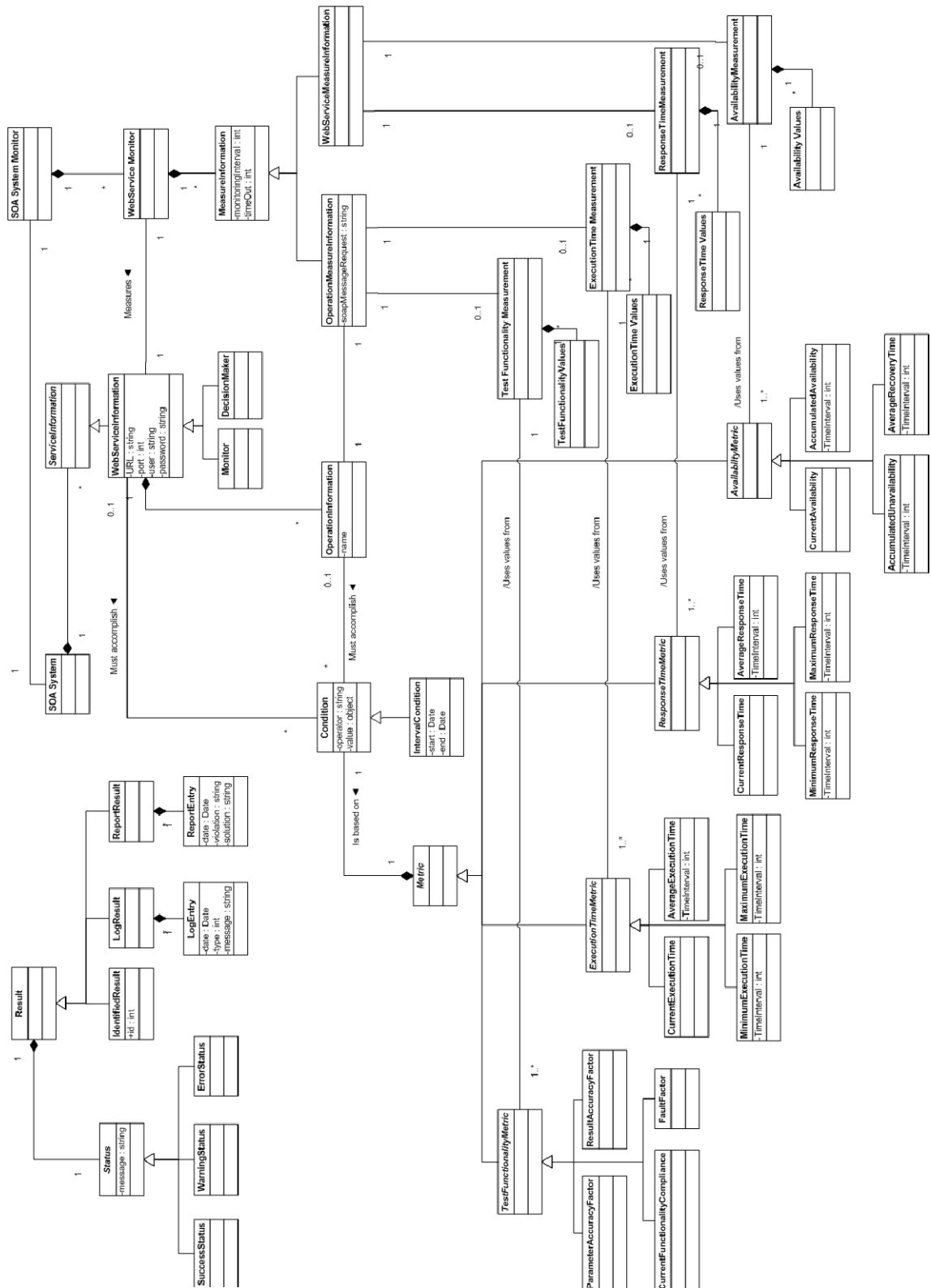


Figure 44: Current Data Model Specification

## 25.3 Analyzer service interface

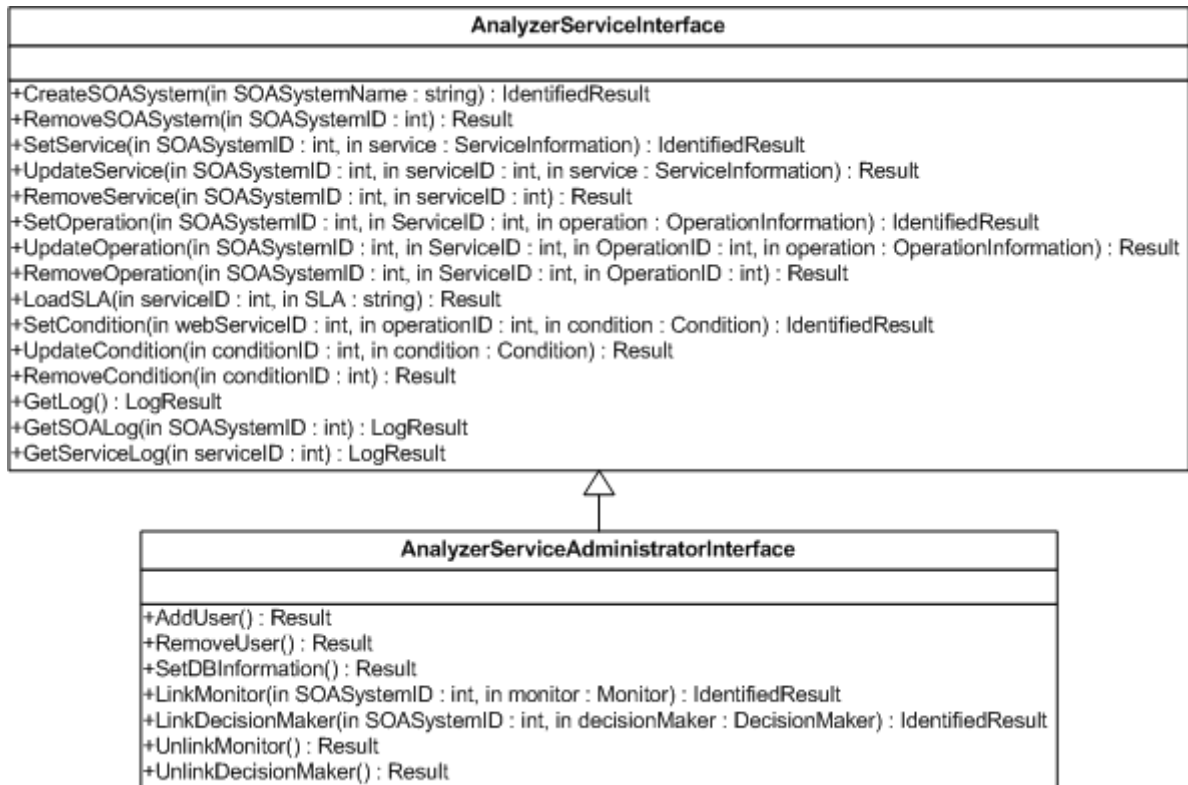


Figure 45: Analyzer service interface

## 25.4 Decision Maker interface

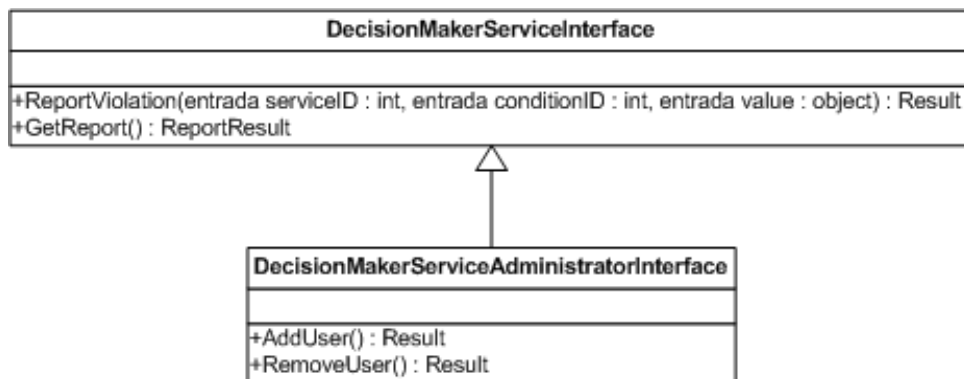


Figure 15: Decision Maker service interface

## 25.5 Monitor interface

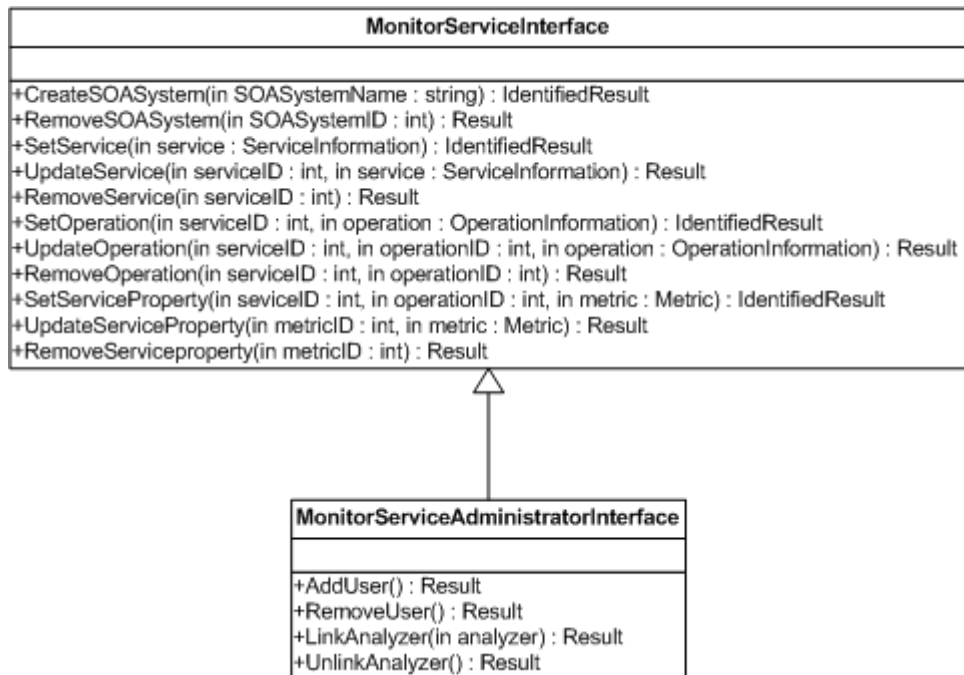


Figure 46: Monitor service interface

## 26 Some implementation details

---

SALMon has been implemented in Java v5.0 under the eclipse platform. The web service engine in which SALMon has been developed is Axis2. The decision of using Axis2 is because it is one of the most used web service engine for the java programming language.

Since it follows the SOA paradigm, each method of the service is responsible to do a small task, which basically ends to the monitor at managing the database information or the measure instruments.

### 26.1 Storage

Regarding to the storage of data, the database chosen has been mySQL. Nevertheless, it is planned to migrate it to a streaming database because of the nature of the data it manages. The architecture chosen for mapping the information into a database has been the Data Mapper pattern. In this pattern, the classes in OO-paradigm have a correspondent table in the database, and each object can be stored or accessed from the database through the use of a mapper that have operations to save, load and delete them.

We have discarded the use of technologies such as Hibernate or Ibatis because of efficiency problems. Additionally, ActiveRecord has also been discarded because it lacks of a stable support for the Java language.

