DOCTEUR EN SCIENCES

Sustaining the Quality of Web services

Lim, Erbin

Award date:
2014

Awarding institution:
Universite de Namur

Erbin Lim (PhD student), Philippe Thiran (Supervisor)

Sustaining the Quality of Web Services

A thesis submitted to the
Faculty of Computer Science of the
University of Namur in fulfillment
of the requirements for the degree of
PhD., Computer Science
PReCISE Research Centre
Faculty of Computer Science
University of Namur

January 3, 2014

Contents

1 Introduction . . . 1
1.1 Introduction . . . 1
1.1.1 Web Services Differences . . . 3
1.1.2 Non-Functional Properties . . . 3
1.1.3 Sustaining Quality of Service . . . 5
1.1.4 Grouping of Similar Web Services . . . 5
1.2 Main Questions . . . 7
1.2.1 User Request Management Questions . . . 7
1.2.2 Member Management Questions . . . 8
1.3 Methodology . . . 9
1.4 Thesis Organisation . . . 10
1.4.1 Related Work . . . 10
1.4.2 Preliminaries . . . 11
1.4.3 Solution . . . 11
1.4.4 User Request Management . . . 12
1.4.5 Member Management . . . 12
1.4.6 Discussion . . . 13
1.4.7 Conclusion and Future Work . . . 14
1.5 Contribution . . . 14
1.5.1 User Request Management Contribution . . . 14

1.5.2 Member Management Contribution 14

2 Related Work 17

2.1 User Request Management - Selection Mechanism 17

2.1.1 Selection - Responsibility 19

2.1.2 Selection - Full or Partial Observability 20

2.1.3 Selection - Central vs Distributed 22

2.1.4 Selection - Immediate vs Batch 25

2.1.5 Selection - Priority 26

2.1.6 Selection - Auction Theory 28

2.1.7 Selection - Utility Function 29

2.2 User Request Management - Substitution Mechanism 31

2.2.1 Substitution - Monitoring 31

2.2.2 Substitution - Replication 34

2.3 Member Management 36

2.3.1 Organisation 37

2.3.2 Communication 39

2.3.3 Client Prediction 40

2.3.4 Admission and Expulsion 41

2.3.5 Group QoS 44

3 Preliminaries 45

3.1 Service Oriented Architecture 45

3.2 Quality of Service 48

3.2.1 Quality of Service Properties 48

3.2.2 Selected QoS Properties 51

3.2.3 Request 52

3.3 Service Level Agreement 52

3.3.1 Background 53

3.3.2 Service Level Agreement Components 55

3.4 Auctions 56

3.4.1 4 Basic Auction Systems . . . 57
3.4.2 Item values . . . 58
3.4.3 Bidder Types . . . 58
3.4.4 Auction Strategies . . . 59
3.4.5 Auction Properties . . . 60
3.4.6 Auction Comparison . . . 61
3.4.7 Fixed Price Auctions . . . 61

4 Solution . . . 65
4.1 Framework - Community of Web services . . . 65
4.1.1 Master of Web services . . . 68
4.1.2 Slave Web services . . . 72
4.1.3 Slave Web Service Quality of Service properties . . . 73
4.2 Contract Net Protocol . . . 74
4.3 Auction Theory in a Community . . . 76
4.4 Fixed Price Auctions in a Community . . . 79
4.5 Supplier-side SLA Framework . . . 81
4.5.1 Step 1: Negotiation . . . 83
4.5.2 Step 2: Signing and Setup . . . 84
4.5.3 Step 3: Monitoring . . . 84
4.5.4 Step 4: Penalties . . . 85
4.5.5 Step 5: Termination . . . 85
4.5.6 Summary . . . 85

5 User Request Management . . . 87
5.1 3-way Satisfaction . . . 87
5.2 User Request Selection Process . . . 88
5.2.1 Client Selection Score . . . 89
5.2.2 SWS Selection Score . . . 90
5.2.3 MWS Selection Score . . . 90
5.2.4 Selection Score . . . 91

5.2.5 SLA Implications . . . 91
5.3 User Request Substitution Process . . . 92
5.3.1 QoS Monitoring . . . 93
5.3.2 SLA Implications for Substitution . . . 94
5.4 Experiments . . . 95
5.4.1 Experiments Introduction and Setup . . . 96
5.4.2 Fixed Price Auction Experiments . . . 97
5.4.3 3-Way Satisfaction Approach . . . 106

6 Member Management . . . 121
6.1 Client Model . . . 123
6.1.1 Number of Requests . . . 124
6.1.2 Predicted Requested Availability . . . 124
6.1.3 Predicted Requested Reliability . . . 125
6.1.4 Predicted Requested Execution Duration . . . 125
6.1.5 Predicted Requested Price . . . 126
6.1.6 Summary of Predicted QoS Values . . . 126
6.2 QoS of the Community . . . 127
6.2.1 QoC - Availability . . . 127
6.2.2 QoC - Reliability . . . 127
6.2.3 QoC - Capacity . . . 128
6.2.4 QoC - Execution Duration . . . 128
6.2.5 QoC - Price . . . 128
6.2.6 QoC - Variance . . . 128
6.2.7 QoC - Summary . . . 129
6.3 Admission . . . 129
6.3.1 Projected Satisfaction - Preliminary . . . 130
6.3.2 Projected Client Satisfaction . . . 131
6.3.3 Projected Web Services Satisfaction . . . 137
6.3.4 Projected Community Satisfaction . . . 137

6.3.5 WS Admission - Total Score 139
6.4 Web service perspective 139
6.5 Expulsion of SWSs 140
6.5.1 Projected Client Satisfaction - Expulsion 141
6.5.2 Projected SWS Satisfaction - Expulsion 144
6.5.3 Projected Community Satisfaction - Expulsion 145
6.5.4 WS Expulsion - Total Score 145
6.6 Experiments 146
6.6.1 Normal Client - Normal Community 148
6.6.2 Sudden Change in Client QoS 152
6.6.3 Sudden Change in Number of Requests 155
6.6.4 Experiments Conclusion 160

7 Discussion 163
7.1 Discussion - User Request Management 163
7.1.1 Selection Responsibility 164
7.1.2 QoS Based Analysis 165
7.1.3 Three-way Satisfaction 165
7.1.4 Submission of price only 166
7.1.5 Fully automated 166
7.1.6 Single point of failure 166
7.1.7 Substitution Mechanism 167
7.1.8 Composition 167
7.1.9 Modelled with Specific QoS properties 167
7.1.10 Summary 168
7.2 Discussion - Member Management 168
7.2.1 Three way satisfaction 169
7.2.2 Auto-updating of QoS Level 169
7.2.3 QoS of a group of Web services 170
7.2.4 Broker and Provider SLA negotiation per request 170

8 Conclusion 171

9 Future plans 175

10 Acknowledgements 177

References 179

1 Introduction

In this chapter, we present the introduction for the thesis in Section 1.1. Next in Section 1.2 we present the main tasks of the thesis. Next in Section 1.3 we describe the methodology of our thesis. In Section 1.4 we describe how our thesis is organised. Finally in Section 1.5 we list the contribution of this thesis to the research community.

1.1 Introduction

Web services have emerged as a technology that is increasing in popularity. Its ability to effectively automate processes is the main reason for its growing usage. We refer to a Web service (WS) as *a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based applications*" (3WC).

The increasing dependence on Web services means that services are affected if the Web services were to fail to deliver its promised level of service. In extreme circumstances, Web services may fail completely, causing the service to be unavailable to users. On the 21st of April 2011, Amazon's Web services failed [120] and affected many other Internet sites. Its web services that provide utility-style computing in which customers pay only for the computing power and storage that they need were unable to be accessed due to the failure and caused widespread inconvenience to many users.

Since Web services provide a service, they are also known as the service provider. In this thesis, we use the two terms interchangeably. The use of a Web service starts with the *service client*, which directs its search to the *service registry* which contains a listing of Web services. This is similar to looking up the phonebook for the phone number of a person. Web services willing to offer its services would have previously published its information on the *service registry*. Once the service client has decided which Web service to use, the client binds with the Web service (service provider). The binding process includes having the service client identify itself and send the request to the service provider. Once the service provider has handled the request, a reply is sent back to the service client. The relationship between service client, service provider and service registry is summarized in Figure 1.1.

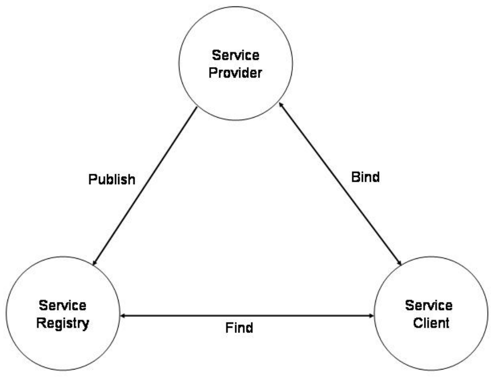


Fig. 1.1. Relationship between the Service Client, Service Provider and Service Registry.

No two Web services can be completely identital, therefore the service registry categorizes Web services into what kind of Web services it is as well as how well the Web service can handle the request. In this thesis, we are also focused with Web services that do not require human interaction, specifically machine to machine Web services. In the next section we elaborate on the differences between Web services.

1.1.1 Web Services Differences

When a client searches for a Web service, the client may be presented with several different Web services. For the client to be able to make a decision on which Web service will be selected to handle the request, there is a need to be able to distinguish the Web services from each other. The difference between Web services can be broadly divided into two categories:

- *Functional Properties* - Defines the particular results of the system [124]. For example, the functional property for a weather reporting Web service is to report the weather.
- *Non-Functional Properties* - Defines the characteristics of the system [124]. Examples of non-functional properties include cost and reliability.

The functional property of a Web service is straightforward, this is decided by the system designer and system administrator as to what kind of service the Web service will provide. We consider Web services with the same functional property to be *similar Web services.*

The non-functional properties of a Web service can be difficult to determine, since it varies from Web service to Web service. We discuss this further in the next section (Section 1.1.2).

1.1.2 Non-Functional Properties

There are many different ways to define how well a Web service performs its functional property. When the client receives the non-functional properties of a Web service, it may include non-functional properties which the client is either not concerned about or does not understand. For this reason, it is important for both the client and Web service to come to an agreement as to which non-functional properties they are concerned about and their respective definitions.

When the client receives the non-functional properties of a Web service, the client can determine whether the non-functional properties are acceptable according to the non-functional requirements of the request.

In this thesis, we focus on technical non-functional properties, a list of which can be found in [33]. Specific to the domain of Web services, non-functional properties define the level of service which the Web service can provide. These non-functional properties are thus known as Quality of Service (QoS) properties. Within this domain, we assume that the values of the QoS properties of Web services can be measured and quantified in a consistent manner. This allows easy comparison between the level of QoS that different Web services can provide.

The nature of the non-functional properties raises a few questions.

- What are the differences between the non-functional properties?
- How should the value of the non-functional properties of each Web service be measured?
- How should the value of the non-functional properties of each Web service be computed?
- How should the value of the non-functional properties of each Web service be compared?
- Who should be responsible for measuring the QoS of each Web service?
- Should the level of QoS of each Web service be tracked over time? This is known as QoS-monitoring.

1.1.3 Sustaining Quality of Service

At the end of the previous section, we briefly mentioned whether the level of QoS should be tracked over time. Web services are susceptible to changes in the level of QoS. This can be due to a number of reasons such as technical failure or overload. This brings inconvenience to the client who is expecting a certain level of QoS from the Web service.

This problem can be minimized if Web services can sustain the level of QoS provided to the client. We define *sustain* as *to make something continue* [95]. If the level of QoS can be sustained at a certain level, the client will receive the level of QoS that was promised during the initial negotiations.

There are different levels of severity at which a Web service cannot sustain the level of QoS. We assume in this thesis that providing a higher level of QoS that was agreed upon gives the same amount of satisfaction to the client as providing the original level of QoS. The problem comes in when a Web service provides a lower level of QoS to the client. In extreme circumstances, the Web service may fail completely, without even being able to warn the client of its failure.

Once the client is aware that the agreed upon level of QoS cannot be provided, the client may have to find another Web service with the same functionality in order to handle the request. This is inconvenient to the client who has to restart the search process for a new Web service and resend the request to the new Web service.

In order to simply what the client needs to do, one possible solution is to group similar Web services together in order to assist each other, which we describe in the next section (Section 1.1.4).

1.1.4 Grouping of Similar Web Services

Instead of having the client send its request to one Web service, one solution is to send its request to a group of similar Web services. Web services in this group are now able to substitute one another in case a Web service is unable to meet the QoS requirements of the request. A substitution occurs when one Web service takes over the handling of the request from another Web service. This substitution process is transparent to the client, this means that the client only has to send the request once to the group of Web services and wait for its reply. Multiple failures can occur within the group of Web services but this is not known to the client. In essence, the group of Web services act as a black-box, where the client sends a request and eventually gets a reply. The details of how the request was handled or the identity of the Web service handling the request is oblivious to the client.

By making this substitution process transparent to the client, the group of Web services has to ensure that the QoS of the Web service that handles the request can meet the QoS requirements of the request.

New challenges occur due to this group of similar Web services. We list them here:

- When a request is sent from the client to the group, which Web service should be selected to handle the request. This is known as the selection process.
- In the event that a Web service fails, which Web service should be selected to substitute the original Web service. This is known as the substitution process.
- How should the group of Web services be organised?
- How should the group of Web services be managed?
- Since there are now multiple Web services (with different QoS levels) within the group, how should the QoS properties of the group be defined?
- Who determines the QoS level of the Web services within the group? Should they be self-declared by each Web service or determined by a 3rd party? Considering that there might be a change in the QoS of each Web service over time, who is responsible for monitoring the QoS of the Web services?

- Which Web services should be included in this group? Should we allow Web services to enter and exit the group? What criteria should be used to decide when the entering or exiting of a Web service should occur?
- Is there an optimal size of the number of Web services of the group?

We can now define the main problem of our thesis:

How can we sustain the expected QoS of Web services in a group of similar Web Services?

1.2 Main Questions

Thesis Topic

We have defined the topic of this thesis as follows:

How can we sustain the expected QoS of Web services in a group of similar Web Services?

There are two main tasks addressing this question:

- *User request management* deals with the situation when the group of Web services receives the request from the client (Section 1.2.1).
- *Member management* deals with the management and organisation of the Web services in the group (Section 1.2.2).

1.2.1 User Request Management Questions

When the group of Web services receives the request, the first question that the group needs to have answered is which entity is responsible for selecting the Web service to handle the request and how this selection is made. There is also the problem of observability for this entity. This determines whether the entity is aware of all Web services in the group or only a subgroup of Web services. A selection mechanism is thus needed to answer this question. This selection mechanism needs to take into consideration the QoS requirements of the request along with the QoS properties of the Web services within the group. The selection mechanism also needs to consider other possible considerations other than matching the QoS, an example of this is whether a Web service has been given too many (or too few) requests to handle.

Once a Web service has been selected to handle the request, the next question concerns the monitoring of the selected Web service. The group of Web service needs to ensure that the selected Web service is handling the request at the pre-agreed upon level of QoS. How this monitoring of QoS is done can be answered by a monitoring mechanism. This monitoring mechanism allows the group of Web services to determine when and if a substitution is needed to handle the request.

In the event that a substitution is required, the next question would then be to choose among the remaining Web services a substitute Web service which would be suitable to handle the request. A substitution mechanism is thus required to answer this question.

1.2.2 Member Management Questions

In order to better manage the Web services in a group, it is important to know what kind of requests the group will be receiving. The first problem is thus whether it is possible to be able to create an accurate client model that is able to anticipate the clients that the group may receive over time. This way the group can plan accordingly.

A member management mechanism is required to monitor and control the quantity and type of Web services in the group. Such a mechanism will allow the group to sustain its QoS over a longer period of time, depending on the requests sent by the clients. The mechanism should be able to evaluate the quantity and type of requests that it is receiving from the clients over a period of time. This allows the group to best serve the clients by adjusting the quantity and type of Web services in the group.

Finally, it is helpful to know how the group of Web services is to be represented. An evaluation mechanism allows a method of computing the QoS of the group as a whole. This allows the comparison of how the group of Web services is doing over time, or also acts as a possible comparison between groups of Web services. An additional reason for this is that it allows an accurate publishing of the group in the Service Registry. This allows potential service clients to have a better idea of the level of QoS that the group of Web service can provide.

1.3 Methodology

In this section we describe the methodology that we use in our thesis. In the introduction, we had presented the framework and scope to which we intend to sustain the quality of Web services. We had proposed a solution by using a group of Web services and broadly categorized the main question into two parts, the user request management and the member management problems.

We plan to look at the literature that has been other presented by other authors which attempt some of the questions that we asked. We plan to look at the solutions presented and compare their pros and cons to determine the best solution in our thesis. This research is done across as many domains as deemed necessary, this includes the domains of Service Oriented Architecture, Multi-Agent Systems, Auction Theory, Utility Computing and Cloud Computing. This is done in order to better explain how the contribution for this thesis sits relative to the current literature.

Adopting some of the ideas in the current literature, we then present a framework that sustains the quality of Web services. Specific mechanisms can then be built upon this framework in order to answer some of the user request management and member management questions.

Using these mechanisms and the framework presented, we run several simulations in order to best illustrate the benefits of our approach. In order to best substantiate the experiments, we plan to use real-world data as far as possible.

In addition, at the end we plan to compare our proposed solution from a qualitative view with our solutions that sustain the quality of Web services.

1.4 Thesis Organisation

In this section, we present the organisation of this thesis. We first present the organisation of the related work related to the three main tasks. We do this in Section 1.4.1. Before we can address the main tasks in this thesis, we need to present preliminary concepts and techniques that will help address the main tasks. These preliminary concepts are presented in Section 1.4.2.

We start our organisation for our contribution in this thesis by presenting our solution in Section 1.4.3. In Section 1.4.4 we present the organisation for mechanisms dealing with the situation when a request is sent to the group of Web services. The second task, the organisation for the member management is next presented in Section 1.4.5. The third main task, the discussion is presented in Section 1.4.6.

Finally, we conclude with the thesis organisation of our conclusion in Section 1.4.7 as well as look at potential future topics that can be built from this thesis.

1.4.1 Related Work

In Chapter 2, we discuss theW work related to this topic. We divide this section based on the related work for each chapter in our work. We first describe the related work concerning the user request management. The related work for the user request management is broken into two different sections, one for the selection mechanism, and another for the substitution mechanism. The related work concerning the member management of the group of Web services comes next.

1.4.2 Preliminaries

Before we introduce the main ideas of our approach, in Chapter 3 we introduce preliminary concepts and techniques and ideas that help to solve the main question. These concepts and techniques are needed to better understand the approach in this thesis. The concepts and techniques that we define in detail here are:

1. Architecture - The Web service interacts with other parties such as the service clients and other parties. In this section we define the architecture which the Web services operates in.
2. Quality of Service properties - Defining the QoS properties and the notations that we use in this thesis. The QoS properties include the QoS requirements of the request.
3. Service Level Agreement - The contractual agreement that binds the client to the group of Web services. Since the group of Web services consists of multiple Web services with different QoS properties, the SLA involving a group of Web service requires detailed elaboration. In addition, we mention what is required of the contractual agreement between the individual Web service and the group.
4. Auction Theory - Our approach includes the implementation of auction theory into the domain of Web services. In this section we look at basic auction theory such as the unique properties of different types of auctions.

1.4.3 Solution

In Chapter 4 we introduce our proposed solution to the problem of sustaining the quality of Web services. We elaborate on the framework chosen to form the group of Web services along with its architecture and some basic mechanisms. We go into detail on the roles and responsibilities of each component in the grouping, along with their relationships and interactions with each other and the client. This sets up the basis of which our user request management and member management mechanisms can use in order to sustain the QoS of Web services.

1.4.4 User Request Management

We proceed to explain the first of the main tasks in Chapter 5. We discuss selection management mechanisms when a request is submitted to the group of Web services. These selection mechanisms select the Web service to handle the request based on several factors including the QoS of the Web services as well as the QoS requirements of the request itself. The selection mechanism will also have to consider how one particular selection might affect the group of Web services as a whole. We limit the discussion of the selection mechanisms to only involve Web services already in the group.

In line with the goal of sustaining the QoS, it is crucial to define a substitution mechanism to handle the situation when a Web service fails. The substitution mechanism includes monitoring the QoS of the selected Web service when a request is being handled. Also included in the substitution mechanism is selection and replacement techniques in the event that the originally selected Web service were to fail. These selection and replacement techniques are important in the sense that aim to minimize the negative impact of a failing Web service. We limit the discussion of the substitution mechanisms to only involve Web services already in the group.

Experiments are conducted in order to better illustrate the benefits of our approach. They provide a quantitative comparison using real-world values.

1.4.5 Member Management

The member management of the group of Web services is the next main task described in in Chapter 6. The first step in managing a group of Web services lies in the requests that are received by the group. Thus, the first step in figuring out how to manage the group of Web services involves being able to accurately model the clients. An accurate model will allow the group of Web services to be able to effectively predict the number and type of requests that the group receives.

Another important element in the member management of a group of Web services is the ability to evaluate the QoS of a group of Web services as a whole. While this can be done by considering the QoS property of each Web service within the group, certain QoS properties require special care due to their definition. The ability to evaluate the QoS of a group of Web services as a whole gives the group a global view of how it is doing relative to the requirements of the clients over time. More importantly, the group of Web services can use this metric to determine the best Web services to admit into the group.

Next, we discuss mechanisms and heuristics to determine how best to manage the members in the group. They also allow the possibility of inviting more Web services into the group. These mechanisms are used to determine the optimal number and quality of members within a group. It is important for the mechanisms to remember to consider the QoS requirements of clients over time. Since clients act independently, they may increase or decrease the number of requests that they send to the group of Web services. It is also highly likely for the QoS requirements for the clients to change over time, since the clients themselves may change.

Finally, experiments are conducted in order to better illustrate the benefits of our approach. They provide a quantitative comparison using possibly real-world values.

1.4.6 Discussion

Having proposed a solution in order to sustain the QoS of Web services, we take a look at qualitative differences between our approach and other approaches in Chapter 7. In this chapter we look at the differences from both the user request management and the member management level.

1.4.7 Conclusion and Future Work

We include a chapter to conclude the thesis by summing up our approach to sustain the quality of Web services. In the final chapter, we look at possible future topics that might be worth looking into. Some of these possible future topics could be raised due to the limitations due to our approach to sustaining the quality of Web services.

1.5 Contribution

In this section we list the specific contributions that this thesis provides to the research community. We first list the user request management contributions in Section 1.5.1. Next we present the member management contributions in Section 1.5.2.

1.5.1 User Request Management Contribution

Our contribution for user request management are:

- A framework of a group of similar Web services that is used to sustain the quality of Web services
- The use of auction theory in a group of similar Web services and how this can improve the selection and substitution
- A unique mechanism to determine how to select and substitute Web services in a group of similar Web services
- A mechanism to monitor the QoS of Web services within a group of similar Web services

1.5.2 Member Management Contribution

Our contribution for member management are:

- A mechanism in order to predict the number of requests and type of requests that a client sends to a group of similar Web services
- A unique mechanism in order to determine which Web services to admit and expel from a group of similar Web services
- A mechanism to determine the Quality of Service of a group of similar Web services

2

Related Work

In this chapter, we discuss alternatives related to our work that have been presented by other authors. These alternatives are presented and a discussion follows on their pros and cons. We have divided this chapter as such:

- In Section 2.1, we discuss the related work regarding the selection mechanism used in the user request management.
- In Section 2.2, we discuss the related work regarding the substitution mechanism used in the user request management in order to sustain the quality of Service in a group of Web services.
- In Section 2.3, we are focused with the related work regarding the member management techniques.

In each section, we compare the solutions provided with each other, discuss the advantageous and disadvantageous of each solution, and explain how it can be adopted to our solution. This way, we can position our thesis with regards to the current research developments and present the contribution of this thesis.

2.1 User Request Management - Selection Mechanism

In some cases, although the client sends the request to a group of service providers, it retains control over the selection of the service provider. The group of service providers thus only act as a directory, merely providing information to the client. A combination of both approach is also possible. We look at the related work on the selection responsibility in Section 2.1.1.

When a request is sent to a group of Web services, a selection entity is responsible for making the selection to determine which Web service in the group will handle the request. The selection entity can make the best selection only based on the information that it has on the service providers in the group. We first look at the information available to the selection entity(s) in Section 2.1.2.

In certain organizations, there is only one central entity who is responsible for the selection process, we consider this as a central coordinated approach. There is also distributed coordination approach where different entities have the responsibility for making the selection. Both approaches have their pros and cons. We look at the related work in Section 2.1.3.

In some cases, when a request is received, it is not necessary to be allocated immediately to a service provider. Some authors prefer to wait till a number of requests are received before allocating them to the various service providers, we call this batch mode. In the case when requests are allocated immediately, we call this immediate mode. The related work on this, especially in the field of grid computing is discussed in Section 2.1.4.