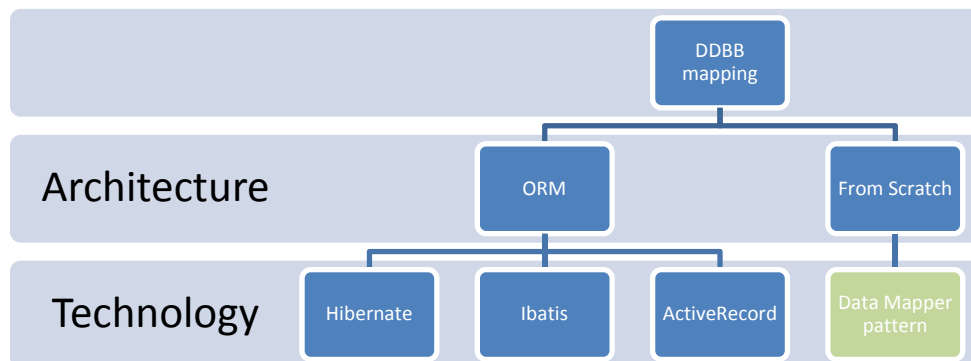


Figure 47: DDBB mapping technologies

## 26.2 Multithreading

Regarding to measure instruments, they have been implement as threads. The objective of this approach is to minimize the effects that a web service behavior could affect do to another one.

We propose the use of multithread testing in order to solve the following problems:

- Each service or operation can be monitored in different time intervals.
- Each service or operation being tested should affect as minimum as possible other service tests.

Therefore, each service or operation being tested should be implemented as an independent thread that performs the testing on the given time interval. In this situation, if a service response crashes or affects in a not desired way the monitoring (i.e. the size of the response is extremely heavy to process it in an effective way), it will only affect at the service itself since the only thread affected will be the one monitoring it.

## 27 Future Work of SALMon

---

The first release of SALMon is still a prototype and an on-going project. Therefore, there are still some functionalities to be implemented for the next releases. Concretely, the following points have been identified:

- Use of standards WSDL and WS-Agreement to set the operations and conditions respectively.
- Implement measure instruments to monitor other kinds of services apart from Web Services.
- Implement SALMon with the passive monitoring approach.
- Provide security capabilities into SALMon.



## 28 References

---

- [1] D. Ameller and X. Franch, "Service level agreement monitor (SALMon)," in *Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, 2008, pp. 224-227.
- [2] F. Casati, M. C. Shan and D. Georgakopoulos, "E-services-guest editorial," *The VLDB Journal*, vol. 10, pp. 1, 2001.
- [3] M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, "Service-oriented computing: State of the art and research challenges," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 40, pp. 38, 2007.
- [4] B. Kitchenham, "Procedures for performing systematic reviews," *Keele University TR/SE-0401/NICTA Technical Report 0400011T*, vol. 1, 2004.
- [5] A. Sillitti, D. Desideri, F. Garijo, F. Pérez-Sorrosal and et al., "NEXOF - reference architecture - state of the art report," 2008.
- [6] I. ISO, "ISO 9126/ISO, IEC (Hrsg.): International Standard ISO/IEC 9126: Information Technology-Software Product Evaluation," 1991.
- [7] Institute of Electrical and Electronics Engineers, "IEEE 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology," 1990.
- [8] I. ISO, "ISO 8402:1994 - Quality management and quality assurance - Vocabulary," 1994.
- [9] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*. Springer, 2003,
- [10] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002,
- [11] X. Wang, T. Vitvar, M. Kerrigan and I. Toma, "A QoS-aware selection model for semantic web services," *Lecture Notes in Computer Science*, vol. 4294, pp. 390, 2006.
- [12] L. Zeng, H. Lei and H. Chang, "Monitoring the QoS for Web Services," *Lecture Notes in Computer Science*, vol. 4749, pp. 132, 2007.
- [13] W3C Member Submission, "OWL-S: Semantic Markup for Web Services: <http://www.w3.org/Submission/OWL-S/>," 2004.
- [14] W3C, "Acknowledged Member Submissions to W3C: <http://www.w3.org/Submission>,"
- [15] C. Zhou, L. T. Chia and B. S. Lee, "QoS measurement issues with DAML-QoS ontology," in *Proc. IEEE International Conference on e-Business Engineering (ICEBE)*, 2005,
- [16] C. Zhou, L. T. Chia and B. S. Lee, "DAML-QoS ontology for web services," in *IEEE International Conference on Web Services, 2004. Proceedings*, 2004, pp. 472-479.

- [17] E. Giallonardo and E. Zimeo, "More semantics in qos matching," in *IEEE International Conference on Service-Oriented Computing and Applications, 2007. SOCA'07, 2007*, pp. 163-171.
- [18] K. Kritikos and D. Plexousakis, "Semantic qos metric matching," in *Web Services, 2006. ECOWS'06. 4th European Conference on, 2006*, pp. 265-274.
- [19] Y. Zhang, Y. Qu, H. Huang, D. Yang and H. Zhang, "An ontology and peer-to-peer based data and service unified discovery system," *Expert Syst. Appl.*, vol. 36, pp. 5436-5444, 2009.
- [20] H. Lausen, A. Polleres and D. Roman, "Web service modeling ontology (WSMO)," *W3C Member Submission*, vol. 3, 2005.
- [21] I. J. Jureta, C. Herssens and S. Faulkner, "A comprehensive quality model for service-oriented systems," *Software Quality Journal*. Accepted for Publication (Available Online at: <http://www.Jureta.net/papers/QVDPdraft.Pdf>).
- [22] A. D'Ambrogio, "A model-driven wsdl extension for describing the qos of web services," in *Proceedings of the International Conference on Web Services (ICWS'06), 2006*,
- [23] K. Zhu, Z. Duan and J. Wang, "Quality of service in web services discovery," in *IEEE Symposium on Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008, 2008*, pp. 1-5.
- [24] Y. H. Kang, "Extended Model Design for Quality Factor Based Web Service Management," *Future Generation Communication and Networking (FgcN 2007)*, vol. 2, 2007.
- [25] J. Kopecky, T. Vitvar, C. Bournez and J. Farrell, "SawSDL: Semantic annotations for WSDL and XML schema," *IEEE Internet Comput.*, vol. 11, pp. 60-67, 2007.
- [26] W3C, "Semantic Annotations for WSDL and XML Schema: <http://www.w3.org/TR/sawSDL>," 2007.
- [27] OASIS, "Web Services Quality Description Language v1.0," 2008,
- [28] OASIS, "OASIS Web Services Quality Model TC Public Documents: [http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=wsqm](http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsqm)," 2008.
- [29] C. C. Lo, D. Y. Cheng, P. C. Lin and K. M. Chao, "A study on representation of QoS in UDDI for web services composition," in *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on, 2008*, pp. 423-428.
- [30] A. Blum and F. Carter, "Representing Web Services Management Information in UDDI," 2004.
- [31] Z. Xu, P. Martin, W. Powley and F. Zulkernine, "Reputation-enhanced qos-based web services discovery," in *IEEE International Conference on Web Services, 2007. ICWS 2007, 2007*, pp. 249-256.
- [32] OASIS, "[http://www.oasis-open.org/committees/download.php/5649/TC\\_FTF\\_Minutes-V1.7-20040210-12.htm#\\_Toc65400707](http://www.oasis-open.org/committees/download.php/5649/TC_FTF_Minutes-V1.7-20040210-12.htm#_Toc65400707)," 2004.