Next, we consider the selection priority of the selection entity. Specifically, we are concerned here with whose satisfaction is the selection entity trying to maximize when making the selection. This concern is evident in the selection process in the domain of Utility Computing, where one service provider is selected to handle the request that is sent from a client. Even though the work done in Utility Computing is mainly focused on computing resources (bandwidth, CPU cycles, etc), there are similarities in terms of choosing the service provider(s) to handle the request sent by the client. In addition, we present several examples in the cloud computing domain as well. We present this work in Section 2.1.5.

Our approach involves the use of auction theory. However, there are different possible ways where auction theory can be incorporated into the selection process. In Section 2.1.6 we discuss the related work various ways auction theory has been incorporated into the selection process and explain how our approach is similar, or different from other solutions.

Lastly, we supplement the material on how the selection is made by looking at the related work regarding utility functions. We look at the existing selection mechanisms in Section 2.1.7 and see how they can be applied to our proposed solution.

2.1.1 Selection - Responsibility

We consider the entity that is responsible to make the selection as the selection entity. In some cases, this responsibility is given to the service client itself. Allowing the service client to make its own selection can be easier for the service client to accept its own consequences (or credit) for making the selection, but this defeats the purpose of having a group of Web services in the first place.

In the domain of Service Oriented Architecture [46] which we elaborate later in Section 3.1, the service registry acts as a depository where service providers can register themselves. Service clients, in search of a service provider, can consult the service registry for information on service providers that might be able to fulfil the functional requirements of the requests. The eventual selection of which service provider to handle the request is left up to the service client.

Service brokers exist in the domain of Service Oriented Architecture to act as a middleman between the service client and service providers. The service brokers thus have the responsibility to select the Web service to handle the request after it receives the request from the service client. This selection is typically done with the eventual QoS provided to the client in mind [127,133]. In the domain of Multi-Agent Systems, [67] defines facilitators as an agent which gathers information about other agents and selects agents to do certain tasks. In the domain of cloud computing, [135] calls such middlemen *proxies*.

[85] proposes a different way of selection. A recommendation system receives the request from a service client and generates a list based on its own recommendations as well as a feedback system depending on the history of the service providers. It trims the full list down depending on the request from the service client and presents this list to the service client. The eventual selection of which service provider to handle the request is still performed by the service client, however the service client only selects from a subset of the full list of service providers.

For our thesis, the context of our work involves having the service client select the group of Web services, and specific selection of which Web service to handle the request is under the responsibility of the group of Web services. This approach benefits the client by allowing it to submit the request to the group of Web services and not be concerned about the precise selection. In addition, any possible required substitutions (See Section 2.2) is left to the group of Web services.

2.1.2 Selection - Full or Partial Observability

We define *observability* as the amount of information a selection entity has on the Web services in the group. If the selection entity has full observability [15, 119], this means that it is fully aware of the functional and non-functional properties of all Web services in the group. On the other hand, if the selection entity only has partial observability [25, 51, 52, 97, 118, 126], it is only aware of the functional and non-functional properties of a sub-group of Web services.

Full and partial observation can affect the selection mechanism for the selection entity. Since full observability gives the selection entity complete information on the entire group, the selection entity is able to make what it considers the most optimal selection. A selection entity with partial observability is limited to a sub-group of Web services to which it has knowledge for, and thus, a more optimal Web service could possibly exist within the group but outside of the sub-group of Web services.

In Figure 2.1, we look at the difference between full and partial observability. In this figure, we consider the organisation where there is only one selection entity. In the example here with full observability, the selection entity is aware of the functional and non-functional properties of all 3 Web services within the group. In the case of partial observability, the selection entity is only aware of the functional and non-functional properties of only 2 Web services within the group.

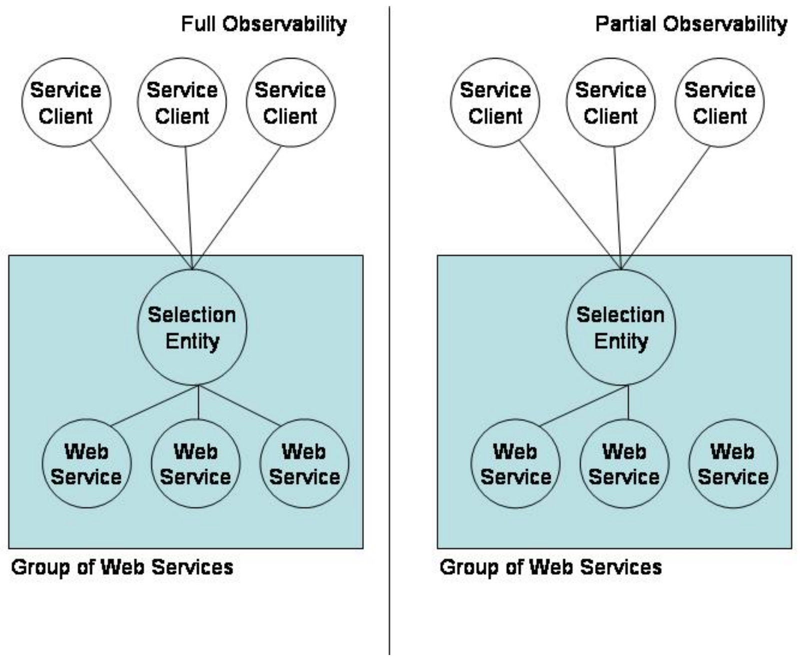


Fig. 2.1. Difference between full and partial observability

The problems with full observability lie in updating of information to the selection entities. In the event that there is more than one selection entities, the updates on functional and non-functional properties of each Web service is required to be sent to all the selection entities. This can be heavy on the network, depending on the size of the group as well as the frequency of updates. This can contribute to scalability problems since as the size of the group grows, the amount of traffic to and from the single selection entity grows as well. The partial observability approach does not suffer from the full extent of these problems since each selection entity is not aware of all Web services in the group. The extent of these problems highly depends on the organisation of the group of Web services (See Section 2.3.1) and the method of updating the information. Since the selection entity is not aware of all information about the entire group, the network bottleneck is not as severe as the group grows. The type of organisation (See Section 2.3.1) can also heavily influence the extent of these problems.

Our approach in this thesis takes upon the full observability approach. The advantage of full observability is that it allows the highest satisfaction among all parties involved. We couple this with a central selection approach where there is only one selection entity in the group of Web services, we look at the related work on a central of distributed selection process in the next section.

2.1.3 Selection - Central vs Distributed

When it comes to whether a central or distributed selection process is conducted, a large part depends on the type of organisation of the group of Web services. We describe later in Section 2.3.1 some possible implementations of how a group of Web services could be organised, but in this section our focus is on a central and distributed selection process.

The difference between a central and distributed selection process is in the entity that is responsible for making the selection. In a central selection process [15, 88, 130], the selection of the Web service to handle every request that is submitted to the group is made by a single entity. In the domain of MAS, [56] calls this a federated or central federated organisation.

In a distributed selection process , we consider two different types of distributed selection processes. The first is a distributed singular selection process, this is the situation where only one entity is responsible for each request, however, different requests may be handled by different entities [112,118]. [56] defines this type of organisation as a congregation. In the second case of a distributed multiple selection process, multiple entities are responsible for the selection process of each request [52]. Both distributed selection processes share similar properties. In Figure 2.2, we present a graphical representation of the difference between central and distributed singular selection process.

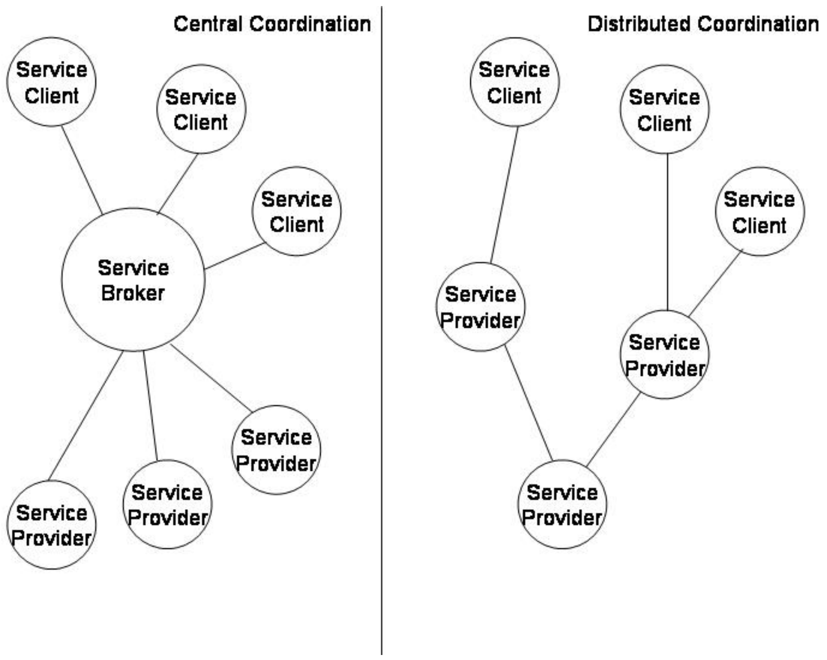


Fig. 2.2. Difference between central and distributed singular selection process

In this thesis, our solution takes the centrally coordinated approach due to its ability to provide a more optimal solution. Some of the mechanisms that we introduce can help mitigate some of the problems that come with the centrally coordinated approach. Naturally we accept that not all problems will be solved, for example, in our approach, if the selection entity were to fail, the entire group will fail to function. We go through some of the differences between central and distributed selection processes here:

- Scalability: In a central selection process, there are scalability concerns since all requests are sent to a single entity. As the number of requests increase, the single entity is faced with additional workload. This is in addition to possible increase in network traffic that is required due to the increase amount of communication between client and the single entity. There will also be an increase network between the single entity and the Web services in the group. In the distributed selection process, there is no single bottleneck that might occur. The network traffic and selection processes are divided among the multiple entities.
- Security: In a central selection process, the single entity may be the target of any security breaches. This is because the central entity is knowledgeable of the entire group. In a distributed multiple selection process, since the responsibility of selection is distributed across multiple entities in the group, or a distributed singular selection process, it can be harder for a malicious entity to breach the security of all entities involved in selection.
- Reliability: In a central selection process, since there is only one entity that is responsible for the selection of Web services in the group, this entity acts as a single point of failure. This means that if this entity were to fail, the Web services in the group will be unable to receive requests and send the corresponding replies. A single point of failure does not exist in the distributed selection process. If one (or a relative small number) of selection entities fail, the group can possible remain functional with limitations.

2.1.4 Selection - Immediate vs Batch

[83] describes two mapping heuristics, immediate mode (or dynamic) and batch mode (or static) heuristics to determine the matching of tasks (or requests) to heterogeneous distributed computing systems. In immediate mode [102, 137], requests are allocated to their respective service provider immediately once it is received by the system. On the contrary, in batch mode [28,44], a number of requests are received before allocation is done.

There are pros and cons for each approach. Since the immediate approach allocates the tasks immediately when it is received, there is no lost time for that task. Batch mode receives tasks and holds onto them for a period of time, or until a certain number of tasks is reached before deciding the allocation. The length of waiting time or number of tasks depends on the system administrator. While the tasks are waiting to be allocated in batch mode, this represents time lost for each task. This is because the tasks are simply sitting there waiting for the system to decide when to allocate the task. This can be an issue for time sensitive tasks that require immediate action.

However, since batch mode allocates a set of tasks, a more optimal allocation can be done compared to the immediate mode approach. In Figure 2.3 we show a quick example of the problems with immediate mode. In this example, there are 3 indivisible tasks - task 1 of size 3, task 2 of size 4, and task 3 of size 5. We also have 2 providers that are responsible for handling the tasks, they are of sizes 5 and 7 respectively. The system cannot predict nor is aware of the sizes of tasks nor their sequence of arrival. In a immediate mode, the selection entity may allocate task 1 to provider 1 upon receiving task 1. When the selection entity receives task 2, since there is insufficient space in provider 1, the selection entity has no choice but to allocate task 2 to provider 3. This leaves no possible solution when task 3 of size 7 is received.

On the other hand, on the right side of Figure 2.3, we show the optimal solution for a batch mode after all 3 tasks have been received. Due to the allocation of all 3 tasks at the same time, the selection entity is able to fit tasks 3 into provider 1 and tasks 1 and 2 to provider 2.

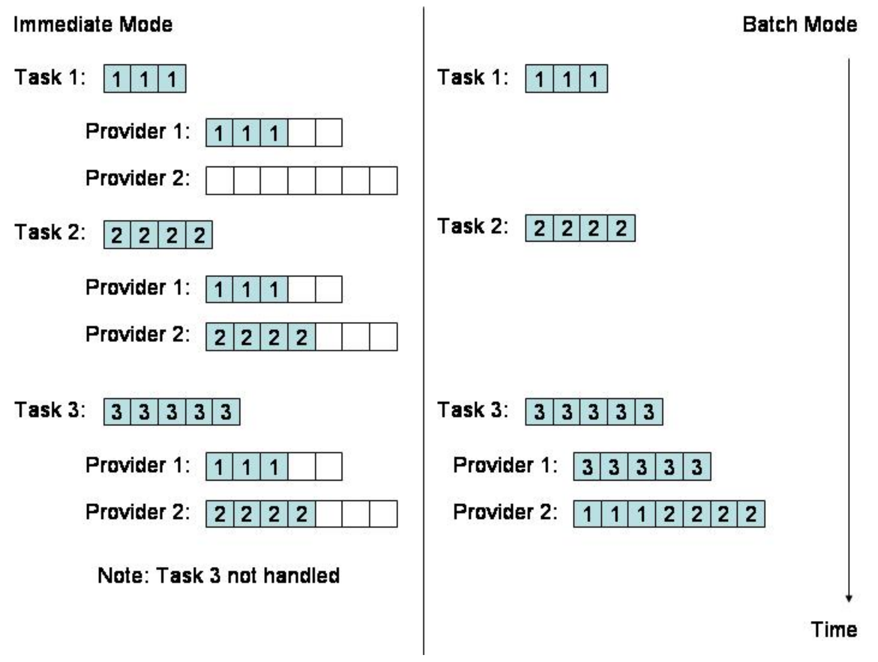


Fig. 2.3. Example of difference between immediate and batch mode task allocation

In our thesis, for a group of Web services, we undertake the immediate mode approach. One main reason is due to its ability to issue requests immediately, no time is wasted waiting for other requests to enter the group.

2.1.5 Selection - Priority

The selection of Web services in a group of Web services bears a similar resemblance to the selection in the domain of Utility Computing [60]. In utility computing, there are producers and consumers which act similar to service providers and service clients in the domain of Web services. Rather than providing a service, producers have resources which the consumers require. Brokers provide strategies to allocate resources from producer to consumer when the consumer requests for them [18]. The main difference in Utility Computing is that the resources (clock cycles, bandwidth, memory, etc) provided by one provider is considered the same as another. This is not the case in Web services, since the QoS provided by each Web service is different.

In cloud computing, a couple of papers [38,135] propose a service client and service provider relationship. The service brokers select the service provider based on maximizing the QoS that is returned to the client.

In Utility Computing, there are 3 main approaches for managing the allocation of resources: user-centric; system-centric and economy-centric.

- User-centric systems [6,29,30,38,135] allocate resources to yield the highest maximum utility to the clients of the system based on their QoS requirements. The maximum utility is determined by an aggregation of the utility function which each client provides. It is up to the individual client on how much utility each resource will provide to the client. The focus on such systems is on the client, with the goal of maximizing the satisfaction of the clients. An example on how this can work is by choosing the Web service that gives the shortest execution duration. This is done regardless of the cost of the Web service but allows the client to receive the highest utility.
- System-centric systems [10, 22–24, 26, 64, 78] allocate resources to yield the highest maximum utility for the entire system. In system-centric systems, the utility of each resource is determined by the system. This kind of systems typically do not take price into consideration. Neither are they concerned with how much utility each resource will provide to different clients. The utility is typically based on properties such as how many requests the system can process or the percentage of the resources that are being used. An example of how this can be done is to choose smaller requests so that more requests are handled. Regardless of the QoS required by the clients of these smaller requests, they are handled in order to maximize the number of requests that the system handles.

- Economy-centric systems [90] only consider the price of service and the budget of clients with the goal of maximizing profit. This can be done by always selecting the cheapest service provider and only serving clients who offer the highest price. In Utility Computing, this is straightforward since the resource (clock cycles, bandwidth, memory, etc) is considered to be the same regardless of which service provider provides the resource. In Web services, the QoS has to be considered too, since the quality of resource provided by different service providers is not the same.

There are some systems that attempt to combine the focus on economy and client [17, 114, 116]. [116] does this by creating a utility function that considers both the utility of the client and the utility of the system. This utility function is then used to allocate the resource to the clients. However, to the best of our knowledge, there are no systems that attempt to combine all 3 approaches within the domain of Web services. In our thesis, we propose a mechanism that considers the selection of a service provider by taking into account the client, the system, as well as the economy.

2.1.6 Selection - Auction Theory

Within the domain of cloud computing, Dastjerdi in his PhD thesis [37] proposed that brokers submit for a call to bid and for service provider to reply with their offers (price). He goes further to propose that counter offers are made to the service providers. At a certain deadline, these offers are sent to the service client who makes the final decision on which service provider to handle the request.

[60] mentions variable and fixed pricing models. In a fixed pricing model, the price for a resource is fixed, and the clients pay the same amount regardless of which service provider is providing the resource. On the other hand, in a variable pricing model, the price for a resource is not fixed, thus clients may pay different prices on the same resource. The variable pricing model leads to the possibility of an auction style bidding process for the resources.

There are several Utility Computing literature that use a bidding process as part of the selection process [6, 30, 99, 123]. This bidding process can be adopted and implemented according to some of the bidding process suggested in the literature of Utility Computing.

Later in Section 3.4, we go into further details on the properties and characteristics of each type of auction and how they can be implemented in a group of Web services. We show the benefits of such a system and possible further extension into other frameworks in the Web services domain.

2.1.7 Selection - Utility Function

The idea behind a composition of Web service allows more complex Web services to be created from simple Web services. By aggregating simple Web services with different functionalities together, more complex services can be provided. The workflow [3], or the flow of the Web services are defined by the selection process. This is in contrast to a grouping of similar Web services since the Web services provide the same functionality.

The selection process in composition of Web services mainly deals with allocating a utility function to determine the utility of a specific workflow [4,5,50,58,79,96,110,130,132]. The utility function is user-centric in the sense that its goal is to provide the highest amount of utility to the user based on the request's requirements. The method taken is to provide utility functions depending on the type of workflow pattern (sequential, combinatorial, conditional, etc [3]). Such workflows do not exist in our approach since there is no such workflow in a group of similar Web services. However, the idea of a utility function is adopted in our approach.

This idea has also been proposed in the domain of Cloud Computing [135] where the maximization of the QoS level and the minimization of cost occurs. [92] uses a multi-criteria decision making technique called Analytic Hierarchy Process [103]. This technique uses complex weighted sum functions that are built in a structured manner. The process allows for mandatory and optional

QoS constraints to be met during the selection process. They also introduce an exact and close criteria whereby certain property values are sufficiently close to the requested value, while others property values are required to be exact.

One good idea is the concept of utility functions mentioned in [130], where a certain selection results in a certain amount of utility for the user. Different selections result in different amounts of utility. This is similar to our thesis where a certain score is given to a certain selection. However, the focus in our approach is not only on the QoS provided to the clients.

[94] takes a rather unique approach by considering the utility that is provided to the client as well as any possible degradation towards current tasks that are being serviced. Specifically, the selection process attempts to maximize the utility provided to the client then minimizing the possible degradation of QoS for tasks that are currently being handled. In our thesis, even though we undertake the immediate mode of allocating requests to Web services as they are received by the group. We consider that any additional tasks that are allocated to the Web service does not degrade the level of QoS that the Web service can provide as long as the Web service has the *capacity*. We go into further detail on our proposed definition of *capacity* later in Section 3.2.

Game theory can be used in the selection process to determine which Web service to handle the request. This has been proposed in the domain of MAS when selecting an agent to handle the task. However, Game theory suffers from the problem that the source of the utility function is rarely discussed [71]. In our thesis, we propose a utility function in order to determine which Web service should be selected to handle the request. We further elaborate on the comparisons between game theory and utility function later in Section 2.3.4 during the discussion on the related work regarding admission and expulsion of Web services from the group.

2.2 User Request Management - Substitution Mechanism

In this section we present the related work by other authors with regards to the substitution mechanism. We first discuss the problem of QoS monitoring, since we need to know if the substitution is required, and when this occurs. This is covered under QoS monitoring of Web services and covered in Section 2.2.1. Next we discuss about the substitution itself. This involves different types of replication which we present similar works in Section 2.2.2. Different types of replication techniques have different pros and cons and we discuss the works by other others to determine which replication technique is best suited for the framework of a group of Web services.

2.2.1 Substitution - Monitoring

There are three general techniques for monitoring QoS.

- A trusted monitor to intercept the messages exchanged [12,36,66,129,133]. This technique involves passing all traffic between the service client and the service provider through a monitor. A monitor is defined as a party that observes the data between two parties. The monitor is responsible for measuring the QoS that is provided by the service provider. One problem with this approach is scalability, since the amount of traffic scales higher as more service provider and service clients are added. [101] regards this as a monitoring technique with no access to the web service implementation. In Figure 2.4 we show the difference in terms of potential traffic in order to illustrate scalability problems. On the left, we present the potential traffic with 2 service clients and 1 service provider, we see that there are only 3 lines of traffic going through the trusted monitor. On the right of the figure, we present the potential traffic with 3 service clients and 2 service providers, we see that there are 5 lines of traffic going through the trusted monitor. As more service clients and service providers are added, the amount of traffic increases, this can cause potential bandwidth issues.

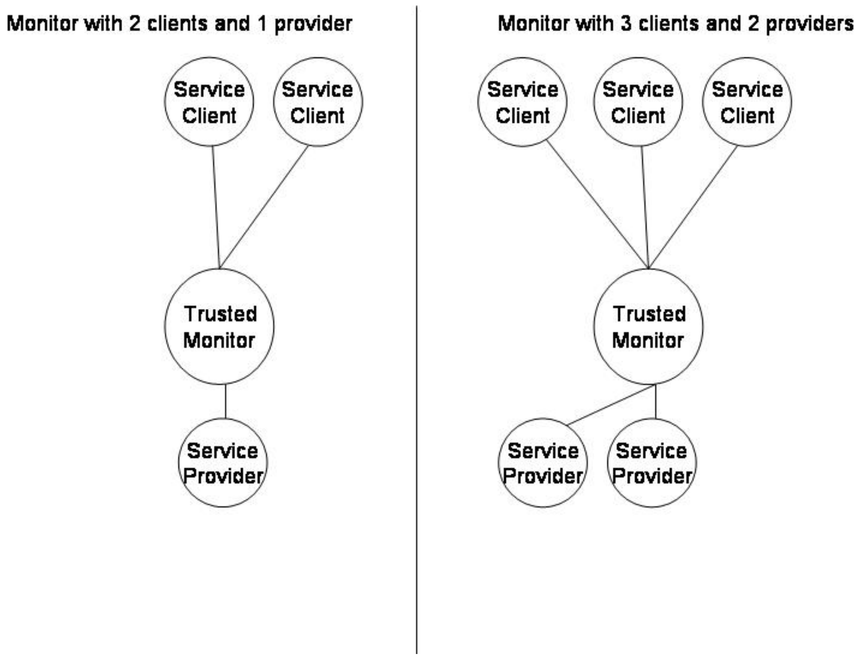


Fig. 2.4. Scalability Problems with a Trusted Monitor

- Monitoring code run on provider's side [7,41,80,81,105]. In this approach, code is run on the service provider as part of the service middleware. This monitoring code intercepts traffic between the service provider and the service client in order to measure the QoS. The monitoring code then computes the QoS. One drawback with this technique is possible tampering of the results by the service provider. A service provider might want to do this in order to project the image that it is providing a level of service that is higher than reality. [101] regards this as a monitoring technique with access to the web service implementation. One problem with this approach is the maintenance of this monitoring code in the event of future updates.

If additional QoS parameters need to be observed, this monitoring need to be updated. This can also get complicated if the service provider has any kind of system updates, the monitoring code needs to be compatible with any version of the service provider in order to function effectively.

- A trusted party can periodically probe and monitor the service provider [54, 75]. This allows the trusted party to get a gauge of the QoS that the service provider can provide. One major difference between this technique and the previous two techniques is that the trusted party does not use the data between the service provider and service client to determine the level of the QoS of the service provider. [62] takes a similar approach, by using the client themselves to act as the trusted party. This trusted party could be a third-party Web service or a service broker that sits between the service provider and service client. The amount of time for each probe is user defined [93].

[121] combines the first two techniques by introducing an external monitor for certain QoS properties while other QoS properties were monitored by sensors attached to the service provider. We undertake this approach because it provides multiple avenues of monitoring the QoS of the Web services within the group of Web services.

[93] uses a combination of the trusted party to probe as well as intercepting messages in order to accurately monitor the QoS of Web services. They argue that the combination of both allow an event-based method of monitoring.

[105] specifically mentions that it is necessary for raw measurement data to be collected in order to ensure the monitoring of an SLA. They explain that this data can be obtained through messages that go in and out of the service provider.

In our thesis, our approach takes the form of a trusted monitor in a group of Web services. The benefits in its ability to accurately monitor the QoS of Web services outweigh the drawbacks of scalability. Another reason we decide to use the form of a trusted monitor is also due to the type of organisation of the group of Web services that we have chosen in our approach (See Section 2.3.1).

2.2.2 Substitution - Replication

Replication forms the core of the substitution process. If the substitution and replication are performed successfully, the entire process is transparent to the client. Replication can be divided into two categories, active replication [108, 109] and passive replication [16]. We elaborate both categories here:

- Active replication - In active replication, more than one Web service handles the same request at the same time. However, there is only one Web service, designated as the primary Web service's reply will be sent back to the client. In the event that the primary Web service were to fail, a secondary Web service will be chosen from among the rest of the Web services (known as replicas) that have been handling the request. In this thesis, we avoid this approach because it might possibly waste the resources of other Web services within the group in case a substitution is not needed. In the domain of active replication in Web services, [127] introduces a middleware that allows active replication in Web services in order to improve reliability, although the author does not address other QoS properties. The middleware communicates directly to the client, representing the Web services that it represents.
- Passive replication - In passive replication, only one Web service is handling the request (primary Web service) while an allocated number of Web services are acting as backups. A failover happens when the primary Web service fails and one of the backups take over [16]. [31] further divides passive replication into hot and cold. In hot passive replication, during the execution of the primary, data is sent to the backups. Thus if the primary were the fail, the backup systems do not need to start from the beginning of handling the request and can pick up from where the primary failed. In

cold passive replication, during the execution of the primary, data is not sent to the backups. Thus if the primary were to fail, the backup system would have to start from the beginning. In this thesis, requests to Web services are modelled as atomic [45], this means that they cannot be broken up to be handled by different Web services. Thus, only cold passive replication can be applied. In the domain of passive replication in Web services, [76] introduces a middleware that allows passive replication in Web services.

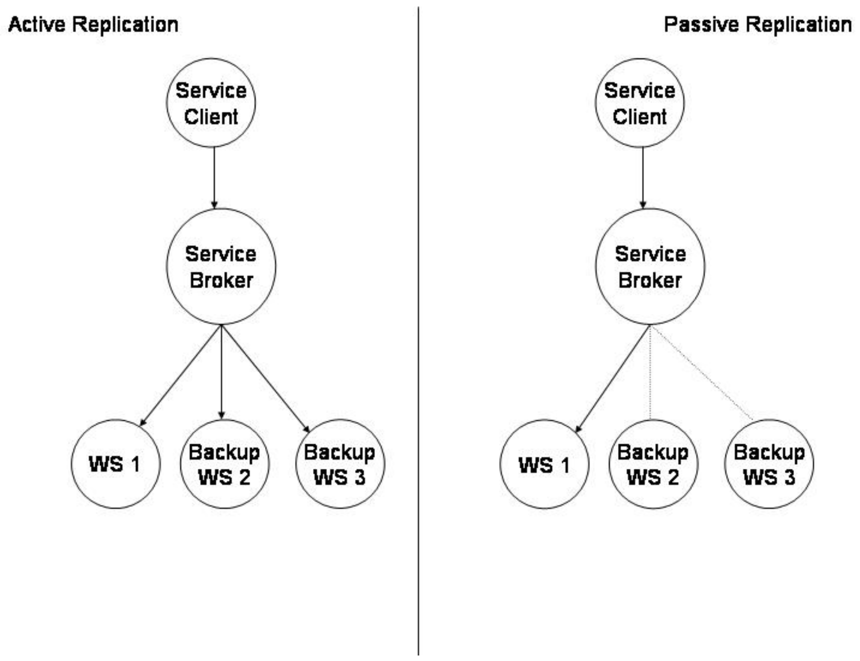


Fig. 2.5. Difference between Active and Passive Replication

In Figure 2.5, we illustrate the difference between active and passive replication. In this figure, for both the active and passive replication examples, we have one service client sending the request to the service broker. We also have a single Web service that has been selected to handle the request, denoted by

(WS 1) and 2 backup Web services, denoted by Backup WS 1 and Backup WS 2. On the left of the figure we show active replication where the service broker sends to the backup Web services the request so that the backup Web services can handle them at the same time as the selected Web service. On the right of the figure, we show passive replication, whereby the service broker puts the backup Web services on standby but do not send the requests to them.

[63] does a combination of the two, which they call a semi-active replication. In this form of Web service replication, one service actively replicates while the rest acts as a backup system. The authors argue that this semi-active approach allows less network traffic. In our thesis, the main focus is not on the network traffic since networking requirements and considerations is beyond the scope of this thesis. Thus we do not take this approach, but stick to the passive replication approach.

[136] goes further by categorizing the different replication strategies and proposes an evaluation framework in order to select the best strategy. In our thesis, this is not necessary since passive replication is used.

2.3 Member Management

There are different ways of grouping Web services together, simply randomly throwing a bunch of Web services together can be inefficient. A proper structure of a group and the relationships of its components have to be defined. This area has been extensively explored in the domain of Multi-Agent Systems. We look at the literature regarding this in Section 2.3.1.

Communication is vital in order to group Web services together. Different models of such communication exist and we present the related work in Section 2.3.2. The related work and proposed approach for communication is applied as well to both the selection and substitution mechanisms, but we present them here in the member management section because it involves the overall operation of the group of Web services.

Next in Section 2.3.3 we look at different prediction mechanism that predict the requests requirements sent by clients. In this thesis we elaborate on a simple prediction mechanism however any of the more complex prediction mechanisms can be used.

In Section 2.3.4, we present the related work on the admission and expulsion process from other related work, mainly in the domain of Multi-Agent Systems. These processes determine which agents to admit and to expel from the group.

Finally, in order to evaluate the group of Web services, there needs to be a mechanism to evaluate the QoS of the group of Web services as a whole. This serves as a way of comparing the group against itself over time, as well as comparing the group of Web services with other Web services (either as a group or individually). This evaluation additionally serves as a method of allowing the group's QoS to be represented on the service registry. We presented the related work in Section 2.3.5.

2.3.1 Organisation

The concept of organising a group of different parties is considered a major issue [48] within the domain of Multi-Agent Systems (MAS). One of the earliest literature to attempt to define the organisation within a MAS can be found in [49]. In this paper, the definition of a *role* is defined which gave the roles of *manager* and *bidder* to the two different components to a type of bidding system.

[35] defines a MAS as a society that can be broken down into two aspects: its population and its organisation. The paper also formally defines an organisation structure [131] of a MAS as a tuple consisting of a set of organisational roles that agents perform within the system, and a set of organisational links between roles. [131] refers to the links as organisational rules within the system.

[42] considers agents in MAS to be independent if they behave in a manner that follows their own agenda. In addition, agents are in communicative cooperation if they communicate with each other in order to cooperate towards a certain goal. They define a deliberative system as a system where the agents cooperate together and behave in a manner to contribute towards a certain goal. Negotiating systems have an additional element of competition among agents.

In our thesis we argue that roles need to be clearly defined in a group of Web services in order to sustain the QoS of Web services. We define both the population and organisation of a group of Web services, each acting as independent parties with a certain agenda. In addition, the relationship (organisational links [35] or organisational rules [131]) between the different components that form the group of Web services are also clearly defined. Our proposed solution presented in Chapter 4 is a negotiation system where the parties in the group of Web service jointly plan their actions yet an element of competition between parties exist.

Our proposed solution resemble the architecture presented as federations within the domain of MAS. In federations, a group of agents cede a certain level of autonomy to a single entity [56]. In this case, the agents allow this single entity to represent them, and this includes the responsibility of receiving requests and sending replies. In addition, the responsibility of rejecting tasks is also given to this single entity. Group members interact only with this single entity, and all outside communication is done through this entity. This is in line with our approach revolving around central coordination as explained earlier in Section 2.1.3. Compared to the coalition or congregation organisations, the federation is more hierarchical. Another advantage of the federations is from the perspective of the clients that require the service of a federation. Since the *leader* represents the federation, it can do so in a single consistent interface with the clients. This avoids the problem of the client being confused with different protocols and interfaces when dealing with different Web services.

In [56], different organisations allocate different interests for each agent. Agents within a congregation and a coalition act on their own best interests. While agents in a team act on the interest of the team. Our approach is similar to the federation where each agent in the federation is acting on its own best interests, however, the mediator (leader) of the federation acts on behalf of the federation and thus acts in the best interests of the federation. In our proposed organisation, the leader of the group acts in the best interests of the group however each Web service in the group acts in the best interests of itself.

The MAS federation Metamorph [86] calls such an entity the mediator. The responsibilities of the mediator can be divided into two, a high-level mediator is responsible for inter-group coordination while the low-level mediator is responsible for intra-group coordination. Their approach for selecting the agents to join the federation is similar to selecting an agent to handle a task. In our approach we propose something similar where the responsibilities of both intra-group and inter-group coordination are handled by one party. This is similar to the responsibilities of the facilitator which was described in [67]. Metamorph also allows these mediators to remove itself from the federation once the binding between agents have been performed, however this is not an approach we undertake. In our approach, the mediator (broker in our approach) remains a key component of the group of Web services.

2.3.2 Communication

Web services have a machine-processable interface that allows machine to machine communication [53]. In our thesis we consider Web services to be able to use Web technologies [34] such as Simple Object Access Protocol (SOAP) and Hyper Text Transfer Protocol (HTTP) in order to communicate with each other. The exact language used between the machines can be customized but this is beyond the scope of this thesis. In this thesis we assume that there is a viable method for Web services to communicate with each other.

However, we still need to consider different possible communication systems that can be implemented in the group of Web services. Different communication systems allow different sets of Web services to communicate with each other. The notion of *view*, which is the membership of a group was introduced in [13]. This allows different sets of entities within a group to communicate with each other if required.

Within our proposed organisation that resembles closely to a federation (See Section 2.3.1), there is a leader of the group that represents the group of Web services. The two types of communication for our proposed organisation is thus single-point to single-point communication and a broadcast type of communication. This allows the leader to contact each Web service individually and also to broadcast a message to all Web services in the group.

Multi-point to multi-point communication does exist in the form of group communication [27] but this is not required in our proposed approach. We note that [107] states that although group communication is not required by any replication technique, it largely simplifies the implementation.

2.3.3 Client Prediction

The purpose of predicting the clients is to ensure that the group of Web services is able to better prepare for the requests that it might be receiving. Although no prediction model can claim 100 percent accuracy, the more accurate the prediction is, the better the group of Web services can prepare and thus better sustain the QoS provided to the clients. In this section we describe several prediction models

One such prediction model is the Markov Chain [55] model. This model is based on memoryless state changes where the next state is dependant on the current state and not on its preceding states. Each state in the Markov Chain model could refer to the level of QoS of the requests. A state transition could occur from one level of QoS of the request to another based on the satisfaction of the client. The satisfaction of the client could be determined by the number of requests that were handled by the group of Web services, and the level of which these requests were handled.

The field of econometrics [125] is focused on the prediction and analysis of statistical data. The proposed model in this paper uses a regression function in order to predict the clients' requirements. This gives a higher emphasis to recent values and a lower emphasis to older values. This approach best predicts the future requirements of the clients. Other prediction models exist in econometrics, depending on the properties, but its implementation into field of Web services is beyond the scope of this thesis.

2.3.4 Admission and Expulsion

In terms of forming a group of agents, there is a lot of literature done on the formation of coalitions. MAS defines coalitions as agents that have different functional properties [56]. These algorithms also consider how agents that can perform different tasks can come together to form the coalition [71]. Each coalition can therefore perform more tasks than each individual agent, since the potential for the tasks to be performed sequentially or in parallel is there. In our approach we only group functionally similar web services together. This approach allow a better method of sustaining the QoS of Web services. By allowing one individual Web service to handle the requests, we avoid the problem of utility sharing, where the problem of how the payment or reward should be divided among the different parties that contributed. Another major difference between coalitions and our approach is that coalitions are generally short-lived [56] whereas we aim to achieve a group of Web services that live longer.

[111] has a 3 step approach algorithm to determine how a coalition should be formed:

- Their first step is in terms of data collection. Each agent contacts other agents asking for their functional and non-functional properties.

- The second step involves choosing among these agents which agent to accept to form a coalition with.
- Their final step involves the payment when a task has been performed successfully.

Their approach on choosing the agents to form a coalition with consist of a greedy iterative algorithm that considers all possible sizes of a coalition among all possible agents. We consider a similar approach during the admission and expulsion processes by considering all possible Web services that might be in the group.

When agents form coalitions, [111–113] mention the need to coordinate between agents prior to forming the coalition. Terms and conditions need to be agreed upon, as well as possible methods of communication. Another important element to decide prior to forming a coalition is the amount of payment and the payment methods for successfully finishing the task. When forming or dismantling a group of Web services, our proposal has a similar approach in the form of Service Level Agreements. We elaborate on the specific and possible additional elements of the Service Level Agreement later in Section 3.3.

From the agent's perspective, [73, 112] argues that an agent will only form a coalition if the benefit it receives from the coalition is at least as much as the benefit it receives if it were out of it. The paper also mentions group rationality, where the group coalition should benefit by having an additional agent in the group than without it. They do this by considering all possible permutations for a group to form a coalition. The main difference in their approach is that their agents perform tasks consecutively in order to achieve their goal. In our approach, Web services in the group support each other through the use of substitution in order to sustain the QoS of Web services. What we consider is therefore the combination of Web services in the grouping. We adopt the same approach for both individual rationality and group rationality through the use of utility functions.

[69] proposes a different type of coalition formation. In their proposal, they propose designating an agent as a leader with other agents as a member. Other agents which have not joined the coalition are considered as candidates. The selection of candidates is dependent on the amount of previous exchanges the leader has with other candidates. If two or more candidates have the same amount of previous exchanges, then the selected candidate is randomly determined. We argue that this randomization can occur often due to the single-dimensional consideration. Since the only factor that is considered is the amount of previous exchanges, we argue that the possibility of multiple candidates with the same amount of previous exchanges can be high. Our approach avoids this randomization by having a multi-dimensional comparison. We consider several different QoS properties in order to determine which Web service to admit into the group. Regarding the departure from a coalition, they propose that an agent join another coalition if they find a bigger coalition. This is from the perspective of the members in a coalition. However, they do not propose a mechanism whether a leader should expel members of a coalition from the coalition. In our approach, We argue that this is something that is needed in order to maintain the QoS of the Web services in a group. In this thesis we propose such a mechanism.

Several approaches form these groups of agents through the use of game theory [11]. Although game theory allow agents to decide whether or not to form a coalition, it does not specify the algorithms that allow for coalition formation [111]. Game theory suffers from the problem that the source of the utility function is rarely discussed [71]. There are proposals to use an agent's belief system of other agents in order to determine the actions of the agent [70], however, they still lack a concrete utility function. In our thesis, we use a weighted utility function which allows the Web services, or the group, to freely incorporate their belief systems in order to determine which action to take.

As we mentioned earlier in Section 2.3.1, the architecture we adopt is close towards federations. However, [67, 86] describe how such federations could be formed, but do not elaborate on what kind of decisions should be made to determine whether or not to admit or expel agents from the federation.

2.3.5 Group QoS

[2] is the closest in terms of literature which comes to attempting to quantifying the QoS of a group of Web services. The authors analyse this in the context of a grid of Web services and mention the need to measure the availability, the capacity and the utilisation of the network bandwidth. Their approach maps each QoS property into a certain cost, and the QoS of a group can be determined by summing up the individual costs. One problem with this approach is in terms of the mapping, the mapping formula was not clearly defined and we argue that by mapping each QoS property into the cost, each QoS property is not represented in the final value. Our approach in this thesis retains identity of each QoS property.

Within the domain of MAS, [112] provides a approach on how to compute the coalition value of a group of agents. Their implementation considers addictive tasks between agents in order to achieve a common greater goal. Their approach considers the summation of the capabilities of the agents within the coalition. In our approach, each Web service in the group have the same functional property thus the summation of Web services is not relevant. However, we propose a similar method (summation) of calculating certain QoS properties of the group of Web services.

3

Preliminaries

In this chapter, we introduce the details and concepts that we use in our proposed solution. We first introduce the Service Oriented Architecture in Section 3.1. Next, we discuss QoS properties in Section 3.2. In Section 3.3 we look at Service Level Agreements that allow two or more parties to come to an agreement. Finally, we present auction theory specifics that we use in our thesis in Section 3.4.

3.1 Service Oriented Architecture

A set of principles and methodologies in order to design and develop Web services is described in the *Service Oriented Architecture* [46] (SOA). We describe the SOA in this section.

There are three basic components to the SOA. The first component is the client, or the service client in SOA terminology, is the component that requires the service. The second component, the web service, or the service provider in SOA terminology is the component that actually provides the service. The third component aids in the client's search for a Web service, we call the third component the service registry. We presented the components and their relationships with each other earlier in Figure 1.1.

In the SOA, each component acts independently. This means that each component can make its own decisions without consideration or consultation for another component. We describe the three components in more detail:

Service Registry

The Service Registry acts as a depository for Web services who would like to publish its service description. The service description contains the functional and non-functional properties of the Web service. This service description is standardized, based on the Service Registry. Publishing also acts as a form of contract between the service registry and the service provider [53]. This allows clients to go through the service description easily which allows the search for a particular service.

A very simple version of a service registry may just be a simple directory which returns the entire list of service descriptions that have been published. In this case, it is up to the service client to go through the list to find what it is looking for. A more sophisticated service registry may also want to implement a search function for the clients and the service providers.

The request that is sent by the service client may include only functional requirements or may contain both functional and non-functional requirements of the request. This decision is entirely up the service client. A sophisticated service registry could also be set up to handle both types of requests. In either cases, a list of service descriptions that can fulfil the request requirements could be returned to the service client.

The service registry may want to ensure that the service description provided by the service providers is accurate. This involves testing both the functional and non-functional properties of the Web services. Whether or not this is done depends entirely on the system administrator of the service registry. However, the advantage of doing this include having a better trust by the service clients who can be certain of a level of correctness in the service descriptions published by the service registry.

Service Client

The responsibility of the service client is to know the functional and non-functional properties of the request. The service client can then contact the service registry to find a service provider that is able to fulfil the request requirements.

Once a service provider has been decided by the service client, the service client is then responsible for binding directly with the service provider. This is a 2-way process that involves communication between service provider and service client in both directions. Binding involves coming to an agreement with the service provider the exact terms of the service. The functional and non-functional properties should be agreed upon by both parties, along with the payment and penalty details. This information is written down in the service contract, or technically known as the Service Level Agreement (SLA) [74], which we elaborate further in Section 3.3. The binding process also includes sending the request to the Service Provider.

In certain cases, depending on the system administrator, the service registry may require feedback of the service rendered by the service provider. The rating system depends entirely on the system administrator, but the service client will have to reveal to the service registry its satisfaction of the level of service that was provided by the service provider.

Since the service client acts independently, it is possible for the number of requests, and the QoS requirements of each request to change over time.

Service Provider

The service provider is responsible for generating its own service description to be published on the service registry. The service provider then sends this service description to the service registry to be published. This can incur a cost, since the service registry can be viewed as a service which allows other service providers to publish their service descriptions there.

If a service client would like to utilize the services of the service provider, the service client directly contacts the service provider. The service provider will then have to come to an agreement, as described earlier under the description of a service client.

Finally, after the SLA has been agreed upon, the service provider has to provide the service that it has agreed upon.

Service providers have to be reusable, this means that they are able to be reused, either by the same client or another client.

Service providers also have to be stateless, this means that resources are freed once the service has been rendered to a client. This frees up the resources for the next client.

It is possible for the functional and non-functional properties of the service provider to change over time.

3.2 Quality of Service

In this section we present the preliminary view of the QoS properties that are use in other literature as well as those selected properties that are used in this thesis.

We first go through the related work regarding QoS properties in other literature in Section 3.2.1. Next we define the QoS properties that we use in this thesis in Section 3.2.2. Finally, in Section 3.2.3, we define the QoS properties that are associated with a request.

3.2.1 Quality of Service Properties

There has been a wide spectrum of QoS properties that have been introduced by the research community, often with varying interpretations. In this thesis we are focused on technical QoS properties, [134] categorizes this as observable IT metrics, and provider-advertised metrics for the price. We list here the metrics and its multiple definitions where applicable:

Availability - Availability is the probability that the system is up for immediate use [57,59,84]. It is represented as a percentage over an observation

[14, 98, 106, 132]. Availability is related to reliability [98] and Time-To-Repair [33].

Price - Price [57, 79, 132] is the amount of money (currency) that the service provider requires the client to play to use the Web service.

Reliability - Reliability represents the ability of a Web service to perform its required function within a maximum time frame [21, 33, 106, 132]. Probability that the Web service can be completed successfully [57].

Capacity - Capacity is the number of requests that can be handled simultaneously at a certain level of guaranteed performance [33, 98].

Execution Duration - Execution duration is the amount of time between sending the request and receiving a response [14, 57, 59, 79] or the guaranteed average time required to complete a service request [98, 106]. Sometimes, it is referred to as latency [84].

Performance - Performance is defined as how fast a service can be executed [33, 98]. A service is considered to perform well when the throughput is high with a low response time [59, 84].

Security - Security is the ability to provide authentication, authorization, confidentiality, traceability, data encryption and non-repudiation [33, 84, 98]. [91] also includes the resilience to denial-of-service attacks in the definition of security. Length of the key [59].

Accessibility - Accessibility is how well a Web service is capable of servicing the client's requests [33, 84].

Robustness - Robustness is the degree to which a Web service can function correctly even if given invalid, incomplete or conflicting inputs [33, 98].

Accuracy - Accuracy is the rate of errors produced by the service [33, 98].

Integrity - Integrity is the ability of the Web service to be protected from unauthorized access or modification of programs or data [33, 84, 98].

Reputation - Reputation is defined as how well a service was handled. The reputation of a Web service mainly depends on end user's experience of using the service and is considered a subjective property [21, 57, 79, 132].

Cost - Cost [20, 21, 59] is considered as the amount of resources (in terms of clockcycles, memory, or human resource) that the Web service consumes.

Interoperability - Interoperability represents the Web service's ability to be able to operate between different development environments [33].

Scalability - Scalability is the ability of increasing computing capacity in order to process more clients' requests [33].

Exception Handling - Exception handling is the ability for a Web service to handle exceptions [33].

Network-Related QoS - Network related QoS is the ability to work together with the transport network to minimize network delay, delay variation and packet loss [33].

Given the numerous number of different QoS properties, [20, 21, 57, 59] use a subset of QoS properties in their QoS model. They argue that similar approaches for their work that are applied to the subset of QoS properties can be extended to other QoS properties not considered in their work. In this thesis, we have selected 5 QoS properties: *price*, *availability*, *capacity*, *reliability* and *execution duration* to use in our QoS model.

The use of capacity [33, 98], price [57, 79, 132] and execution duration [14, 57, 59, 79] is straightforward in the sense that these QoS properties can be quantified. The authors consider these QoS properties as the number of requests, amount of currency and amount of time respectively. In our QoS model, we use *capacity*, *price* and *execution* similarly.

Regarding availability and reliability, we have taken the approach similar to [57, 59, 84] where availability and reliability is regarded as a probability.

In Section 3.2.2, we present the definitions of the QoS properties that we use in this thesis. We also elaborate on why we have chosen *capacity*, *price* and *execution duration* to be deterministic values while *availability* and *reliability* to be modelled as a probability.

3.2.2 Selected QoS Properties

In this section, we describe the QoS properties and their respective notations that we use in this thesis. The notations for each QoS property have to be done from both the perspective of the client's request as well as the service provider. For easier readability and understandability, we list the notations for the request in Section 3.2.3 while the notations for the service providers are presented later in Section 4.1.3. The QoS properties of the request and the service provider have to be the same in order for a proper comparison to be done.

In this thesis we chose 5 properties - *capacity*, *price*, *execution duration*, *availability* and *reliability* for our QoS model in this thesis. We elaborate the definitions here:

- Availability is defined as the probability [57] that the Web service is up and ready for consumption. We choose to define availability as a probability because it can be defined a period of time. For example, one method of calculating the availability could be to 'ping' the Web service every second to determine the percentage of ping replies were received.
- Reliability is defined as the probability [57] that the Web service returns a reply which the client expects. We choose to define reliability as a probability because it can be defined over a number of requests. For example, if there are 10 requests and 9 of them were replied with a reply which the client expects, then the reliability of the Web service is 90%.
- Capacity is defined as the number of requests which the Web service can handle at any point in time. We choose to define capacity as a deterministic value because by relating it with the number of requests, the capacity can be calculated easily.
- Execution duration is defined as the amount of time which the Web service takes to handle the request. The time is calculated from the point at which the request is received by the Web service to the point at which the reply

is sent to the client. In this thesis, we ignore networking complexities and assume that the communication between the Web service and the client is negligible.