- [33] A. Naumenko, S. Nikitin, V. Terziyan and J. Veijalainen, "Using UDDI for publishing metadata of the semantic Web," *INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING-PUBLICATIONS-IFIP*, vol. 188, pp. 141, 2005.
- [34] A. ShaikhAli, O. Rana, R. Al-Ali and D. Walker, "Uddie: An extended registry for web services," in *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, 2003, pp. 85-89.
- [35] M. A. Serhani, R. Dssouli, A. Hafid and H. Sahraoui, "A QoS broker based architecture for efficient web services selection," in *2005 IEEE International Conference on Web Services, 2005. ICWS 2005. Proceedings*, 2005, pp. 113-120.
- [36] E. Badidi, L. Esmahi, M. A. Serhani and M. Elkoutbi, "WS-QoS: A Broker-based Architecture for Web Services QoS Management," *Innovations in Information Technology*, 2006, pp. 1-5, 2006.
- [37] R. Tabein and A. Nourollah, "Dynamic broker-based service selection with QoS-driven recurrent counter classes," in *Service Systems and Service Management, 2008 International Conference on*, 2008, pp. 1-6.
- [38] T. Yu and K. J. Lin, "Service selection algorithms for Web services with end-to-end QoS constraints," *Information Systems and E-Business Management*, vol. 3, pp. 103-126, 2005.
- [39] Y. Liu, A. H. Ngu and L. Z. Zeng, "QoS computation and policing in dynamic web service selection," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, 2004, pp. 66-73.
- [40] E. Al-Masri and Q. Mahmoud, "QoS-based discovery and ranking of web services," in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 2007, pp. 529-534.
- [41] M. A. Serhani, E. Badidi, A. Benharref and M. Salem, "A cooperative approach for QoS-aware web services' selection," in *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*, 2008, pp. 1084-1088.
- [42] V. Cardellini, E. Casalicchio, V. Grassi and R. Mirandola, "A framework for optimal service selection in broker-based architectures with multiple QoS classes," *Services Computing Workshops, SCW*, pp. 105-112, 2006.
- [43] S. Ran, "A model for web services discovery with QoS," *ACM SIGecom Exchanges*, vol. 4, pp. 1-10, 2003.
- [44] J. Matai and D. S. Han, "Learning-Based Trust Model for Optimization of Selecting Web Services," *Lecture Notes in Computer Science*, vol. 4505, pp. 642, 2007.
- [45] P. Brittenham, *Understanding the WS-I Test Tools*, 2003.
- [46] D. Chappell, *Enterprise Service Bus*. O'Reilly Media, Inc., 2004,

- [47] R. B. Halima, K. Guennoun, K. Drira and M. Jmaiel, "Non-intrusive QoS monitoring and analysis for self-healing web services," in *Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the*, 2008, pp. 549-554.
- [48] A. Benharref, R. Dssouli, M. A. Serhani and R. Glitho, "Efficient traces' collection mechanisms for passive testing of Web Services," *Information and Software Technology*, vol. 51, pp. 362-374, 2009.
- [49] R. Jurca, B. Faltings and W. Binder, "Reliable QoS monitoring based on client feedback," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 1003-1012.
- [50] J. Hielscher, R. Kazhamiakin, A. Metzger and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing," in *1st International Conference of the Future of the Internet of Services (ServiceWave 2008), Madrid, Spain*, 2008,
- [51] C. Bartolini, A. Bertolino, E. Marchetti and A. Polini, "WS-TAXI: A WSDL-based testing tool for web services," in *Proceedings of the 2009 International Conference on Software Testing Verification and Validation-Volume 00*, 2009, pp. 326-335.
- [52] A. Bertolino, P. Inverardi, P. Pelliccione and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in *ESEC/FSE 2009*, 2009,
- [53] A. Bertolino, G. De Angelis and A. Polini, "A QoS test-bed generator for web services," *Lecture Notes in Computer Science*, vol. 4607, pp. 17, 2007.
- [54] H. G. Song and K. Lee, "sPAC (Web Services Performance Analysis Center): Performance analysis and estimation tool of web services," *Lecture Notes in Computer Science*, vol. 3649, pp. 109, 2005.
- [55] D. A. D'Mello and V. S. Ananthanarayana, "Quality and business offer driven selection of web services for compositions," in *Information Systems, Technology and Management: Third International Conference, ICISTM 2009, Ghaziabad, India, March 12-13, 2009. Proceedings*, 2009, pp. 76.
- [56] W. L. Lin, C. C. Lo, K. M. Chao and M. Younas, "Fuzzy consensus on QoS in web services discovery,"
- [57] P. Wang, "QoS-aware web services selection with intuitionistic fuzzy set under consumer's vague perception," *Expert Syst. Appl.*, vol. 36, pp. 4460-4466, 2009.
- [58] P. Wang, K. M. Chao, C. C. Lo, C. L. Huang and Y. Li, "A fuzzy model for selection of QoS-aware web services," in *IEEE International Conference on e-Business Engineering, 2006. ICEBE'06*, 2006, pp. 585-593.
- [59] K. T. Atanassov, P. P. Angelov, S. C. Bennett, K. Atanassov, G. Gargov, S. C. Bennett, J. Bezdek, J. Bezdek, E. K. C. Blake and C. J. Merz, "Intuitionistic fuzzy sets," *Intelligent Systems for Finance and Business*, vol. 25, pp. 111-133,
- [60] L. Taher, H. El Khatib and R. Basha, "A framework and QoS matchmaking algorithm for dynamic web services selection," in *Second International Conference on Innovations in Information Technology (IIT'05)), Dubai, UAE*, 2005,