- Price is defined as the amount of currency which the Web service changes for handling a request. Choosing it as a deterministic value means that it is easy to related to with real world terms. For example, we can define the price of a Web service to be \$400.

3.2.3 Request

In this section we define the QoS properties that define a request, as well as the notations used. It is the responsibility of the client to determine the value of the QoS properties. The properties, its definitions and the notations are listed in Table 3.1 for request r that is sent by client c at time t. The notation C^t represents the set of all clients that sent requests to the group of Web services at time t.

Table 3.1. List of Request Properties

Property	**Notation**	**Description**
Requested Availability	$reqAv^t_{c-r}$	Probability of *availability* requested
Requested Reliability	$reqRl^t_{c-r}$	Probability of *reliability* requested
Requested Execution Duration	$reqEx^t_{c-r}$	*Execution duration* requested
Requested Price	$reqPr^t_{c-r}$	Amount of currency client is willing to pay

We define $reqQoS^t_{c-r}$ to be the tuple containing $\{reqAv^t_{c-r}$, $reqRl^t_{c-r}$, $reqEx^t_{c-r}$, $reqPr^t_{c-r}\}$.

3.3 Service Level Agreement

The SLA acts as a contractual agreement between two parties. It states the terms and conditions that both parties have to fulfil. They usually involve one party requesting the service of another party in return for some form of financial payment. In the event that the service provider cannot fulfil the request after agreeing to the term and conditions, the party that is providing the service is usually faced with a penalty, also included in the SLA.

In the framework of a group of Web services, not only do we have to consider the SLA between the client and the group of Web services (client-side SLA), it is also necessary to consider the SLA between the group of Web services and the individual service providers (supplier-side SLA) [61]. We first present the background of the SLA in Section 3.3.1. Next, in Section 3.3.2 we describe some specific components of the SLA used in this thesis.

3.3.1 Background

One major reason for having an SLA is to ensure that the service provided are in agreement to the satisfaction of all parties [32] involved. Several authors have thus introduced an SLA framework [36, 61, 65, 66] in the domain of Web services.

There are two main components to a SLA framework, the first component involves the definition or negotiation of the SLA which decides which terms and conditions go into the SLA. The second component is with regards to SLA monitoring [105], ensuring that the terms and conditions specified in the SLA is carried out:

- Negotiation [36] refers to the negotiation component of the SLA framework as the Web service contracting component. The negotiation component defines the terms and conditions that go into the contractual agreement, known as the SLA itself, between the parties involved. The SLA contains [61, 65, 66]:
  1. The involved parties
  2. The validity period that defines the period of time covered by the SLA
  3. Service Level Objectives (SLO), such as the level of QoS

4. The method of computing the QoS properties
5. The method of measuring the QoS properties
6. The consequences if a violation of the SLA occurs (For example a financial penalty to the guilty party)
7. Amount of payment in the event that the service is provided to the satisfaction of all parties

- Monitoring ensures that the terms and conditions stated in the SLA are going as planned, and if a violation occurs, the corresponding consequences are implemented. This is similiar to the monitoring of the QoS. We discussed this in Section 2.2.1 in detail.

Although the SLA generally only concerns the service provider and the service client, [61] creates the distinction between two kinds of SLAs in a composition of Web services, supplier-side SLA and client-side SLA. In their paper, supplier-side SLA involves the service provider and the service broker, and client-side SLA involves the service client and the service broker. This is something that we adopt in this thesis, where supplier-side SLA refers to the SLA between the group of Web services and the service provider; client-side SLA refers to the SLA between the group of Web services and the client.

[105] also mentions the fact that SLAs can be between two (or more) service providers, however we do not use this approach in our thesis because we think that such an approach can be complicated. Our approach considers requests to be atomic and therefore we can make this simplification.

In [36], the authors consider the difference between resource metrics and composite metrics. They define resource metrics as QoS properties such as availability or response time that can be retrieved directly from the service provider or by intercepting client transactions. Composite metrics, on the other hand, are created by aggregating several resource (or other composite) metrics according to a specific algorithm. In our thesis, we are only concerned with resource metrics.

[100] introduce the idea of using soft contracts instead of hard contracts in the SLA. The authors define hard contracts to be QoS properties in contracts in the form of hard bounds. For example, response times less than a certain fixed value is considerd to be a hard bound. Soft contracts are a probability distribution for the considered QoS property. Although the authors focused only on response time, the concept can be extended to other properties such as *availability* and *reliability*. In this thesis we use both hard and soft contracts since *availability* and *reliability* are defined as a probability while *capacity*, *execution duration*, *cost* and *price* are defined as deterministic values.

While there are SLA frameworks involving service provider to service client [36, 65, 72, 129], to the best of our knowledge, there is no current literature proposing an SLA framework for specific use in a community of Web services.

3.3.2 Service Level Agreement Components

In this section, we elaborate on the components of the SLA that might occur in both client-side SLA and supplier-side SLA. Each SLA should include:

- The involved parties - The parties that are involved in this SLA should be named here. This involves either the identity of the client and the group of Web services for the client-side SLA or the identity of the group of Web services and the individual Web service. There should not be any ambiguity in the identity of both parties.
- Validity period - The SLA should last for as long as it takes for the request to be handled. For the client-side SLA, this refers to the period when the request was sent to the group until the client receives a reply. For the supplier-side SLA this would refer to the length of time which the Web service remains in the group of Web services.
- Service Level Objectives - For the client-side SLA, the minimum level of QoS that is expected should be stated clearly. This has to be done for each and every QoS property. This sets the standard to which the level of QoS

the group of Web services should aim to provide for the client. For the supplier-side SLA, the level of QoS which the service provider should be able to provide when a request is assigned to it.

- The method for computing the QoS properties - This deals more with syntax, for example, whether the execution duration is calculated per second or per milisecond.
- The method for measuring the QoS properties - How should each QoS property be measured? For example, regarding the measurement of the execution duration, does the timing start at the time when the client sends the request or when the group of Web services receives the request?
- The consequence if a violation of the SLA occurs - This usually involves some form of financial payment.
- Amount of payment in the event that the service is provided to the satisfaction of all parties - This usually involves some form of financial payment.

3.4 Auctions

Auctions allow the providers of goods and services to come to an agreement with consumers over the price of the goods and services that is being sold. We consider the goods and services that is being sold to be an item. Providers offer an item up for bid, and the consumers make a bid for the item. Consumers making a bid are known as 'bidders'. Depending on the auction system, the method of arriving to an agreement on the price differs. However, for a sale to occur, both the provider and the bidder have to agree on the price.

Auctions are typically conducted under the supervision of a neutral party, known as the auctioneer. The auctioneer picks the type of auction system, described in Section 3.4.1 to use to conduct the auction. This auction system is one element in determining the final bid price of the item.

Two other elements that affect the final bid price of the item include how bidders value the item described in Section 3.4.2 and how much risk the bidders are willing to take which we describe in Section 3.4.3.

Bidders on their own can form strategies in order to get a favourable price, we describe potential strategies that bidders can employ in Section 3.4.4. Finally, in Section 3.4.5, we identify properties in the auction systems that deter bidders from using these strategies.

3.4.1 4 Basic Auction Systems

In this Section we describe the four basic auction systems:

- English auction - Also known as the open-ascending auction is the most common type of auction. In the English auction, the auctioneer starts the bid at a low price. Bidders openly publicize the amount of their bid, with each subsequent bid higher than the previous bid. The auction ends when no bidder is willing to bid at a higher price than the currently highest price. The winning bidder is the bidder with the highest price and pays the price which he announced.
- Dutch auction - Also known as the open-descending auction. The Dutch auction starts with the auctioneer announcing an absurdly high price. The auctioneer lowers this bid price until a bidder is willing to accept the last announced price. The bidder pays the price which was last announced.
- First-price sealed bid auction - In this type of bidding, all bids are private and sent directly to the auctioneer. Each bidder does not know the bid of other bidders. The winner of the auction is the bidder that submitted the highest bid. The winner pays the price which he submitted.
- Second-price sealed bid - In this type of bidding, all bids are private and sent directly to the auctioneer. Each bidder does not know the bid of other bidders. The winner of the auction is the bidder that submitted the highest bid. However, the winner pays the price of the second highest bid, and not the bid which he submitted.

3.4.2 Item values

The bidder's value of the item can affect the eventual winning price. This is important for the auctioneer when choosing which auction system to use. In this section we describe the different possible value-types.

The three types of values an item in auction can have are:

- Private: In a private value auction, each bidder knows how much he values the item for sale and this value is considered private. The bidder's value of the item is unchanged upon knowing the valuation of other bidders.
- Common: In a common value auction, the information about the value of the item for sale is shared among bidders. But each bidder has different private information about what the value of the item actually is. An example would be treasury bills, where the value of the treasure bill is known to all parties, however, due to the fluctuation in interest rate and other financial commodities, investors may value the bill differently.
- Correlated: Correlated value auctions is neither private nor common. An example could be an artwork. Each bidder values the work of art differently, and thus have its own personal private value. However, the possibility of selling the artwork later indicates a common value among bidders.

3.4.3 Bidder Types

In this section we describe 3 different types of bidders. The three types of bidders revolve around the amount of risk that the bidder is willing to take. The bidding strategies can be different depending on what how much risk the bidder is willing to take. They are:

- Risk averse: A risk averse bidder is reluctant to take risks. The bidder prefers to take a lower payoff at a lower risk than a higher payoff at a higher risk.
- Risk seeking: A risk seeking bidder prefers to take a higher risk at a higher payoff.

- Risk neutral: A risk neutral bidder is indifferent to risk. He will neither pay to avoid it nor to take it. Risk does not affect his decisions.

In this paper, we are focused to finding the best auction system to be used in a group of Web services. Our focus is not on defining what kind of risks bidders are willing to take. Since most research is done assuming risk-neutrality, throughout this paper, we assume bidders to be risk neutral.

3.4.4 Auction Strategies

In general, the provider of the item taking part in an auction is looking for the highest possible price. Conversely, the consumer of the item hopes to get a bargain by acquiring the item at the lowest possible price.

This creates a game that providers and consumers can participate in. In economics this is known as Game Theory [122].There are other factors that can affect the final winning price of the item in this game. We assume that bidders are smart intelligent beings who can employ different strategies in order to achieve a lower final winning price. We discuss several of them here:

- Bidder collusion: Several bidders can coordinate their bid prices so that the bids stay artificially low. In this case, the bidders can get the item at a lower price than they would without colluding.
- Untruthful bidding: Bidders may not want to reveal their true valuation of the item. In certain times they may not even be sure of the valuation of the item themselves. Wanting to get the item at a lower price, the bidder may not bid as high as his own true valuation.
- Counterspeculation: If a bidder knows the bid of other bidders, he can make a more information decision on his own bid. In certain cases, it is possible for the bidder to pay to gather knowledge about other bidders.

The best strategy (highest payoff) for each bidder is known as the dominant strategy. We assume that bidders will always bid according to his own dominant strategy since it returns the highest payoff.

3.4.5 Auction Properties

In this section we describe the properties that auction systems can have in the context of Web services.

- Truthful dominant strategy (TDS): This property indicates whether each bidder will have the dominant strategy where he bids its own true value. This cuts down on any possible counterspeculation and forces the bidder to always tell the truth.
- Final Bid Value: This property compares the final bid value.
- Time-requirement: This property defines the amount of time required to conduct the bid. Sealed bid auctions, for example, would most likely end earlier than an English auction.

In the context of Web services, the time-requirements of the auction is critical because the nature of the Web service can be very time dependent. Keep in mind that from the user's perspective, the user sends requests to the group of Web services. Some requests, depending on the functionality of the Web services are time dependent and require immediate response. Examples of these can include the sending of an ambulance, or estimating the prices of stocks in a stock exchange. It is possible for the time taken to execute the auction to impact the significance of the QoS that the Web service can provide. Thus, we have chosen to include time-requirements as one of the properties to consider when comparing the different auction systems.

We mentioned in the previous section how bidders may choose to employ strategies for their benefit. Two of the strategies - counterspeculation and untruthful bidding can be avoided by choosing an auction system with the property of truthful dominant strategy. The third strategy, bidder collusion, can be avoided by having the bidders bid anonymously. A computerized system could be employed to prevent bidders from knowing the identities of other bidders. This system can be employed regardless of the auction system being used. In the context of Web services, this is easy to implement since the bidders are not expected to be in the same room, as well as the fact that the language and protocol can be chosen to fulfil this property.

We considered another property - the amount of bandwidth required to conduct the auction, however we decided not to include it in our model. The main reason is because in today's context of Web services, the amount of bandwidth required to conduct the auction is negligible.

3.4.6 Auction Comparison

In this section we consider the different properties auction systems can have if the bidders value the item differently (See Section 3.4.2) and different risks that they are willing to take (See Section 3.4.3).

The time requirement for the auctions are not affected by the way bidders value the item or whether or not the type of risks that they are willing to take. We see this in Table 3.2.

Table 3.2. Time Requirements for different auction systems

	English	**Dutch**	**First-Price Sealed**	**Second-Price Sealed**
Time-Requirement	High	High	Low	Low

We next examine the truthfulness and revenue properties of the four auction systems in Table 3.3.

These properties are used in our consideration of our solution on an auction system that can be used in a group of Web services, we present this later in Section 5.2.

3.4.7 Fixed Price Auctions

Extensive work on the properties of auctions with a fixed price has been done in [39] (known as Fixed-Revenue Reverse Auction). In [39], a seller wishing to raise a fixed price sells a quantity of a certain good. This fixed price is known

Table 3.3. Truthful and Revenues for different auction systems

	English	**Dutch**	**First-Price Sealed**	**Second-Price Sealed**
Private Value				
Truthful Dominant Strategy	Yes	No	No	Yes
Revenue [89]	Same	Same	Same	Same
Common Values				
Truthful Dominant Strategy	Yes	No	No	No
Revenue	Highest	Low	Low	High
Correlated Values				
Truthful Dominant Strategy	Yes	No	No	No
Revenue	Highest	Low	Low	High

and bidders bid on the amount which the bidders are willing to buy. The bidder who bids on the lowest quantity of good wins the bid. The results from [39] indicate that in such a system of fixed-price, the characteristics are similar to standard auction theory (i.e. English and Second-Price sealed auction have the TDS property). However, they did stress that sealed bid auction systems lead to lower quantities (similar to higher prices in a fixed quantity auction, a more favourable situation for the seller). Quantity equivalence also does not hold across the different auction systems when having a fixed-price. We summarize the findings in Table 3.4.

Table 3.4. Truthful and Final Bid Values for Different Fixed Revenue Auction Systems

	English	**Dutch**	**First-Price Sealed**	**Second-Price Sealed**
Truthful Dominant Strategy	Yes	No	No	Yes
Final Bid Value	Low	Low	High	High

An important finding in [39] indicate that the outcomes are not affected by the dimension (number of properties) of the auction. This means that a model with 3 QoS properties will have the same outcome as a model with 9 QoS properties. A similar conclusion was reached in multi-dimensional auctions [43]. This finding is important in our paper because the number of QoS properties in different models can vary greatly.

4

Solution

In this chapter we finally introduce the proposed solution to the main problem of sustaining the QoS of Web services. Our proposed solution involves a group of Web services, specifically using the framework of a *Community of Web services*. A community of Web services takes on the central coordination approach (See Section 2.1.3). The drawbacks mentioned are inherent in a community of Web services, however we mitigate these drawbacks by various proposed mechanisms presented later in Chapters 5 and 6.

We start with the basic framework of a community in Section 4.1. Next in Section 4.2 we present the Contract Net Protocol (CNP) which is used in a community of Web services. The introduction of the CNP is critical in understanding how auction theory can be introduced into the framework of a community of Web services. We then extend the CNP by including auction theory in Section 4.3. Next, we introduce a fixed-price auction system in a community in Section 4.4. Finally, we present the framework for forming the supplier-side SLA in Section 4.5.

4.1 Framework - Community of Web services

Benatallah et al. [8] define a community as a collection of Web services with a common functionality, although these Web services have distinct non-functional properties like different providers and different QoS properties. In this work, we consider a community as a means for providing a common description of a desired functionality (e.g., FlightBooking) without explicitly referring to any concrete Web service (e.g., EKFlightBooking) that will implement this functionality at run-time [9]. A framework to engineer communities can be found in [82].

We present the architecture of communities of Web services in Figure 4.1. In this architecture, we demonstrate how Web services, Universal Description Discovery and Integration (UDDI) registries, and communities interact with each other.

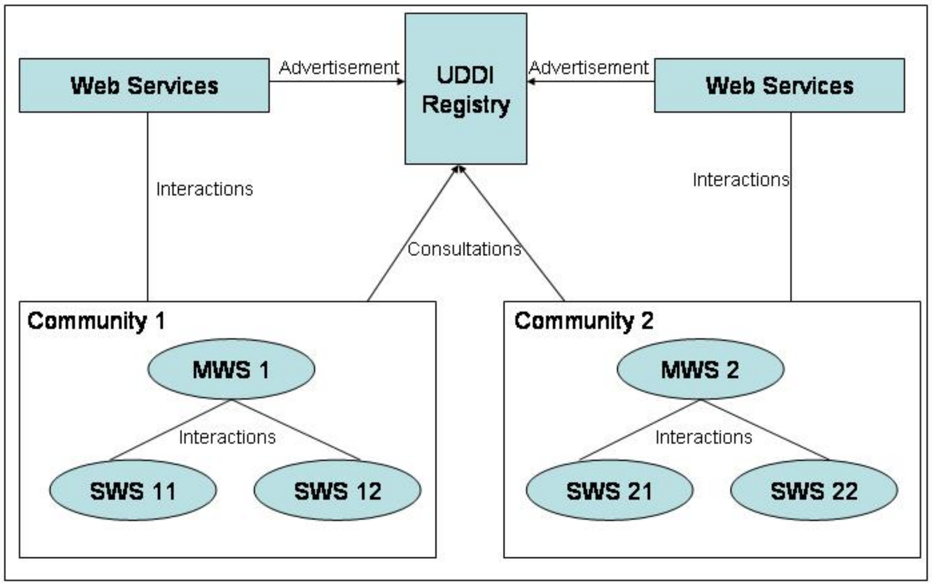


Fig. 4.1. Community of Web Services

In Figure 4.1, two communities are shown. Each community could offer different functionalities and they are considered dynamic by nature. This means that there are components in a community that may enter or leave the community. Specifically, Web services may enter or leave the community subject to certain conditions.

Individual Web services advertise their services on UDDI registries. This creates an awareness for both communities looking for additional Web services, but also for other clients who are interested in individual Web services. The UDDI registry thus provides the community with the ability to consult its database on which Web service might be available to invite into the community. If the community decides to invite a particular Web service, it is up to the community to continue its interaction with the Web service directly.

As shown in Figure 4.1, there are two components in a community of Web services. The first component is a Web service that leads the community, this is known as the Master Web service (MWS). The second component are the Web services that handle the requests. They are known as *slaves*, or *Slave Web Services* (SWS). In a community of Web services, SWSs offer the same functionality but with possibly different levels of QoS.

Our definition of a community differs from [82] in the sense that their approach allow for SWSs to interact with each other. They proposed the possibility of alliances, or micro-communities to exist within a community in order to substitute each other as a method of exception handling. In our approach, we have designated the MWS to be responsible for such a substitution. It is the responsibility of the MWS to ensure that the substitution is carried out successfully in order to sustain the QoS provided to the client. By allowing the MWS to perform the substitution, it allows for a more optimal substitution since the MWS is able to select a substitute SWS from the entire community.

The framework of a community of Web services can be built on top of the SOA framework mentioned in Section 3.1. This means that instead of an individual Web service publishing its service description on a service registry, a community can publish a service description on the service registry. When the service client asks for a list of service descriptions from the service registry, it can receive a list of service descriptions consisting of both individual Web services and communities of Web services. Thereafter, if the service client decides to use a community, it binds directly with the community. The binding between service client and service provider does not occur since the community represents the individual SWSs within it. In Figure 4.2, we show how a community of Web services can fit into the SOA framework.

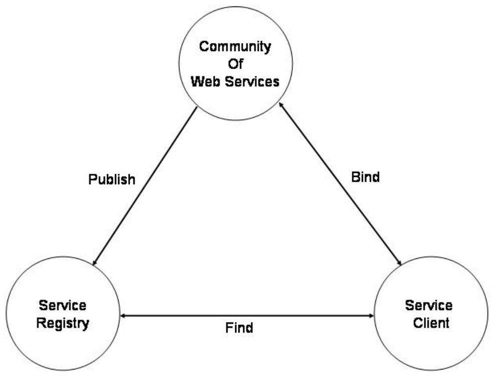


Fig. 4.2. Components in the SOA with a community

In Section 4.1.1, we describe the responsibilities of the MWS and in Section 4.1.2, we describe the responsibilities of the SWS. In addition, we include the notations for the SWS QoS properties in Section 4.1.3.

4.1.1 Master of Web services

We define the MWS as a Web service that is designated at design-time to lead and manage a community of Web services. Although the SWSs within a community can change, the identity of the MWS does not change over time.

We list the responsibility of the MWS here:

- to accept or invite new Web service providers to be members of the community. The MWS does this by consulting the UDDI registry for suitable Web services. This consultation can be done on a frequent basis in order for the MWS to get updated information on the available Web services that might give additional value to the community. There are two important criteria which the MWS needs to consider to determine which service providers to admit. The first criteria is that the service provider needs to have the same functionality as the one designated by the community. This ensures the homogeneity within the community, and allows the MWS to allocate requests directly without doubt as to whether the service provider can functionally handle the request. The second criteria is that the service provider meets the level of QoS that is required by the MWS. A service provider with a low QoS can be assumed to handle the request at a low QoS, and this can affect the satisfaction of the client. A low client satisfaction can then lead to less requests being sent to the community. Other than consulting the UDDI registry for suitable Web services, it is also possible for individual service providers to apply directly to the community. Regardless of how the community discovers the service provider, both criteria should be considered by the MWS to determine whether or not the admit the service providers into the community.
- to expel existing SWSs. A service provider may change its functionality over time, this could be due to technical difficulties or system designers deciding that another functionality would be more desired. In such cases where the functionality of the service provider is changed, it is straightforward for the MWS to decide to expel the service provider from the community. The MWS would do this in order to ensure that the SWSs within the community have the same functionality. A more complicated situation may arise if a service provider changes its QoS over time, or if it is only able to provide a different level of QoS than initially promised. If the provided QoS is not what the MWS was expecting from the MWS,

the MWS would need to decide at what threshold should it expel the SWS from the community. The expulsion of a SWS would have to be done in accordance with the SLA (See Section 3.3) that both SWS and MWS agreed upon when the SWS initially joined the community.

- to search for additional sources of requests. In general, we assume that the higher the number of requests that are successfully handled by the community, the better off it is for the community. Therefore, in order to achieve this, it is the responsible for the MWS to search for additional sources of requests. In addition to having potentially more requests, additional sources could lead to these sources offering more budget for each request. The MWS can do this by one of two methods, it can either advertise itself on the UDDI registries or individually contacting new clients regarding the types of services and the level of services that it can provide. In addition, the possibility of asking for references from existing clients is also present, however, the implementation of this is beyond the scope of this thesis.
- to select the SWS that will handle the client request. This is done when the MWS receives a request from the client on behalf of the community. Since the functionalities of the SWSs within the community are the same, the MWS can consider that any of the SWSs within the community is functionally capable of handling the request. The main consideration is thus the level of QoS that can be provided to the client. It is also important for the MWS to consider the budget and requesting level of QoS that is asked for by the client. Once this selection is made, the MWS has to notify the SWS that it has to handle the request, and then send the request accordingly to the selected SWS.
- to execute the substitution process. The substitution process is important in order to sustain the QoS of Web services. In the event that the originally selected SWS is not able to handle the request, either at the level of QoS that was desired, or at all (in the event of complete failure), the MWS

would need to begin the substitution process. The goal of the substitution is to be able to select another SWS to handle the request at a satisfactorily level. It is also the responsibility of the MWS to determine when such a substitution might be needed. The constant monitoring of the QoSs of the SWSs that are handling a request is also thus necessary. Once the substitution is considered necessary, the MWS then needs to select among the remaining SWSs which SWS to handle the request, inform the selected substitution SWS and then send the request to this SWS. To complicate things further, the MWS needs to consider multiple failures, and thus multiple substitutions.

- monitoring the functionality and the level QoS of the SWSs in the community. The community consists of service providers that have the same functionality, and the MWS is responsible for the monitoring of their functionality. This is to ensure that when a request is send to the SWS, it can satisfy the functional requirements of the request. In addition, the level of QoS that can be provided by the SWS is also important, in order for the MWS to have an idea of what to expect from the SWS. There might be possible expulsions in the event that the functionality or level of provided QoS is not what the MWS expects.
- storing of information regarding the SWSs. As the MWS collects information about the SWSs in the community, the MWS may want to store this information in order to better monitor the level of QoS of the SWSs. The more information the MWS can store about the SWS, the better the prediction it can make on the level of QoS that the SWS can provide in the future.
- check the liveness of the SWSs within the community [82]. In addition to monitoring the SWSs, a more frequent ping function could be used to detect any possible failures of SWSs within the community. This can ensure a faster substitution or a more accurate update of the level of QoS that the community can provide.

- to submit the reply back to the client. Once a SWS has handled the request, the SWS sends the reply back to the MWS. It is then the responsibility of the MWS to send this reply to the client. One important feature of having a group of Web services is that the client is unaware of which Web service handled the request. In addition, this allows multiple failures to occur without the client knowing.

We consider the MWS to be a dedicated Web service that is in charge of the management of the community, the MWS does not directly take part in the handling of any request, but instead sends the request to the SWSs within the community and replies to the client on behalf of the SWS. The selection of the MWS is a conceptual one. One problem that a community has is that if the MWS were to fail, the community would fail to function. In practice, a SWS may be promoted to become the MWS of a community or a backup MWS could have been implemented just in case. However, the implementation of these solutions are beyond the scope of this paper.

In addition, we assume the MWS to be fair in the selection of SWSs. This means that there is no additional biasness the MWS over certain SWSs in the community or Web services that have yet to join the community.

4.1.2 Slave Web services

SWSs have the following responsibilities:

- to assess whether it is able to handle a request and reply accordingly when the MWS asks for bids using the Contract Net Protocol (CNP). We elaborate later in Section 4.2 the CNP that is used in a community of Web services. As part of the CNP, the SWS within a community has additional responsibilities which we elaborate later.
- to handle requests. The SWS handles the request that it receives and submit the reply to the MWS.

- to retire from the community. The SWS should be allowed to leave the community in accordance to the terms and conditions that were agreed upon in the SLA (See Section 3.3). This may result in certain penalties that the SWS would have to make to the community. However, the SWS may want to retire from the community due to several reasons, they could be internal reasons such as its change in functional or non-functional properties, technical problems, or external reasons such as not receiving enough requests from the community.
- to reply to the ping function from the MWS. This ping function tells the MWS that the SWS is still alive.

4.1.3 Slave Web Service Quality of Service properties

In this section we define the QoS properties that are associated with an SWS. We define J^t to be the set of SWSs in a community at time t, the number of SWSs in a community is thus defined as $|J^t|$. The properties, its definitions and the notations are listed in Table 4.1 for SWS, SWS_j at time t, where $j \in J$.

Table 4.1. List of SWS Properties

Property	**Notation**	**Description**
Availability	$swsAv_j^t$	Probability that SWS_j is up and ready for consumption at time t
Reliability	$swsRl_j^t$	Probability that SWS_j handles the requests reliably at time t
Capacity	$swsCp_j^t$	The total number of requests that SWS_j can handle at time t
Execution Duration	$swsEx_j^t$	Amount of time which SWS_j takes to handle any request at time t
Price	$swsPr_j^t$	Amount of currency which the SWS_j charges at time t

We define $swsQoS_j^t$ to be the tuple containing $\{swsAv_j^t$, $swsRl_j^t$, $swsCp_j^t$, $swsEx_j^t$, $swsPr_j^t\}$.

In addition, we consider the current capacity, $swsCurCp_j^t$, of SWS to be the number of requests that SWS_j is handling at time t.

4.2 Contract Net Protocol

Within the community of Web services, the Contract Net Protocol (CNP) [117] takes place which allows the MWS to make a better selection of which SWS would be handling the request. We first present the most basic type of the CNP where the MWS only asks the SWSs whether they are able to handle the request. The 6 steps of the CNP can be found in Figure 4.3.

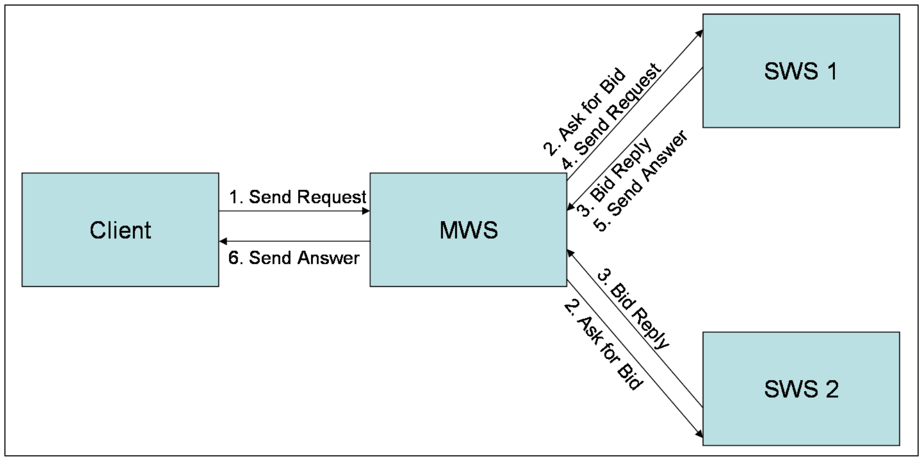


Fig. 4.3. Contract Net Protocol

The 6 steps for the CNP are described as follows:

1. Client sends the request to the MWS. We have earlier provided the QoS properties that will be used in our thesis (See Section 3.2) as well as its associated notations. The values of these QoS properties are used in the request that the client sends to the request.

2. MWS asks the SWSs for bids. Using the values of the QoS that was received, the MWS can forward the request to each SWS. The purpose of this step is to allow the SWS to consider whether or not they can handle the request.
3. The SWSs make their bid to the MWS. The decision on whether or not the SWS can handle the request is made by the SWS and subsequently sent back to the MWS. We take note that only positive replies are sent from the SWS to the MWS. In the event that the SWS concludes that it cannot handle the request, a non-reply is sufficient enough for the MWS to know that the SWS cannot handle the request.
4. The MWS makes its selection on which SWS to handle the request. The MWS makes this selection based on the replies that it received in the previous step. Only SWSs with positive replies will be considered by the MWS. The criteria which the MWS use to select can vary from community to community. It may not always be the situation that the MWS would select the Web service that can provide the best service to handle the request. We elaborate on the selection process later in Chapter 5.
5. The SWS handles the request and returns the answer to the MWS. In our proposed approach of using the community of Web services, the SWS does not have direct contact with the client. We can consider that the SWS is not aware of the identity of the client. Therefore, the MWS has to forward the reply from the SWS to the client.
6. The answer is sent from the MWS to the client. The client receives the reply from the MWS and the protocol is done.

This selection process is also used when the MWS needs to find a replacement SWS. A replacement SWS is needed when a SWS which is asked to handle a request fails.

Reasons for using the CNP include:

- The CNP fits well within the framework of a community of Web services. Since the framework insists that all requests and replies go through the MWS.
- The CNP gives the responsibility of selecting which SWS to handle the request to the MWS. Since the MWS has a global view of the SWSs within the community, we argue that this allows the MWS to give the most optimal selection within the community.
- In addition to the selection, we also consider the substitution process where the MWS needs to consider the situation where the originally selected SWS were to become unable to handle the request, either entirely, or at the level which was expected of it. We can use the CNP again for the substitution process by resending the request to the SWSs within the community. Since the MWS has full observability of the SWSs within the community, the substitution SWS can be considered to be the most optimal SWS.

We have described the most basic version of the CNP. In our approach, we extend the basic CNP to use different auctioning techniques mentioned in Section 2.1.6 such as the English Auction or the Dutch Auction. If an auctioning method is applied, the SWSs within the community bid on the request as the request comes in. The MWS's selection process can then also take price into the consideration. We extend this in the next section.

4.3 Auction Theory in a Community

In this section we elaborate how auction theory can be incorporated into a community by improving the CNP. They both share similar elements such as two way communication which allow auction theory to be implemented. In addition, since the community has a leader in the form of the MWS, it forms the prime candidate to be the auctioneer responsible for conducting the auction.

One approach that the MWS can take is by only providing all QoS properties provided by the client except the price ($reqPr^{t}_{c-r}$)to the SWSs. Any other particular QoS property could be omitted from the tuple, however from the perspective of revenue of the community, the MWS would benefit the most by omitting the price. For example, since the MWS acts as a broker between the client and the SWSs, a dishonest MWS would receive payment from the client and could subsequently pay the lowest priced SWS for handling the service. This results in the biggest profit for the community.

Another consequence of such an approach is that this forces each SWS to make a decision on how low a price each SWS can provide based on the other QoS properties provided. This replaces step 3 in Figure 4.3. The benefits for such an approach include:

1. The MWS is able to make a more informed decision on which SWS to handle the request. For example, it may choose the SWS that provides the lowest price for the benefit of the client or the community.
2. In terms of QoS monitoring, the MWS is able to track the levels of price of each SWS more accurately. By receiving the exact price which the SWS will handle the request for, the MWS is able to monitor the levels of QoS of each SWS based on the level of QoS of the request.

This approach allows the implementation of an auction system within a community of Web services. The MWS can act as the auction broker with the SWSs acting as bidders. For example, an English auction could be conducted by the MWS where each SWS openly bids the price which it can handle the service for. Other SWSs receives these bids and subsequently outbids the winning SWS by offering a lower price. The auction ends when no other SWSs is willing to outbid the current price. The MWS thus allocates the request to the winning SWS at the price which it bid on.

In the case of communities, we observe the item values are of type private (see Section 3.4). This is because each SWS has different emphasis on different

QoS properties. Some SWSs may find it easier to offer a higher availability but limit the amount of reliability that it can offer. We have also assumed here that the bidders are all risk neutral. In terms of auction properties (see Section 3.4.5), we notice that the following properties comparing the 4 auction systems in Table 4.2.

Table 4.2. Truthful, Revenue and Time-requirement properties for Fixed-QoS Auctions in Communities

	English	**Dutch**	**First-Price Sealed**	**Second-Price Sealed**
Private Value				
Truthful Dominant Strategy	Yes	No	No	Yes
Revenue [89]	Same	Same	Same	Same
Time-Requirement	High	High	Low	Low

It is clear from Table 4.2 that the best type of auction for the MWS to conduct is the second-priced sealed auction. This type of auction has a truthful dominant strategy for the bidders (SWSs) and requires the lowest amount of time to conduct. Implementing this into the CNP, it replaces step 3, we now have SWS_j at time t to reply:

- Yes, the SWS can handle the request at $swsPr_j^t$
- No, the SWS cannot handle the request

The MWS then selects the SWS which replies the lowest $swsPr_j^t$ to handle the request, however, the winning SWS only charges the second lowest price to handle the service. This is done in order to maintain a truthful dominant strategy, i.e., all the SWSs will make bids according to their true valuation.

This auctioning technique can be further improved by allowing fixed price auctions to take place, we describe this in the next section (Section 4.4).

4.4 Fixed Price Auctions in a Community

In the previous section, we described how an auctioning technique could be implemented in a community of Web services. The previous auctioning technique involves having the MWS send all QoS properties provided by the client except the price($reqPr^t_{c-r}$) to the SWSs. In return each SWS sends a bid reply with a price which the SWS is willing to handle the request at. In this section we further improve the technique by incorporating a fixed-price auction.

Instead of sending all QoS properties provided by the client except with the price to the SWSs, our solution is to have the MWS send only the price to each SWS. In return, the SWSs will send a bid reply containing a tuple with the values of all QoS properties (minus the price) which each SWS can handle the request in return for the specified price. We call this a fixed price auction.

We propose that instead of sending the requested price that was received from the client, the MWS sends a new bid-price to the SWSs. The difference between the requested price and the bid price will be absorbed as a form of revenue by the community. This approach gives more flexibility as well as a possible source of income and incentive for the MWS to lead the community. In this case the bid price will always be lower than or equal to the requested price. The requested price is hidden from the SWSs.

The MWS can customize the bid price based on the SWS that it is sending the price too, this allows the MWS to determine a different price for each SWS. The MWS might want to set different prices depending on how well the SWS is able to provide its promised level of QoS as well as its bid history. The minimum bid price that a MWS can send to each SWS would be 0, in other words, what level of QoS would the SWS be willing to do for free.

The fixed price auction does not change the fact that multiple SWSs are competing against each other to provide a certain service to the user. There are four properties to this approach:

- The fixed price auction handles the situation where users have a fixed budget and are looking for the best service that they can get.
- SWSs are allowed to submit bids according to their strengths. The approach allows the SWS to make its own assessment on how good a service it can provide and submit this bid to the MWS.
- One challenge which the community faces when it comes to managing SWSs is the estimation of the QoS of each SWS. This QoS can change over time due to equipment failure, power shortages, software errors, etc. This approach allows the community to assess the level of QoS that each SWS can provide. The community will still have to monitor the eventual level of QoS that the selected SWS provides and compares that with the bid of the same SWS. With more bids, the community is able to get a better idea of the level of QoS the SWS can provide, based on the prices that users provide.
- The value of the bid-price relative to the requested price influences the level of QoS that the SWSs provide. The MWS has to keep this in mind when determining the difference between the bid-price and the requested price.

In Section 3.4.7 we presented the properties of such a fixed price auction. From the properties in Table 3.4 we see that the second-price sealed auction has the highest final bid value as well as a truthful dominant strategy. Our approach is to thus have the MWS conduct a fixed-price second-QoS sealed auction. The sealed auction also has the same amount of time-requirement as the CNP. The full mechanism (replaces the CNP) is as follows:

1. Client sends the request, $reqQoS^{t}_{c-r}$ (See Section 3.2.3), to the MWS
2. MWS sends the individual bid price, $bidSelPr^{t}_{j-c-r}$, to each SWS
3. The SWSs reply with a tuple containing the values of the QoS properties that it can handle the request at the stated bid price. Since these

values are dependant on the bid price, we define the QoS properties as $(SWSAv^t_{j-reqPr_{j-c-r}}, SWSRl^t_{j-reqPr_{j-c-r}}, SWSEx^t_{j-reqPr_{j-c-r}})$

4. The MWS makes its selection on which SWS to handle the request
5. The service provider handles the request and returns the answer to the MWS
6. The answer is sent from the MWS to the client
7. Payments are made by having the client send the requested price, $reqPr^t_{c-r}$, to the MWS, and the MWS making payment to the SWS by sending the bid-price, $bidSelPr^t_{j-c-r}$, to the SWS that handled the request.

The second-price sealed auction can be conducted in a similar manner for substitution as well, however in this case, the MWS sends the substitution bid price $bidSubPr^t_{j-c-r}$ instead to SWS j at time t.

4.5 Supplier-side SLA Framework

A community can be formed by the MWS inviting Web services into the community. In order to formally define the relationship between the MWS and SWSs, a supplier-side SLA is needed. We consider the SLA to be generated by the MWS. Such an SLA also acts as a specific model of a community that is used in this thesis. In Figure 4.4, we show where the client-side SLA and supplier-side SLA apply in a community.

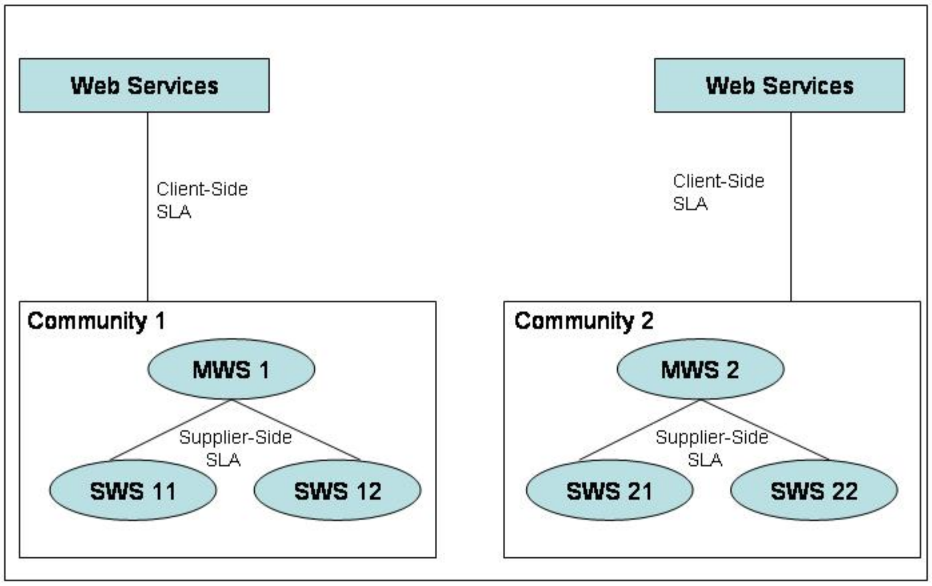


Fig. 4.4. SLA for a Community of Web Services

The supplier-side SLA states the terms and conditions which both parties have to fulfil, as well as any payment issues. Before a Web service can join the community, the potential Web service will have to agree to these terms and conditions. Likewise, the MWS has to ensure that it follows to the terms and conditions in the SLA. In the situation where either party violates these terms and conditions, the penalty which the guilty party incurs should also be defined in the SLA. In this section we propose the framework of the supplier-side SLA between the community and Web service. Once a Web services is admitted into the community, it can then be formally called a SWS of the community. Our proposed framework resembles the framework in [66], however our framework is specific to a community of Web services. The framework of forming the SLA between Web service and community can be performed in 5 steps:

- Step 1: Negotiation (Section 4.5.1)
- Step 2: Signing and Setup (Section 4.5.2)
- Step 3: Monitoring (Section 4.5.3)
- Step 4: Penalties (Section 4.5.4)
- Step 5: Termination (Section 4.5.5)

Finally in Section 4.5.6 we summarize and conclude the supplier-side SLA framework.

4.5.1 Step 1: Negotiation

In the negotiation process, the community presents the Web service with the supplier-side SLA and the Web service negotiates the terms and conditions in the SLA. Since the community consists of similar Web services, the community has to ensure in this step that the potential Web service can fulfil the same functional requirements.

The following list is a proposed list of what might be included in the SLA between the community and Web services:

- Identity of the community and Web service - In the case of the community, it is represented by the MWS. The MWS and Web services would be technically identified by the IP address. However, the identity could also include the company name and its legal representation that is in charge of running the service.
- Functional requirements of the Web service - What kind of service can the Web service provide.
- QoS properties of the Web service - The level of QoS can the Web service can provide when handling a request.
- Workload of the Web service - The community can guarantee a minimum number of requests that the Web service receives. This is done to entice

Web services to join the community, we denote this as gWl_j^t and is defined as a proportion of requests that the Web service is handling to the capacity of the Web service, thus having a value of 0 to 1.

- Monitoring - The method of calculating the values of the QoS properties of the Web service. This includes the technical syntax, the method and frequency of monitoring.
- Privacy Issues - The MWS may want to keep records of the level of QoS that the SWS has provided. Some SWSs may feel that such records is a violation of privacy, and thus should negotiate with the MWS on the type of information that are stored as well as the length of period which the information is stored.
- Method of Payment - How should any kind of payment be made between community and Web service, this includes specifics such as the currency.
- Length of service - Period of time which the Web service remains in the community.
- Penalties - Penalties either party receives in case of a breach of agreement.
- Termination - The proper method for either party to go about terminating the agreement with each other.

4.5.2 Step 2: Signing and Setup

When the terms and conditions in step 1 have been agreed upon by both parties, they can proceed with the formal signing of the SLA. This can occur with a third-party Web service which oversees SLAs.

The setup phase includes any kinds of tests that might be needed as well as ironing out the technical syntax and specifications. This can include the deployment of monitoring software on the Web service.

4.5.3 Step 3: Monitoring

The monitoring step of the SLA is a real-time component. In this step, the community monitors each SWS to determine if the terms and conditions in the SLA are followed. This could include whether the SWS has been handling requests at the promised level of QoS. Conversely, the SWS monitors whether the community does what is agreed upon in the SLA, for example, whether the community is giving enough requests to the SWS.

4.5.4 Step 4: Penalties

Penalties may be required if a SWS cannot handle the requests up to the predetermined level of QoS. The community detects this through the monitoring step. The supplier-side SLA should include the exact penalty and method of payment.

4.5.5 Step 5: Termination

The termination can occur at the end of the length of service that was determined previously. The termination may also incur by either party before the full length of service. If termination of the SLA occurs before the full length of service, it may be necessary for one party to compensate the other party. This can be a form of penalty, therefore, the SLA should include the exact penalty and method of payment.

4.5.6 Summary

The supplier-side SLA framework sets the basic requirements for creating a SLA between community and SWSs in a community of Web services. It also elaborates on the model of a community of Web services that we are using in this thesis. Additional terms and conditions may need to be set according to the mechanisms that the community chooses to implement. Later in Sections 5.2.5 and 5.3.2, we elaborate on the specific changes that are required to the SLA due to proposed mechanisms.

5 User Request Management

In this chapter we introduce our approach in handling user requests within the framework of a community of Web services in order to sustain the QoS of Web services. Our approach takes into account the satisfaction of the three main parties [77] - service client, SWS and MWS. We call this the 3-way satisfaction approach, and it can be applied for both the selection and substitution processes. In addition, auction theory is implemented in our approach for both the selection and substitution processes. We have earlier described how this can be done in a community of Web services in Section 4.4.

We first introduce the 3-way satisfaction approach in Section 5.1. Next we see how it can be applied to the selection process in Section 5.2. We discuss the substitution process in Section 5.3. In order to better illustrate the rationale for the introduction of auction theory into the user request management section, we present experiments using real world values in Section 5.4.

5.1 3-way Satisfaction

The 3-way satisfaction approach takes into consideration the satisfaction of all 3 parties:

- Service Client - the service client can always look for other Web services that are able to handle its requests if they are not handled up to satisfaction. In this case a service client's satisfaction is based on the level of QoS that the request was handled. Furthermore, if the client is satisfied, it may direct more requests to the community [68], and conversely so.

- SWS - the SWS joins a community expecting a certain level of work. A minimum level of work or minimum number of requests might be stated in the SLA between the SWS and community. The more requests it handles, the more possible income the SWS earns. If the SWS is satisfied with the amount of work that it is given, it may possibly increase the number of requests that it can handle by commiting more resources towards the community, and conversely so.
- MWS - the MWS is concerned with the revenue generated of the entire community. The MWS ultimately decides the allocation of the requests to the SWSs and is concerned with making the most amount of revenue. The revenue of the community can be generated by sending a higher price to the SWSs and having the MWS keep the difference. However, too high a difference and this may impact the level of QoS that is provided to the client. The more revenue the community makes, the higher the MWS satisfaction. This satisfaction is a reflection of how well the MWS is at managing the client requests and SWSs within the community. This satisfaction is also an indication of how well the MWS is doing with respect to other competing communities.

The MWS is responsible for tracking the satisfaction levels of the 3 parties. This is a normal extension from having the responsibility of allocating requests as they are directed to the community.

5.2 User Request Selection Process

In our approach, the MWS selects a SWS to handle the request when a request is submitted to the community. This selection is done in step 4 of the fixed priced auction presented in Section 4.4. The selected SWS is the SWS that provides the highest selection score. The selection score is determined from the client selection score (Section 5.2.1), the SWS selection score (Section 5.2.2), and the MWS selection score (Section 5.2.3). We present how to compute the selection score in Section 5.2.4. In addition, we analyse the implications of our approach to the SLA in Section 5.2.5.

5.2.1 Client Selection Score

The client selection score is focused on the perspective on the client and is focused on the level of QoS that is provided to the client. In order to determine the client selection score for SWS j for request r from client c at time t, we consider the selection scores for the availability, the reliability and the execution duration. They are defined in Equation 5.1, Equation 5.2, and Equation 5.3 respectively.

$$selScoreAv^t_{j-c-r} = \begin{cases} 1 & \text{if } reqAv^t_{c-r} \leq SWSAv^t_{j-reqPr_{c-r}} \\ \frac{SWSAv^t_{j-reqPr_{c-r}}}{reqAv^t_{c-r}} & \text{if } reqAv^t_{c-r} > SWSAv^t_{j-reqPr_{c-r}} \end{cases} \tag{5.1}$$

$$selScoreRl^t_{j-c-r} = \begin{cases} 1 & \text{if } reqRl^t_{c-r} \leq SWSRl^t_{j-reqPr_{c-r}} \\ \frac{SWSRl^t_{j-reqPr_{c-r}}}{reqRl^t_{c-r}} & \text{if } reqRl^t_{c-r} > SWSRl^t_{j-reqPr_{c-r}} \end{cases} \tag{5.2}$$

$$selScoreEd^t_{j-c-r} = \begin{cases} 1 & \text{if } reqEd^t_{c-r} > SWSEd^t_{j-reqPr_{c-r}} \\ \frac{reqEd^t_{c-r}}{SWSEd^t_{j-reqPr_{c-r}}} & \text{if } reqEd^t_{c-r} \leq SWSEd^t_{j-reqPr_{c-r}} \end{cases} \tag{5.3}$$

Finally we can define the client selection score in Equation 5.4.

$$\begin{aligned} selClientScore^t_{j-c-r} &= w_{selScAv} \cdot selScoreAv^t_{j-c-r} \\ &+ w_{selScRl} \cdot selScoreRl^t_{j-c-r} \\ &+ w_{selScEd} \cdot selScoreEd^t_{j-c-r} \end{aligned} \tag{5.4}$$

Where $w_{selScAv}$, $w_{selScRl}$, and $w_{selScEd}$ are the weights for the selection score for availability, reliability and execution duration respectively.