- [61] K. Yoon and C. L. Hwang, *Multiple Attribute Decision Making: An Introduction*. Sage Pubns, 1995,
- [62] R. Tabein, M. N. Moghadasi and A. Khoshkbarforousha, "Broker-based web service selection using learning automata," in *Service Systems and Service Management, 2008 International Conference on*, 2008, pp. 1-6.
- [63] G. Wu, J. Wei, X. Qiao and L. Li, "A bayesian network based qos assessment model for web services," in *Proceedings of the IEEE International Conference on Services Computing (SCC'07)*, 2007,
- [64] IBM, "Introducing the Web Services Flow Language, <http://www.ibm.com/developerworks/library/ws-ref4/>,"
- [65] Microsoft, "XLANG/s Language, <http://msdn.microsoft.com/en-us/library/aa577463.aspx>,"
- [66] OASIS, "Web Services Business Process Execution Language Version 2.0, <http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm>,"
- [67] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1981,
- [68] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," in *Proceedings of the 14th Australasian Database Conference-Volume 17*, 2003, pp. 191-200.
- [69] D. Menasce, "QoS issues in Web services," *IEEE Internet Comput.*, vol. 6, pp. 72-75, 2002.
- [70] B. Kiepuszewski, A. H. M. Hofstede and C. J. Bussler, "On structured workflow modelling," *Lecture Notes in Computer Science*, vol. 1789, pp. 431-445, 2000.
- [71] J. Cardoso, A. Sheth, J. Miller, J. Arnold and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, pp. 281-308, 2004.
- [72] T. Yu and K. J. Lin, "A broker-based framework for qos-aware web service composition," in *E-Technology, e-Commerce and e-Service, 2005. EEE'05. Proceedings. the 2005 IEEE International Conference on*, pp. 22-29.
- [73] B. Li, X. Y. Tang and J. Lv, "The research and implematation of services discovery agent in web services composition framework," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, 2005,
- [74] S. Y. Hwang, H. Wang, J. Tang and J. Srivastava, "A probabilistic approach to modeling and estimating the QoS of web-services-based workflows," *Inf. Sci.*, vol. 177, pp. 5484-5503, 2007.
- [75] D. Mukherjee, P. Jalote and M. G. Nanda, "Determining QoS of WS-BPEL compositions," in *Proceedings of the 6th International Conference on Service-Oriented Computing*, 2008, pp. 378-393.

- [76] X. Gu, K. Nahrstedt, R. Chang and C. Ward, "QoS-assured service composition in managed service overlay networks," in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, 2003, pp. 194-201.
- [77] P. C. Xiong, Y. S. Fan and M. C. Zhou, "Web Service Configuration under Multiple Quality-of-Service Attribute," *IEEE Trans.on Automation Science and Engineering*, 2007.
- [78] R. Berbner, M. Spahn, N. Repp, O. Heckmann and R. Steinmetz, "Heuristics for QoS-aware web service composition," in *Web Services, 2006. ICWS'06. International Conference on*, 2006, pp. 72-82.
- [79] W. Zhang, Y. Yang, S. Tang and L. Fang, "Qos-driven service selection optimization model and algorithms for composite web services," in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, 2007,
- [80] Q. Liang, H. C. Lau and X. Wu, "Robust application-level QoS management in service-oriented systems," in *IEEE International Conference on e-Business Engineering, 2008. ICEBE'08*, 2008, pp. 239-246.
- [81] G. Canfora, M. Di Penta, R. Esposito and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, 2005, pp. 1069-1075.