5.2.2 SWS Selection Score

The SWS selection score is focused on the perspective of the SWS and is determined by the workload of the SWS. In order to determine the SWS selection score, $selSWSScore_j^t$, for SWS j at time t, we consider workload of the SWS if the request would be allocated to SWS_j. We also consider the minimum guaranteed workload by the community, gWl_j^t (See Section 4.5.1). We define the SWS selection score in Equation 5.5.

$$selSWSScore_j^t = \begin{cases} \frac{swsCurCp_j^t+1}{swsCp_j^t} & \text{if } \frac{swsCurCp_j^t+1}{swsCp_j^t} < (gWl_j^t) \\ 1 & \text{if } \frac{swsCurCp_j^t+1}{swsCp_j^t} > (gWl_j^t) \\ 0 & \text{if } swsCurCp_j^t = swsCp_j^t \end{cases} \tag{5.5}$$

5.2.3 MWS Selection Score

The MWS selection score is focused on the perspective of the MWS and its objective is to maximize the revenue generated. The MWS selection score depends on the revenue that the MWS might receive by selecting the SWS to handle the request. In order to determine the MWS selection score for SWS j for request r from client c at time t, we consider the requested price, $reqPr_{c-r}^t$, and the selection bid price, $bidSelPr_{j-c-r}^t$. We define the MWS selection score in Equation 5.6.

$$selMWSScore_{j-c-r}^t = \begin{cases} 1 - \frac{reqPr_{c-r}^t - bidSelPr_{j-c-r}^t}{reqPr_{c-r}^t} & \text{if } (bidSelPr_{j-c-r}^t) < (reqPr_{c-r}^t) \\ 0 & \text{if } (bidSelPr_{j-c-r}^t) \geq (reqPr_{c-r}^t) \end{cases} \tag{5.6}$$

5.2.4 Selection Score

Having provided the formulas to determine the selection scores for the client, the SWS and the MWS, we can now present the selection score. The selection score for SWS j for request r sent by client c at time t is defined in Equation 5.7.

$$\begin{aligned} selScore^{t}_{j-c-r} &= w_{selClientScore} \cdot selClientScore^{t}_{j-c-r} \\ &+ w_{selSWSScore} \cdot selSWSScore^{t}_{j} \\ &+ w_{selMWSScore} \cdot selMWSScore^{t}_{j-c-r} \end{aligned} \tag{5.7}$$

Where $w_{selClientScore}$, $w_{selSWSScore}$, and w_{selMWS} are the weights for the selection score for client, SWS and MWS respectively. These 3 weights sum up to 1.

5.2.5 SLA Implications

The implementation of auction theory in the community does not affect the community's relationship with the client. The client-side SLA between the community and client is not affected. Our main consideration in this section is the supplier-side SLA between the MWS and the SWS.

The implementation of auction theory into the community changes the relationship between the MWS and the SWS, specifically, the following items have to be discussed prior to such an implementation:

- The MWS has to make it clear to the SWS that an auction process is used in the selection of the SWS to handle the request. The type of auction process has to be clarified to the SWS as well as the expectations of the SWS in the event that it successfully wins the bid. For example, in the fixed-price second-QoS sealed auction, the MWS has to clearly state that the expected level of QoS is the second highest QoS instead of the level of QoS which the winning SWS bid on.

- The fixed-price second-QoS sealed auction implies that each SWS truthfully submits the level of QoS that it can handle for the fixed-price. The MWS may want to use this information in order to better monitor the level of QoS that it provides to the community. The supplier-side SLA may want to explicitly state this in order to avoid any privacy issues that the SWS might have.
- The risk of bidder collusion [122] is always present when an auction takes place. In the context of a community of Web services, it can usually be safe to assume that the SWSs do not know the identities of one another. However, the MWS may still consider to include penalties in case of collusion among SWSs. Such penalties have to be stated clearly in the supplier-side SLA.
- During the bidding process the SWS declares the level of QoS which it can service the request. This is specific to the request, and the supplier-side SLA would need to include the penalties for not meeting the declared level.

These considerations are a direct consequence of the implementation of auction theory into a community of Web services. Whether the MWS decides to switch to the auction system or whether it is inviting a new SWS into the community, these items have to be made clear to the SWSs.

5.3 User Request Substitution Process

In our approach, the MWS selects a SWS to perform a substitution when the MWS determines that a originally selected SWS is unable to satisfactorily handle the request. The MWS determines this through QoS monitoring techniques which we describe in Section 5.3.1.

The 3-way satisfaction approach proposed earlier during the selection process can also be used in the same manner in the substitution process. The substitution process involves calculating the highest selection score among the SWSs in the community. One main difference is that when determining which

SWS to perform the substitution, the MWS has to ignore SWSs that had previously failed for the same request. The MWS can do this by keeping track of which SWS is handling which request. Lastly for the substitution process, we present the SLA implications for the substitution process in Section 5.3.2.

5.3.1 QoS Monitoring

QoS monitoring forms the first component to the substitution process. This is because the MWS needs to know if and when a substitution needs to occur. The MWS can proceed with the substitution only after it discovers that the original SWS cannot satisfactorily handle the request.

In Section 2.2.1 we mentioned three general techniques for monitoring QoS. The first, a trusted monitor, exists inherently in a community of Web service. This is because all communication between the client and the SWSs goes through the MWS. The second technique, running monitoring code on the provider's side can be implemented in our approach. However, further additions have to be added to the SLA in order for the monitoring code to be effective. The third technique, a trusted party is not required in our approach because the MWS can already perform the monitoring role. In addition, we assume the MWS to be trustworthy from the perspective of the community.

Since all communication between the client and the SWSs goes through the MWS, the MWS is aware of the moment when a SWS has completed the handling of the request and the QoS of the reply. An exception to this can include QoS properties that require feedback from the client. For example, the way that reliability has been defined in this thesis is determined by the expectation of the client. Therefore, a feedback mechanism needs to be in place where the client informs the MWS the level of reliability that the request was handled.

One mechanism that the MWS can use is the *timeout* [79]. The way a timeout works is that the MWS waits a certain period of time before deciding that the SWS is taking too long to handle the request, the MWS can then proceed with the substitution process. One crucial factor in how well the timeout can work is in choosing the length of time. If the timeout is chosen to be too short, the SWS may not have be given enough time to handle the request. On the contrary, if given too much time, the SWS may have failed and the MWS may have waited longer than necessary to start the substitution process.

One method of avoiding the problem with a long timeout is to introduce monitoring code on the provider's side. The MWS can periodically receive a signal (or a ping) from the monitoring code informing the MWS that the SWS is still running.

5.3.2 SLA Implications for Substitution

A substitution is never ideal but nevertheless its implications should be agreed upon in the respective SLAs. The supplier-side SLA and the client-side SLA need to be approached differently since they concern different parties. In this section we first describe the SLA implications for the client-side SLA and then describe the SLA implications for the supplier-side SLA.

5.3.2.1 Client-side SLA Implications for Substitution

The biggest implication to the client when a substitution is necessary is the longer execution duration from the perspective of the client. For each substitution that takes place, it takes longer for the client to receive a reply from the MWS. The execution duration should be considered alongside any other QoS property. When the MWS sends the SLA to the client, the MWS needs to keep this in mind, and it might want to only be minimally penalised if the execution duration QoS property is not adhered to.

5.3.2.2 Supplier-side SLA Implications for Substitution

There are more implications on the supplier-side SLA. The MWS has to consider the following:

- How the monitoring of the SWS occurs. There might be contention if the SWS that was substituted feels that the substitution was unjust (for example, if the timeout was too short).
- Imposing a penalty on the SWS that failed to handle the request.
- Giving a reward to the SWSs that replied positively to the request for substitution bid.
- Giving a reward to the SWS that carried out the request successfully.

These 4 factors have to be considered when the MWS submits the supplier-side SLA to each SWS before joining the community. The MWS may want to give a higher reward for replying positively to the request for substitution bid in order to give SWSs more incentive to handle substitution requests. The MWS may also want to include that the reward for successfully handling a substitution request is higher than the reward for handling a normal request.

Although the MWS may not explicitly mention that a request is normal or a substitution, SWSs within the community might consider a request to be a substitution if it receives a request with the same price within a short period of time. On the other hand, the MWS may want to explicitely mention that the request is a substitution since a higher reward for handling a substitution request may mean that the SWSs will attempt to handle the request at a higher QoS level.

5.4 Experiments

Before going into the actual experiments, we first present the experimental set up in Section 5.4.1. This includes how the experiments were conducted and where the values came from. This set up is applicable for all experiments in this thesis.

Next, we present two different set of experiments, the first set focuses on the use of the fixed-price auction in a community of Web services (Section 5.4.2). We compare the fixed-price auction with other methods of selection within a community. The second set of experiments show the 3-way satisfaction approach which considers the satisfaction of all 3 parties for the selection of Web services when a request is handled (Section 5.4.3).

5.4.1 Experiments Introduction and Setup

All the experiments were written in Java using the Eclipse SDK. Each MWS, SWS and client were modelled as agents acting with their own property values. The behaviour of each agent is explained prior to each experiment, this can differ depending on the objective of the experiment.

Most values of the SWS properties were real QoS values taken from [1]. This dataset represents 2507 real Web services that exist on the Web. It includes the QoS values of 9 properties including *availability*, *throughput*, *response time*, and *reliability*. These QoS values were determined by monitoring the Web services over a 6 day period. We have chosen to use the values of *response time* in the real data to represent the *execution duration* for the SWSs in the simulation. Even though *throughput* and *capacity* are different in definition, we have elected to use the values of *throughput* in the real data to represent the *capacity* for the SWSs in the simulation. This is the closest QoS that we could find in the dataset to represent *capacity*. In the dataset, there was no transform needed since *availability* and *reliability* were already presented as percentages while *throughput* and *response time* were presented as discrete values.

Despite our best efforts to find a complete set of real world experimental values, some experimental values for example price still had to be simulated. Another area where the experimental values had to be simulated included the requested QoS property values. For the purposes of this thesis we assume these simulated values were chosen in order to best simulate the real world situation, this is done by randomizing the values across a range of values. The range of values for each simulated values are presented prior to each experiment.

For all experimental results, they were determined by repeating the experiment 100 times and taking the average. This is done in order to ensure that there are no obscure results due to randomness.

5.4.2 Fixed Price Auction Experiments

In this section, we present the experiments focused on the fixed-price auction system. The objective of these experiments is to compare the fixed-price auction mechanism with previous methods of selection. This is done by measuring the number of handled requests and the QoS of the requests provided by the different mechanisms. These two properties can help measure the effectiveness of sustaining the quality of Web services. This is done in order to see what effects or improvements (if any) there are in using the fixed-price auction system. The mechanisms that we compare against each other are:

1. Basic method of selecting the SWS to handle the request based on the CNP (Selection mechanism 1)
2. Auction based selection where the MWS sends the SWSs all QoS requirements except the price (Selection mechanism 2)
3. Fixed price auction where the MWS only sends the price to the SWSs (Selection mechanism 3)

We perform the simulations first in the situation where no substitution is necessary, and then where substitution is possible. This is done in order to analyse the performance of the selection mechanism only and the substitution mechanism separately.

In order to illustrate the differences between the 3 methods, we observe the simulations where the request QoS requirements increase over time. One way of interpreting this behaviour are greedy clients where they ask for a higher level of QoS for the same price. The price offered by the clients over time does not change in our simulations. For the case of the second mechanism where the MWS sends the SWSs all QoS requirements except the price, we consider a property which we call the price satisfaction. This is the gain in satisfaction for the client when it is charged a lower amount for the request.

This section is divided as such. First we present the preliminaries required for the experiments in Section 5.4.2.1. In Section 5.4.2.2 we present the values of certain properties that we used in our experiments. Next we present the experiments in the following sections:

- Number of Handled Requests for all 3 selection mechanisms (Section 5.4.2.3)
- QoS measurements for all 3 selection mechanisms (Section 5.4.2.4)
- Number of Handled Requests for all 3 substitution mechanisms (Section 5.4.2.5)
- QoS measurements for all 3 substitution mechanisms (Section 5.4.2.6)

Finally we conclude the fixed-price auction experiments section in Section 5.4.2.7.

5.4.2.1 Auction Experiments - Preliminary

When the client receives a reply from the community, it does not know which SWS handled the request. It receives the reply of the handled request from the community. We therefore define the QoS properties of the handled request in Table 5.1.

Table 5.1. List of Handled Request Properties

Property	**Notation**	**Description**
Handled Availability	$hanAv_{c-r}^{t}$	Handled Probability of *availability*
Handled Reliability	$hanRl_{c-r}^{t}$	Handled Probability of *reliability*
Handled Execution Duration	$hanEd_{c-r}^{t}$	Handled *Execution duration*
Handled Price	$hanPr_{c-r}^{t}$	Handled Amount of currency client is charged

In order to evaluate how well the community handles a request, we define client satisfaction as a function of the requested availability, requested reliability, and requested executed duration matched with their handled equivalents. In general, a client is more satisfied if the QoS requirements of the client was met and the client is less satisfied if the QoS requirements was not met.

We introduce the experiment client satisfaction for availability, reliability, and execution duration in Equation 5.8, Equation 5.9, and Equation 5.10 respectively.

$$exClientSatisAv^t_{c-r} = \begin{cases} 1 & \text{if } reqAv^t_{c-r} \leq hanAv^t_{c-r} \\ \frac{hanAv^t_{c-r}}{reqAv^t_{c-r}} & \text{if } reqAv^t_{c-r} > hanAv^t_{c-r} \end{cases} \tag{5.8}$$

$$exClientSatisRl^t_{c-r} = \begin{cases} 1 & \text{if } reqRl^t_{c-r} \leq hanRl^t_{c-r} \\ \frac{hanRl^t_{c-r}}{reqRl^t_{c-r}} & \text{if } reqRl^t_{c-r} > hanRl^t_{c-r} \end{cases} \tag{5.9}$$

$$exClientSatisEd^t_{c-r} = \begin{cases} \frac{reqEd^t_{c-r}}{hanEd^t_{c-r}} & \text{if } reqEd^t_{c-r} < hanEd^t_{c-r} \\ 1 & \text{if } reqEd^t_{c-r} \geq hanEd^t_{c-r} \end{cases} \tag{5.10}$$

Finally we can define the overall experiment client satisfaction in Equation 5.11.

$$\begin{aligned} exClientSatis^t_{c-r} = {} & w_{exClientSatisAv} \cdot exClientSatisAv^t_{c-r} \\ & + w_{exClientSatisRl} \cdot exClientSatisRl^t_{c-r} \\ & + w_{exClientSatisEd} \cdot exClientSatisEd^t_{c-r} \end{aligned} \tag{5.11}$$

Where $w_{exClientSatisAv}$, $w_{exClientSatisRl}$, $w_{exClientSatisEd}$ are the weights for the experiment client satisfaction for availability, reliability, execution duration and price respectively.

For the purposes of the simulations, we have set these 3 weights to be of the same value. In practice, a system administrator is free to set them differently according to its own priorities.

The satisfaction due to the price is calculated separately in Equation 5.12. This is because the satisfaction is only applicable in the second mechanism

Table 5.2. Auction Experiment Properties

Experiment Properties	**Mean**	**Min**	**Max**
SWS Price	12	10	14
SWS Failure Probability	0	20	20
Requested Budget	15	10	20
Requested Availability	32.5	30	35
Requested Reliability	22.5	20	25
Requested Execution Duration	4250	3500	5000

where the MWS sends all QoS except to the price to the SWSs. In the other two mechanisms, the handled price is always the same as the requested price.

$$exClientSatisPr_{c-r}^{t} = 1 - \frac{hanPr_{c-r}^{t}}{reqPr_{c-r}^{t}} \tag{5.12}$$

5.4.2.2 Auction Experiments - Properties

In these experiments we simulate 50 SWSs over 10 time units. At the end of each time unit, the QoS requirements for the requested availability, requested reliability was increased by 10% while the requested execution duration was decreased by 10%. In order to illustrate the benefits of the selection mechanisms, the community was sent more requests that it can handle (according to its capacity determined by the SWSs). In the experiment we set this number to 400 requests per unit time.

In Table 5.2, we list other values of the properties that were used in the experiment.

SWS Failure Probability refers to the probability (in percentage) of how likely it is for the SWS to fail. This property is only used for the simulations involving the substitution mechanism.

5.4.2.3 Number of Handled Requests

For the first 2 selection mechanisms, individual SWSs do not bid if it cannot match the QoS requirements.

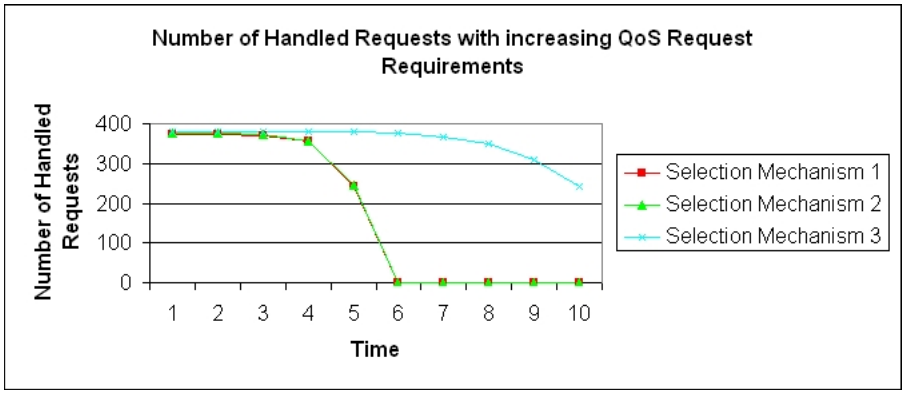


Fig. 5.1. Number of Handled Requests for all 3 selection mechanisms

In Figure 5.1, we display the number of number of handled requests for all 3 selection mechanisms. We see that when the client increases the QoS requirements to a certain point, the first two approaches fail to handle any of the requests. This happens when the community does not receive a single bid from the SWSs. In the third selection mechanism, SWSs are allowed to return a QoS bid which allows the SWS to handle the request at a lower QoS. In the 3rd selection mechanism, the limit of the number of requests that are handled depends on the capacities of the SWSs in the community.

The important part that can be derived from this experiment is to note that the 3rd selection mechanism does not perform much worse than the previous 2 mechanisms. Its benefit of being able to handle all requests as the QoS requirements increases is a given. However while the QoS requirements still remain low and the Web services are able them regardless of the price, the fixed-price auction mechanism is still able to keep pace with the previous 2 selection mechanisms.

5.4.2.4 QoS measurements

Although all the requests are handled in the 3rd selection mechanism, the lower QoS provided by the SWSs lead to a lower client satisfaction. We observe this in Figure 5.2. Another important observation in the client satisfaction is that in the initial period when the first 2 selection mechanisms are able to handle the requests, there is not much difference between the client satisfaction. This means that during the period which the first 2 mechanisms can handle all requests, the 3rd selection mechanism can perform equally as the first 2 selection mechanisms. Since client satisfaction for each request is 0 when the request is not handled, we observe that the client satisfaction for the first 2 selection mechanisms drop to 0 when the number of handled requests for both selection mechanisms drop to 0.

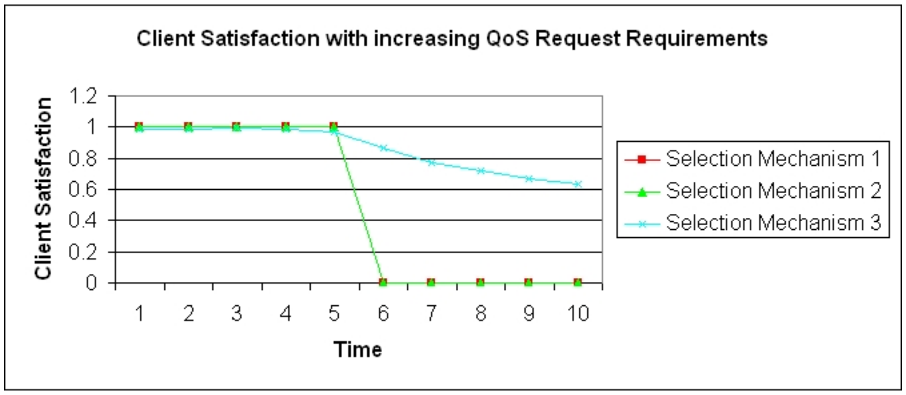


Fig. 5.2. Client Satisfaction for all 3 selection mechanisms

5.4.2.5 Number of Handled Requests With Substitution

For the substitution experiments to occur, we have to introduce the possibility for SWSs to fail. This gives the MWS the possibility to substitute the failed SWS with another functional SWS. For simulation purposes, we have set all substitutions to successfully handle the request, this prevents the situation of the MWS searching for more than 1 substitute SWS to handle a request. i.e. The handling of each request can fail at most once.

Similar to the first 2 selection mechanisms, for the first 2 substitution mechanisms, individual SWSs do not bid if it cannot match the QoS requirements.

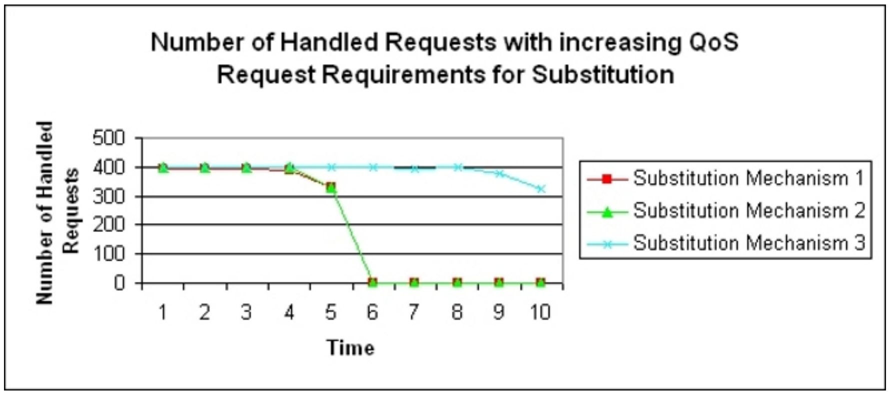


Fig. 5.3. Number of Handled Requests for all 3 substitution mechanisms

In Figure 5.3, we display the number of number of handled requests for all 3 substitution mechanisms. We see that when the client increases the QoS requirements to a certain point, the first two approaches fail to handle any of the requests. This happens when the community does not receive a single bid from the SWSs. In the third substitution mechanism, SWSs are allowed to return a QoS bid which allows the SWS to handle the request at a lower QoS. In the 3rd substitution mechanism, the limit of the number of requests that are handled depends on the capacities of the SWSs in the community.

The important part that can be derived from this experiment is to note that the 3rd selection mechanism does not perform much worse than the previous 2 mechanisms. Its benefit of being able to handle all requests as the QoS requirements increases is a given. However while the QoS requirements still remain low and the Web services are able them regardless of the price, the fixed-price auction mechanism is still able to keep pace with the previous 2 selection mechanisms.

In addition to the number of handled request, another additional property that is interesting is the number of substitutions that occured over time. In Figure 5.4, we see the number of substitutions for all 3 substitution mechanisms. We notice that the number of substitutions drop to 0 after a certain point for the first 2 substitution mechanisms. This is because the SWSs cannot bid to provide a lower QoS for a lower price. SWSs do not submit a bid if it cannot meet the request QoS requirements.

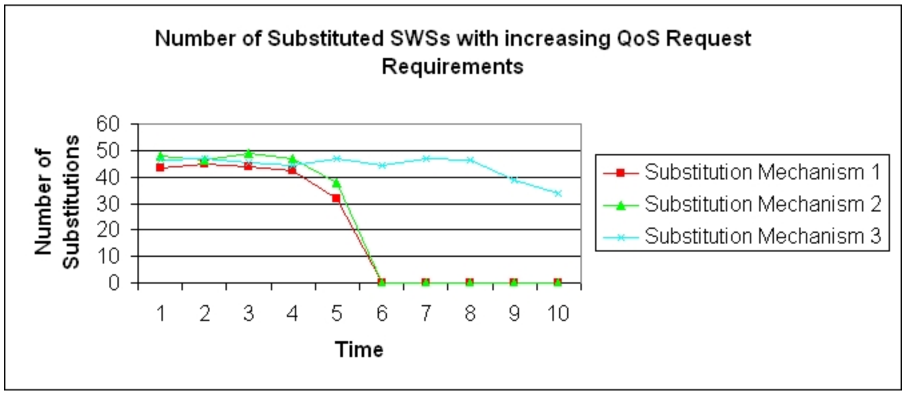


Fig. 5.4. Number of Substitutions for all 3 substitution mechanisms

5.4.2.6 QoS measurements With Substitution

Although all the requests are handled in the 3rd substitution mechanism, the lower QoS provided by the SWSs lead to a lower client satisfaction. We observe this in Figure 5.5. Another important observation in the client satisfaction is that in the initial period when the first 2 substitution mechanisms are able to handle the requests, there is not much difference between the client satisfaction. This means that during the period which the first 2 mechanisms can handle all requests, the 3rd substitution mechanism can perform equally as the first 2 substitution mechanisms. Since client satisfaction for each request is 0 when the request is not handled, we observe that the client satisfaction for the first 2 substitution mechanisms drop to 0 when the number of handled requests for both substitution mechanisms drop to 0.

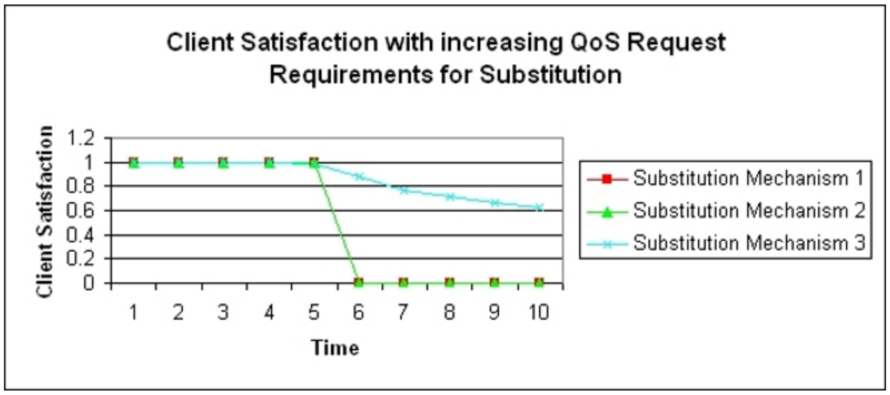


Fig. 5.5. Client Satisfaction for all 3 substitution mechanisms

5.4.2.7 Auction Experiments - Conclusion

In our simulations we demonstrate the benefits of implementing an auction system in a community. We showed that an auction mechanism that give communities the flexibility to provide the same service to the client at a lower cost, thus increasing the client's price satisfaction. The fixed-price auction allows the community to select the SWS which is able to provide the best service based on the fixed price by client. In cases where the client underestimates the price it is willing to pay with respect to the other QoS requirements, the community is still able to find a SWS to handle the request, albeit at a lower QoS than the original request.

We have also shown through the experiments that the fixed-price auction selection mechanism does not provide a worse level of QoS to the clients compared to the previous 2 mechanisms. This is applicable even at cases where the level of QoS is able to be handled by the Web services. This is important in order to sustain the level of QoS to Web services.

This approach of selection may not be applicable to all cases. For example, some clients may insist on having its request handled at that level of QoS, or not handled at all. In such a scenario, the community can revert to either the first or second selection mechanism.

The substitution mechanisms presented help to show how the same benefits in the selection mechanism can be replicated when the community is searching for a substitute SWS.

5.4.3 3-Way Satisfaction Approach

In this section we present the experiments conducted for the 3-way satisfaction selection in our approach.

The objectives of the 3-way selection experiments is to illustrate the benefits of using the 3-way satisfaction selection by comparing this with existing approaches. One such approach is a selection process focused only on the QoS provided to the client. This is commonly found within the domain of composition of Web services. We call this the client satisfaction only selection approach. This approach is done by using only the QoS provided to the client as a benchmark to determine which Web service handles the request. On the other hand, in the 3-way satisfaction selection approach, the QoS provided to the client (Client satisfaction), the workload of the Web services (SWS Satisfaction) as well as the revenue generated (MWS Satisfaction) is considered in the selection process. In both cases, the fixed-price auction system is used.

The SWS and MWS satisfaction is important in sustaining the satisfaction of the client. This is because at the end of the day the MWS is responsible for making the selection and the SWS is ultimately responsible for actually handling the request. If the SWS satisfaction were to drop too low, the SWS can always choose to leave the community based on the terms and conditions agreed upon in the SLA.

In order to illustrate the benefits of the 3-way satisfaction selection approach, we observe the simulations where the client increases the number of requests over time if it is satisfied. The level of QoS required by the clients does not change over time in our simulations.

The measurements that we measure to compare both selection mechanisms is as follows:

- Client Satisfaction
- SWS Satisfaction
- MWS Satisfaction
- Total Satisfaction
- MWS Revenue
- Number of Handled Requests

These measurements are chosen in line with the goal to sustain the quality of Web services.

In Section 5.4.3.1 we present the properties that we used in our experiments. Next we present the experiments in the following sections:

- Selection Mechanism (Section 5.4.3.2)
- Substitution Mechanism (Section 5.4.3.3)

Finally we conclude the 3-way satisfaction experiments section in Section 5.4.3.4.

5.4.3.1 3-way Satisfaction Experiment - Properties

In these experiments we simulate 10 SWSs over 20 time units. At the end of each time unit, the number of requests sent by the client is increased or decreased depending on the client's satisfaction. In the experiment we set this number to 60 requests per unit time.

In Table 5.3, we list other values of the properties that were used in the experiment.

SWS Bid Price Difference refers to the difference in the bid price with the requested price (See Section 4.4). These values determine the amount of

Table 5.3. 3-Way Satisfaction Experiment Properties

Experiment Properties	**Min**	**Max**
SWS Price	12	35
SWS Bid Price Difference	1	5
SWS gWl	0.3	0.3
Requested Budget	10	40
Requested Availability	30	80
Requested Reliability	30	80
Requested Execution Duration	3500	5000

revenue that the MWS receives, and thus affects which SWS performs the selection or substitution.

5.4.3.2 3-way Satisfaction Approach - Selection Mechanism

The experiments were conducted and the comparison between the 3-way selection and a selection based on client satisfaction only were carried out. For the selection mechanism experiments, we have modelled the SWSs to always successfully handle the request, thus no substitution is necessary.

Client Satisfaction Comparison

We present the client satisfaction when selection is done via 3-way versus client satisfaction only in Figure 5.6.

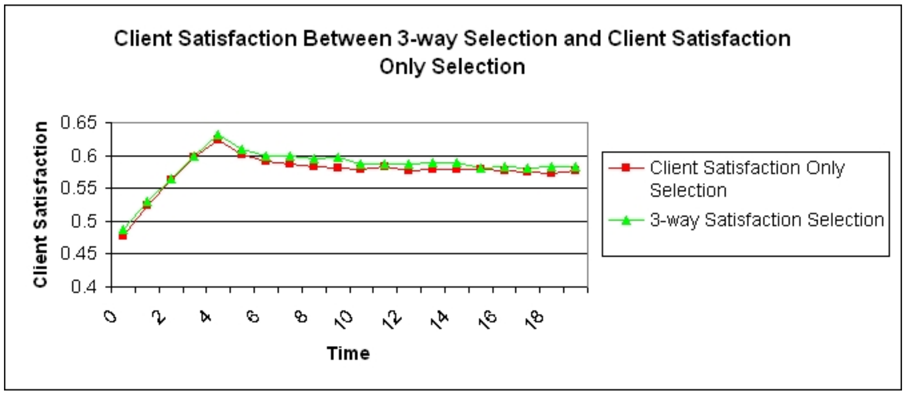


Fig. 5.6. Client Satisfaction Comparison

We see in figure 5.6 that the client satisfaction for both selection mechanisms remain fairly consistently the same. This is because both mechanisms consider client satisfaction. In the case where client satisfaction is only considered, it is the main priority for the selection of Web services. Even though client satisfaction only forms a part of the total satisfaction in the 3-way satisfaction selection mechanism, the client satisfaction remains close to the client satisfaction when the client satisfaction only selection mechanism is used.

SWS Satisfaction Comparison

We present the SWS satisfaction when selection is done via 3 way versus client satisfaction only in Figure 5.7.

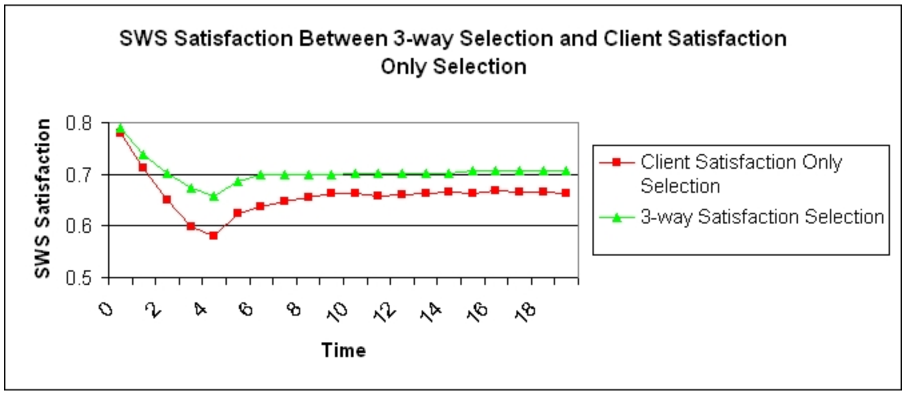


Fig. 5.7. SWS Satisfaction Comparison

We see in figure 5.7.that the 3-way satisfaction selection can sustain the SWS satisfaction better than the selection using client satisfaction only. This is because in the 3-way satisfaction selection, the community takes into account the SWS satisfaction in the selection process. For the client satisfaction selection only, SWS satisfaction is not taken into account when selecting a Web service to handle the request.

MWS Satisfaction Comparison

We present the MWS satisfaction when selection is done via 3 way versus client satisfaction only in Figure 5.8.

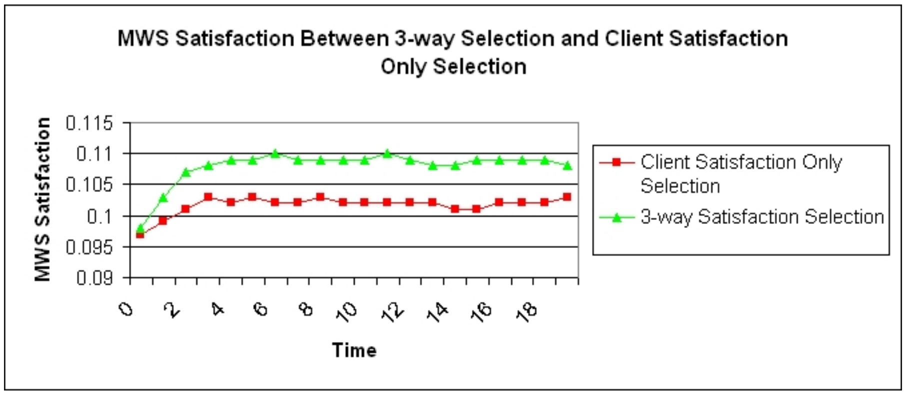


Fig. 5.8. MWS Satisfaction Comparison

We see in figure 5.8.that the 3-way satisfaction selection can sustain the MWS satisfaction better than the selection using client satisfaction only. This is because in the 3-way satisfaction selection, the community takes into account the MWS satisfaction in the selection process. For the client satisfaction selection only, MWS satisfaction is not taken into account when selecting a Web service to handle the request.

Total Satisfaction Comparison

We sum up the client, SWS and MWS satisfaction to determine the total comparison over time. We present the total satisfaction when selection is done via 3 way versus client satisfaction only in Figure 5.9. We see here that the total satisfaction is higher using the 3-way satisfaction selection mechanism. This is expected from the previous results. Since the 3-way satisfaction selection did not adversely affect the client satisfaction too badly, it is clear that when the total satisfaction is considered, the 3-way satisfaction selection will give a higher total satisfaction than the selection mechanism where only client satisfaction is considered.

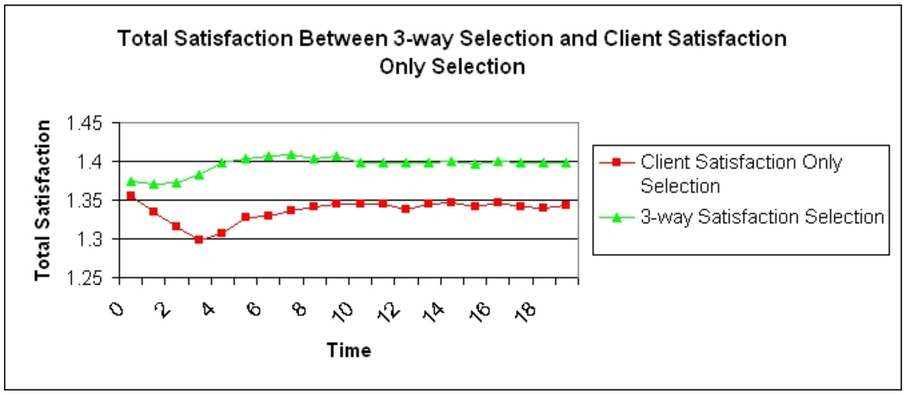


Fig. 5.9. Total Satisfaction Comparison

MWS Revenue Comparison

Another property that we can measure is the MWS revenue. The MWS revenue is affected by the number of requests that were handled as well as the selection decision. We present the MWS revenue when selection is done via 3 way versus client satisfaction only in Figure 5.10. We see here that the MWS revenue for the 3-way satisfaction selection mechanism is higher than if using the client satisfaction only selection mechanism.

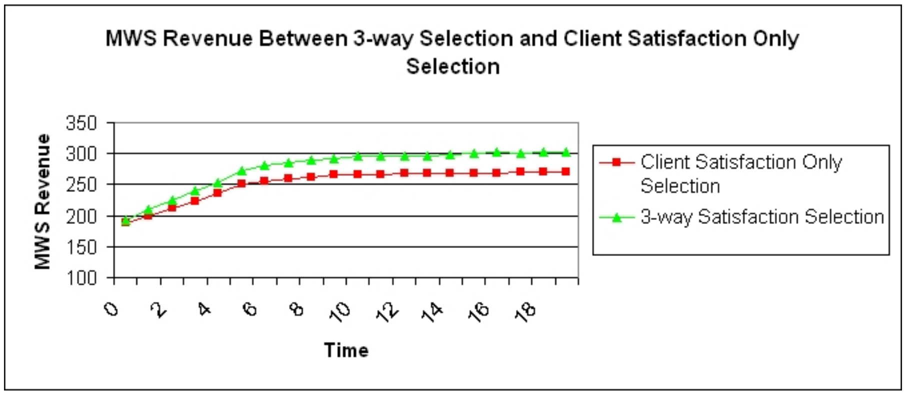


Fig. 5.10. MWS Revenue Comparison

Number of Handled Requests Comparison

For completeness, we look at the number of handled requests comparing both selection mechanisms. We present the number of handled requests when selection is done via 3 way versus client satisfaction only in Figure 5.11. We see here that the number of handled requests for the 3-way satisfaction selection mechanism is similar to the number of handled requests if using the client satisfaction only selection mechanism.

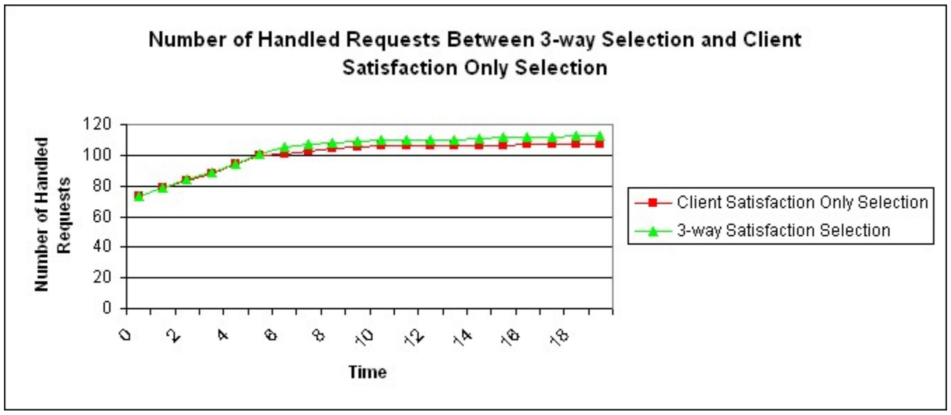


Fig. 5.11. Number of Handled Requests Comparison

5.4.3.3 3-way Satisfaction Approach - Substitution

We replicate the same selection experiment properties for the substitution process. In this case we have to add an additional property which we consider the probability that a SWS fails to handle the request. We have set this value to be 20%. When a SWS fails to handle the request, the MWS finds a substitution SWS to handle the request. In addition, the response time for the originally failed SWS is added to the response time for handling the request. For this simulation, we have set the maximum time a SWS can fail to be once, a substitution SWS will always successfully handle the request. We feel that such an experiment is sufficient to illustrate the benefits of the 3-way satisfaction approach.

The substitution experiments were conducted and the comparison between the 3-way substitution and a substitution based on client satisfaction only were carried out.

Client Satisfaction For Substitution

We present the client satisfaction when substitution is done via 3 way versus client satisfaction only in Figure 5.12. We notice that the client satisfaction for both substitution mechanisms remain consistently the same. Our focus here again is on the impact of taking into account the SWS and MWS satisfactions during the 3-way satisfaction substitution mechanism. Since the client satisfaction is the only consideration during the client satisfaction only substitution mechanism, it can be considered the benchmark which the 3-way satisfaction substitution mechanism would like to target. We see here that since the client satisfaction for both substitution mechanisms are fairly similar, we can conclude that the inclusion of the SWS and MWS satisfaction in the 3-way satisfaction substitution mechanism does not too adversely affect the client satisfaction.

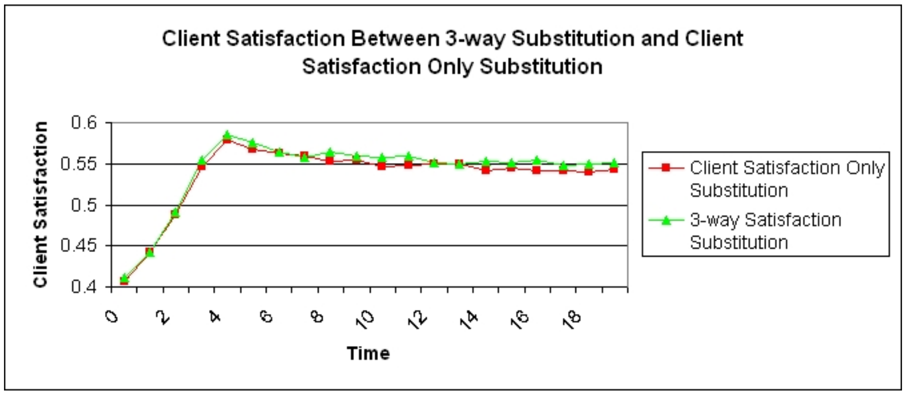


Fig. 5.12. Client Satisfaction For Substitution

SWS Satisfaction For Substitution

We present the SWS satisfaction when substitution is done via 3 way versus client satisfaction only in Figure 5.13. The SWS satisfaction remains consistently higher for the 3-way satisfaction substitution mechanism than the substitution based on client satisfaction only. This is due to the consideration of the SWS satisfaction in the 3-way satisfaction substitution mechanism. Since only the client satisfaction is considered during the client satisfaction only substitution mechanism, SWS satisfaction is neglected during the substitution process.

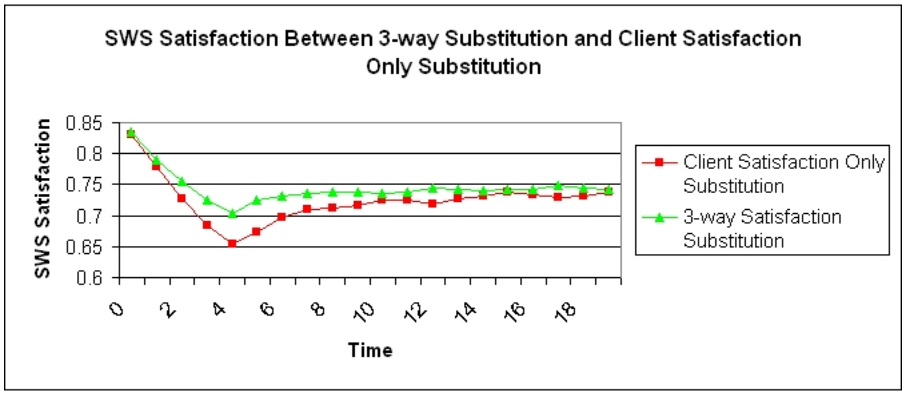


Fig. 5.13. SWS Satisfaction For Substitution

MWS Satisfaction For Substitution

We present the MWS satisfaction when substitution is done via 3 way versus client satisfaction only in Figure 5.14. The MWS satisfaction remains consistently higher for the 3-way satisfaction substitution mechanism than the substitution based on client satisfaction only. This is due to the consideration of the MWS satisfaction in the 3-way satisfaction substitution mechanism. Since only the client satisfaction is considered during the client satisfaction only substitution mechanism, MWS satisfaction is neglected during the substitution process.

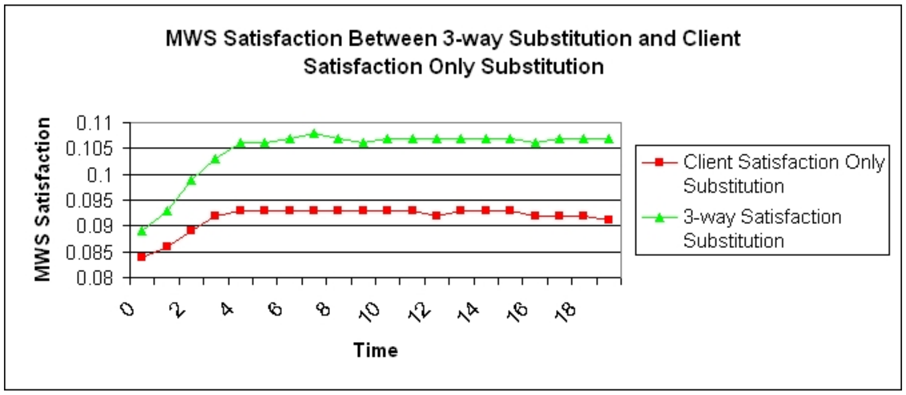


Fig. 5.14. MWS Satisfaction For Substitution

Total Satisfaction For Substitution

We sum up the client, SWS and MWS satisfaction to determine the total comparison over time. We present the total satisfaction when substitution is done via 3 way versus client satisfaction only in Figure 5.15. This graph is evident from the past few experiments, however we still made the graph for clarity. Since the client satisfaction is fairly close between two substitution mechanisms, however the SWS and MWS satisfaction is higher for the 3-way satisfaction substitution mechanism compared to the client satisfaction only substitution mechanism, the total satisfaction is thus higher for the 3-way satisfaction substitution mechanism.

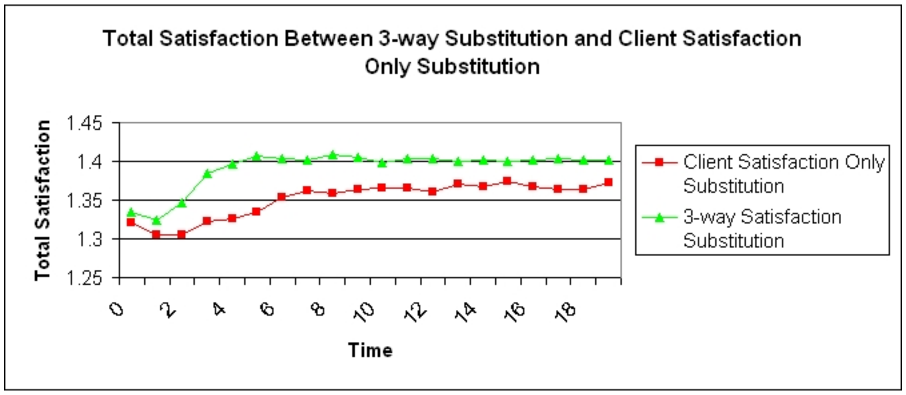


Fig. 5.15. Total Satisfaction For Substitution

MWS Revenue For Substitution

A quantifiable property that can be measured is the MWS revenue. The MWS revenue is affected by the number of requests that were handled as well as the selection and substitution decision. We present the MWS revenue when substitution is done via 3 way versus client satisfaction only in Figure 5.16.

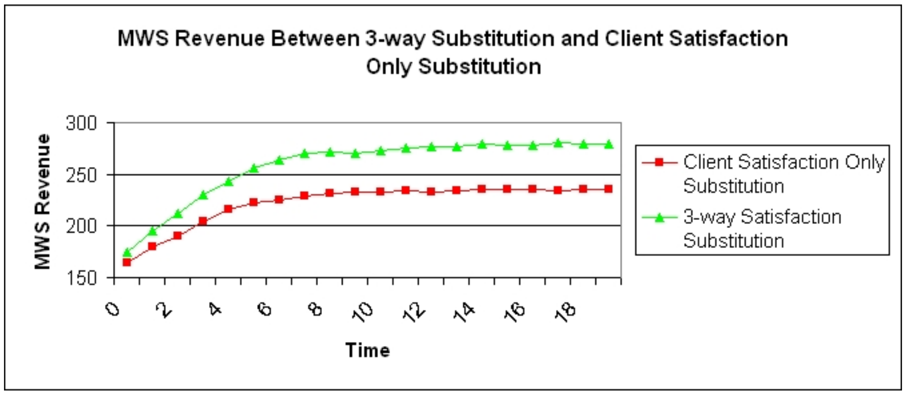


Fig. 5.16. MWS Revenue For Substitution

Number of Handled Requests For Substitution

For completeness, we look at the number of handled requests comparing both substitution mechanisms. We present the number of handled requests when substitution is done via 3 way versus client satisfaction only in Figure 5.17. We see here that the number of handled requests for the 3-way satisfaction substitution mechanism is similar to the number of handled requests if using the client satisfaction only substitution mechanism.

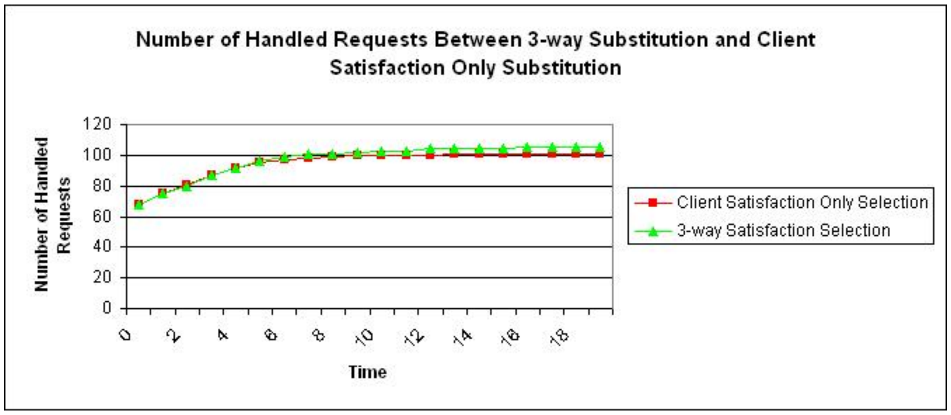


Fig. 5.17. Number of Handled Requests For Substitution

5.4.3.4 3-way Satisfaction Approach - Conclusion

For the second series of experiments in the user request management chapter, we presented experimental results using real-world properties to show the benefits of using the 3-way selection and substitution mechanisms. In our experiments we see that although the client satisfaction remain close regardless of the selection mechanism used, we see the difference where it comes to the SWS and MWS selection. The 3-way satisfaction selection mechanism was able to provide a higher SWS and MWS satisfaction compared to the selection mechanism which only considered the client satisfaction. The advantage of the 3-way selection and substitution mechanisms are further emphasized in the MWS revenue.

6 Member Management

In this chapter we address the member management of a community of Web services. In this chapter we take a more macro-approach in order to sustain the quality of Web services using the framework of a community of Web services. Instead of looking at each individual request as we did in Chapter 5, in this chapter we look at the requests sent by clients over time, and restructure the community in order to better sustain the quality of Web services. This restructuring of the community includes the invitation of additional Web services and expulsion of SWSs in the community. The responsibility of deciding which Web services to invite and which SWSs to expel lies with the MWS of the community.

At the very basic level, expulsion of SWSs from the community may not be needed since CNP is used. This is because if the MWS feels that a SWS is unable to handle the requests at a certain level, subsequently less and less requests are redirected to that particular SWS. This leads to the SWSs eventual departure from the community since it is not receiving any requests from the community. However, we mentioned in Section 4.5 that there might be certain terms and conditions defined in the supplier-side SLA that complicate the situation. For example, in the event that a guaranteed workload was stated in the SLA, then the MWS is required to direct a certain number of requests to the SWS. If the SWS consistently fails to handle the request at a certain level, the MWS may thus require the SWS's expulsion.

Before any restructuring of the community can take place, it is important to know what requirements are needed in order for the restructuring to be done. It is impossible to tell beforehand with 100 percent certainty what exactly these requirements might be because the clients have not sent the requirements yet. Nevertheless, we propose a mechanism (Section 6.1) in order to predict the future requirements of clients, this includes the number of requests and the type of requests (QoS properties) of each client.

Having predicted the requirements of future requests, the community would then need to decide whether its current number and type of Web services in the community is able to handle these future requirements. In order to do so, a mechanism needs to be in place in order to measure what the values of each QoS property of the community are. We call this the Quality of Community. The QoC of a community also allows the community to accurately publish its QoC on service registries and allows the community to measure and compare its QoS over time. Whether the QoC is published on service registries or any kind of comparison over time depends on the choices of the system administrator. This mechanism of calculating the QoC of a community is presented in Section 6.2.

After knowing the requirements needed in order to handle future requests and the QoS of a community, the community can decide whether more Web services are needed or some Web services can be expelled from the community. From the pool of potential Web services that might join the community, we propose a mechanism in order to decide which Web service(s) need to be invited to the community in Section 6.3.

We model the Web service's decision making process when it receives an invitation from the community. The Web service has to decide based on the terms and conditions presented in the SLA whether or not it would join the community. This is presented in Section 6.4.

In Section 6.5, we introduce a mechanism which allows the community to decide which SWSs it should expel from the community. This allows the community to retain the SWSs which it considers will be most beneficial.

Finally, in order to illustrate the benefits of the proposed mechanisms, we present some experiments in Section 6.6 and conclude the chapter.

6.1 Client Model

In this section, we propose a framework in order to model the client. The model will use the QoS properties defined in Section 3.2. In addition, the client model needs to also consider the number of requests the community receives from the client. Following the notations defined in Section 3.2.3, we define R_c^t to be the set of requests that client c sends out at time t and the number of requests that client c sends out at time t is therefore defined as $|R_c^t|$. Each request is defined by the tuple $reqQoS_{c-r}^t$ where $r \in R$.

In our thesis we present a prediction mechanism that could be used to model the client. Other more elaborate prediction mechanisms exist and were discussed earlier in Section 2.3.3. In this thesis, we assume that inviting and expelling WSs in a community can be done within the same time unit.

In Section 6.1.1 we present the formula to determine a predicted number of requests that are sent by clients. Next in Section 6.1.2 we present the formula to determine the predicted availability for the requests sent by the clients. In Section 6.1.3 we present the formula to determine the predicted reliability for the requests sent by the clients. In Section 6.1.4 we present the formula to determine the predicted execution duration for the requests sent by the clients. In Section 6.1.5 we present the formula to determine the predicted price for the requests sent by the clients. Finally in Section 6.1.6, we summarize the prediction properties.

6.1.1 Number of Requests

We first propose a mechanism in order to predict the number of requests $pR_c^{t_1}$ at time t_1 from client c in Equation 6.1. This mechanism takes into account *decay* by placing a higher emphasis on recent values and placing a lower emphasis on values that occurred further in the past.

$$pR_c^{[t_1,\text{-}]} = \frac{\sum_{t=t_0}^{t_1} (|R_c^t| \times e^{-\lambda(t_1-t)})}{\sum_{t=t_0}^{t_1} e^{-\lambda(t_1-t)}} \tag{6.1}$$

Where e is the euler's number and λ is the decay rate.

This mechanism allows the community to identify the number of requests that the community may receive in the future. However, one limitation is that this mechanism is less accurate to sudden big changes in the number of requests sent by the client.

The community can now consider all clients in the set of clients, C^{t_1} that sent requests to the community at time t_1. The predicted number of requests is thus a summation across all clients, defined in Equation 6.2.

$$pCR_C^{[t_1,\text{-}]} = \sum_{c \in C^{t_1}} pR_c^{[t_1,\text{-}]} \tag{6.2}$$

6.1.2 Predicted Requested Availability

Since a client is allowed to send multiple requests of different QoS properties at once, it can be challenging to predict the requested availability of all the requests. The mechanism proposed takes into account the mean of the availability of all requests that have been received. The formula also takes into consideration the concept of decay, whereby the availability of requests that were received more recently have a higher weightage than the availability of requests that were received further into the past. We define the predicted requested availability, $pReqAv_c^{t_1}$ of client c at time t_1 in Equation 6.3.

$$pReqAv_c^{[t_1,\text{-}]} = \frac{\sum_{t=t_0}^{t_1} (\frac{\sum_{r \in R_c^t} reqAv_{c-r}^t}{|R_c^t|} \times e^{-\lambda(t_1-t)})}{\sum_{t=t_0}^{t_1} e^{-\lambda(t_1-t)}} \tag{6.3}$$

The community can now consider all clients in the set of clients, C^{t_1} that sent requests to the community at time t_1. The predicted requested availability for the community is thus the mean across all clients, defined in Equation 6.4.

$$pCReqAv_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqAv_c^{[t_1,\text{-}]}}{|C^{t_1}|} \tag{6.4}$$

6.1.3 Predicted Requested Reliability

Following the same reasoning presented for the predicted requested availability, we define the predicted requested reliability of client c at time t_1 in Equation 6.5.

$$pReqRel_c^{[t_1,\text{-}]} = \frac{\sum_{t=t_0}^{t_1} (\frac{\sum_{r \in R_c^t} reqRel_{c-r}^t}{|R_c^t|} \times e^{-\lambda(t_1-t)})}{\sum_{t=t_0}^{t_1} e^{-\lambda(t_1-t)}} \tag{6.5}$$

The community can now consider all clients in the set of clients, C^{t_1} that sent requests to the community at time t_1. The predicted requested reliability for the community is thus the mean across all clients, defined in Equation 6.6.

$$pCReqRel_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqRel_c^{[t_1,\text{-}]}}{|C^{t_1}|} \tag{6.6}$$

6.1.4 Predicted Requested Execution Duration

Following the same reasoning presented for the predicted requested availability, we define the predicted requested execution duration of client c at time t_1 in Equation 6.7.

$$pReqEx_c^{[t_1,\text{-}]} = \frac{\sum_{t=t_0}^{t_1} (\frac{\sum_{r \in R_c^t} reqEx_{c-r}^t}{|R_c^t|} \times e^{-\lambda(t_1-t)})}{\sum_{t=t_0}^{t_1} e^{-\lambda(t_1-t)}} \tag{6.7}$$

The community can now consider all clients in the set of clients, C^{t_1} that sent requests to the community at time t_1. The predicted requested execution duration for the community is thus the mean across all clients, defined in Equation 6.8.

$$pCReqEx_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqEx_c^{[t_1,\text{-}]}}{|C^{t_1}|} \tag{6.8}$$

6.1.5 Predicted Requested Price

Following the same reasoning presented for the predicted requested availability, we define the predicted requested price of client c at time t_1 in Equation 6.9.

$$pReqPr_c^{[t_1,\text{-}]} = \frac{\sum_{t=t_0}^{t_1} \left(\frac{\sum_{r \in R_c^t} reqPr_{c-r}^t}{|R_c^t|} \times e^{-\lambda(t_1-t)}\right)}{\sum_{t=t_0}^{t_1} e^{-\lambda(t_1-t)}} \tag{6.9}$$

The community can now consider all clients in the set of clients, C^{t_1} that sent requests to the community at time t_1. The predicted requested price for the community is thus the mean across all clients, defined in Equation 6.10.

$$pCReqPr_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqPr_c^{[t_1,\text{-}]}}{|C^{t_1}|} \tag{6.10}$$

6.1.6 Summary of Predicted QoS Values

We present the summary of the predicted QoS values in Table 6.1.

Table 6.1. List of Predicted QoS Values for the Community

Property	**Notation**
Predicted Number of Requests	$pCR_C^{[t_1,\text{-}]} = \sum_{c \in C^{t_1}} pR_c^{[t_1,\text{-}]}$
Predicted Requested Availability	$pCReqAv_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqAv_c^{[t_1,\text{-}]}}{\|C^{t_1}\|}$
Predicted Requested Reliability	$pCReqRel_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqRel_c^{[t_1,\text{-}]}}{\|C^{t_1}\|}$
Predicted Requested Execution Duration	$pCReqEx_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqEx_c^{[t_1,\text{-}]}}{\|C^{t_1}\|}$
Predicted Requested Price	$pCReqPr_C^{[t_1,\text{-}]} = \frac{\sum_{c \in C^{t_1}} pReqPr_c^{[t_1,\text{-}]}}{\|C^{t_1}\|}$

6.2 QoS of the Community

In this section we define the Quality of Community(QoC) for the 5 QoS properties. The QoC represents a collective measurement of the QoS values of the Web services within the community. In addition, we also define the variance of the community. The variance of the community indicates the spread of QoS among the SWSs in the community. A higher variance indicates that the SWSs in the community provide a bigger difference in the level of QoS. The variance can be a useful measurement for the community in order to determine whether it would prefer SWSs providing various levels of QoS or a homogeneous level of QoS among the SWSs in the community.

We present the formula for the availability of a community first in Section 6.2.1. In Section 6.2.2, we present the formula for the reliability of a community. In Section 6.2.3, we present the formula for the capacity of a community. In Section 6.2.4, we present the formula for the execution duration of a community. In Section 6.2.5, we present the formula for the price of a community. In Section 6.2.6, we present the formula for the variance of a community.

6.2.1 QoC - Availability

We consider the availability of a community, U, as the mean of the availability of all SWSs within the community. We define this in Equation 6.11.

$$cAv_U^t = \frac{\sum_{j \in J^t} swsAv_j^t}{|J^t|} \tag{6.11}$$

6.2.2 QoC - Reliability

We consider the reliability of a community, U, as the mean of the reliability of all SWSs within the community. We define this in Equation 6.12.

$$cRl_U^t = \frac{\sum_{j \in J^t} swsRl_j^t}{|J^t|} \tag{6.12}$$

6.2.3 QoC - Capacity

We consider the capacity of a community, U, as the sum of the capacity of all SWSs within the community. We define this in Equation 6.13.

$$cCp_U^t = \sum_{j \in J^t} swsCp_j^t \tag{6.13}$$

6.2.4 QoC - Execution Duration

We consider the execution duration of a community, U, as the mean of the execution duration of all SWSs within the community. We define this in Equation 6.14.

$$cEx_U^t = \frac{\sum_{j \in J^t} swsEx_j^t}{|J^t|} \tag{6.14}$$

6.2.5 QoC - Price

We consider the price of a community, U, as the mean of the price of all SWSs within the community. We define this in Equation 6.15.

$$cPr_U^t = \frac{\sum_{j \in J^t} swsPr_j^t}{|J^t|} \tag{6.15}$$

6.2.6 QoC - Variance

The variance of the community, $varC^t$, shows the spread in the QoS of the SWSs within the community. We first define the variance for each individual QoS property in Table 6.2, where J^t refers to the set of SWSs in community U at time t.

We define the variance for community, U, in Equation 6.16. A higher variance indicates that the difference between the QoS among the SWSs in the community is higher, and conversely so.

$$varC_U^{[t]} = varCAv_U^t + varCRl_U^t + varCCp_U^t + varCEx_U^t + varCPr_U^t \tag{6.16}$$

Table 6.2. List of 5 QoC Variance properties for the Community

Property	**Notation**	**Formula**
QoC Variance Availability	$varCAv_U^t$	$\frac{\sum_{j \in J^t} (swsAv_j^t)^2 - \frac{(\sum_{j \in J^t} (swsAv_j^t))^2}{n}}{n}$
QoC Variance Reliability	$varCRl_U^t$	$\frac{\sum_{j \in J^t} (swsRl_j^t)^2 - \frac{(\sum_{j \in J^t} (swsRl_j^t))^2}{n}}{n}$
QoC Variance Capacity	$varCCp_U^t$	$\frac{\sum_{j \in J^t} (swsCp_j^t)^2 - \frac{(\sum_{j \in J^t} (swsCp_j^t))^2}{n}}{n}$
QoC Variance Execution Duration	$varCEx_U^t$	$\frac{\sum_{j \in J^t} (swsEx_j^t)^2 - \frac{(\sum_{j \in J^t} (swsEx_j^t))^2}{n}}{n}$
QoC Variance Price	$varCPr_U^t$	$\frac{\sum_{j \in J^t} (swsPr_j^t)^2 - \frac{(\sum_{j \in J^t} (swsPr_j^t))^2}{n}}{n}$

6.2.7 QoC - Summary

We summarize the QoC for the 5 QoC properties (minus variance) in Table 6.3.

Table 6.3. List of 5 QoC properties for the Community

Property	**Formula**
QoC Availability	$cAv_U^t = \frac{\sum_{j \in J^t} swsAv_j^t}{\lvert J^t \rvert}$
QoC Reliability	$cRl_U^t = \frac{\sum_{j \in J^t} swsRl_j^t}{\lvert J^t \rvert}$
QoC Capacity	$cCp_U^t = \sum_{j \in J^t} swsCp_j^t$
QoC Execution Duration	$cEx_U^t = \frac{\sum_{j \in J^t} swsEx_j^t}{\lvert J^t \rvert}$
QoC Price	$cPr_U^t = \frac{\sum_{j \in J^t} swsPr_j^t}{\lvert J^t \rvert}$

6.3 Admission

We have introduced a mechanism in order to predict the requirements of clients and introduced a mechanism in order to evaluate the QoS of a community as a whole. In this section, we introduce a mechanism to decide which Web service(s) to admit into the community. When the community decides that additional Web services are needed to be admitted into the community, the community selects from a set of potential Web services. This set of Web services include Web services which the community found from the service registry as well as Web services that have independently applied to join the community. We first present notations in Section 6.3.1. The admission mechanism is based on perspective of 3 parties:

- Client Satisfaction (Section 6.3.2) - We consider whether the admission of Web services into the community can help satisfy the predicted future requirements of the client. We compare the QoS offered by the current SWSs in the community and propose a mechanism on how many and which Web services should be admitted into the community.
- Web services Satisfaction (Section 6.3.3) - We look at what effects the admission of additional Web services can have on the SWSs already in the community as well as to the potential Web services joining the community. A mechanism is proposed on how best to admit Web services based on the perspective of the Web services.
- Community Satisfaction (Section 6.3.4) - We look at the admission of Web services from the community's perspective.

The mechanism for each perspective gives a score which allows the community to deterministically decide which Web service(s) to admit into the community. We combine the 3 scores in Section 6.3.5.

6.3.1 Projected Satisfaction - Preliminary

In this section we define some of the notations used in the mechanism on admitting Web services into the community. We start with defining the set of potential Web services at time t as W^t, where each individual such Web services is denoted as w, where $w \in W^t$. Each potential Web service has the QoS properties defined in Table 6.4.

In order to systematically consider all potential Web services, we define $P(W^t)$ as the set of all subsets of W^t. We note Binomial coefficients state that $|P(W^t)| = 2^{W^t}$. We also define an element in $P(W^t)$ to be S ($S \in P(W^t)$)

Table 6.4. List of QoS properties for potential Web services

Property	**Notation**
Web service Capacity	$wsCp_w^t$
Web service Availability	$wsAv_w^t$
Web service Reliability	$wsRl_w^t$
Web service Execution Duration	$wsEx_w^t$
Web service Price	$wsPr_w^t$

and $|S|$ to be k. S therefore refers to a set of Web services. Each set of Web services, S is considered to have the QoS values defined in Table 6.5

Table 6.5. List of QoS properties for a set of Web services

Property	**Notation**	**Formula**
Set Capacity	$setCp_S^t$	$\sum_{w \in S} wsCp_w^t$
Set Availability	$setAv_S^t$	$\frac{\sum_{w \in S} wsAv_w^t}{k}$
Set Reliability	$setRl_S^t$	$\frac{\sum_{w \in S} wsRl_w^t}{k}$
Set Execution Duration	$setEx_S^t$	$\frac{\sum_{w \in S} wsEx_w^t}{k}$
Set Price	$setPr_S^t$	$\frac{\sum_{w \in S} wsPr_w^t}{k}$

6.3.2 Projected Client Satisfaction

In this section we look at the projected client satisfaction if a set is accepted into the community. This is the satisfaction of the client based on current community QoC properties and predicted future client requirements. There are 5 components that make the projected client satisfaction, and we base it on the QoS properties - Capacity, Availability, Reliability, Execution Duration and Price.

6.3.2.1 Projected Client Satisfaction - Capacity

Although there is no way for the community to be completely sure of the number of requests that it might receive in the future, we assume that the community admits Web services according to the predicted value. We take into account communities which prefer to over and under compensate for the difference in predicted capacity requirements and the current capacity. We call this value the community capacity target modifier, $cCpTargetMod_U^t$, and this is a value starting from 0. Note that this value can exceed 1.0 if the community feels that additional capacity is required in order for the community to feel comfortable. The value is ultimately decided by the system administrator designing the community. For community, U, and set of clients, C, the target capacity, tCp_{C-U}^t, at time t is defined in Equation 6.17.

$$tCp_{C-U}^t = cCpTargetMod_U^t \cdot pCR_C^t \tag{6.17}$$

We now define the projected client capacity satisfaction based on the number of requests by admitting a set of Web services S into the community, $projClientSatisCp_{C-S-U}^t$, in Equation 6.18.

$$projClientSatisCp_{C-S-U}^t = \begin{cases} \frac{setCp_S^t + cCp_U^t}{tCp_{C-U}^t} & \text{if } (setCp_S^t + cCp_U^t) \leq tCp_{C-U}^t \\ 1 & \text{if } (setCp_S^t + cCp_U^t) > tCp_{C-U}^t \end{cases} \tag{6.18}$$

6.3.2.2 Projected Client Satisfaction - Availability

The community may want to admit Web services in order to handle a different availability than the predicted availability. One reason might be in order for the community to have a buffer in case the client eventually submits a higher requested availability than what is predicted. Another reason might also be that the community would want to take a risk and require a lower availability than what was predicted. We introduce the community availability target modifier, $cAvTargetMod_U^t$ for community U at time t. This value starts from 0 and can exceed 1.0 if the community feels that additional availability is required in order for the community to feel comfortable. The eventual target availability, however, cannot exceed 1.0. The target availability, tAv_{C-U}^t, at time t is defined in Equation 6.19.

$$tAv_{C-U}^t = cAvTargetMod_U^t \cdot pAv_C^t \tag{6.19}$$

Where tAv_{C-U}^t has a maximum value of 1.0.

We define the projected client availability satisfaction based on the availability by admitting set of Web services S into the community, $projClientSatisAv_{C-S-U}^t$, in Equation 6.20.

$$projClientSatisAv_{C-S-U}^t = \begin{cases} \frac{projAv_{S-U}^t}{tAv_{C-U}^t} & \text{if } (projAv_{S-U}^t) \leq tAv_{C-U}^t \\ 1 & \text{if } (projAv_{S-U}^t) > tAv_{C-U}^t \end{cases} \tag{6.20}$$

Where the projected availability, $projAv_{S-U}^t = \frac{setAv_S^t \cdot |S| + cAv_U^t \cdot |J^t|}{|S| + |J^t|}$.

6.3.2.3 Projected Client Satisfaction - Reliability

The community may want to admit Web services in order to handle a different reliability than the predicted reliability. One reason might be in order for the community to have a buffer in case the client eventually submits a higher requested reliability than what is predicted. Another reason might also be that the community would want to take a risk and require a lower reliability than what was predicted. We introduce the community reliability target modifier, $cRlTargetMod_U^t$ for community U at time t. This value starts from 0 and can exceed 1.0 if the community feels that additional reliability is required in order for the community to feel comfortable. The eventual target reliability, however, cannot exceed 1.0. The target reliability, tRl_{C-U}^t, at time t is defined in Equation 6.21.

$$tRl^t_{C-U} = cRlTargetMod^t_U \cdot pRl^t_C \tag{6.21}$$

Where tRl^t_{C-U} has a maximum value of 1.0.

We now define the projected client reliability satisfaction based on the reliability by admitting set of Web services S into the community, $projClientSatisRl^t_{C-S-U}$, in Equation 6.22.

$$projClientSatisRl^t_{C-S-U} = \begin{cases} \frac{projRl^t_{S-U}}{tRl^t_{C-U}} & \text{if } (projRl^t_{S-U}) \leq tRl^t_{C-U} \\ 1 & \text{if } (projRl^t_{S-U}) > tRl^t_{C-U} \end{cases} \tag{6.22}$$

Where the projected reliability, $projRl^t_{S-U} = \frac{setRl^t_S \cdot |S| + cRl^t_U \cdot |J^t|}{|S| + |J^t|}$.

6.3.2.4 Projected Client Satisfaction - Execution Duration

The community may want to admit Web services in order to handle a different execution duration than the predicted execution duration. One reason might be in order for the community to have a buffer in case the client eventually submits a lower requested execution duration than what is predicted. Another reason might also be that the community would want to take a risk and require a higher execution duration than what was predicted. We introduce the community execution duration target modifier, $cExTargetMod^t_U$ for community U at time t. This value starts from 0 and can exceed 1.0 if the community feels that it wants a higher execution duration. The target execution duration, tEx^t_{C-U}, at time t is defined in Equation 6.23.

$$tEx^t_{C-U} = cExTargetMod^t_U \cdot pEx^t_C \tag{6.23}$$

We now define the projected client execution duration satisfaction based on the execution duration by admitting set of Web services S into the community, $projClientSatisEx^t_{C-S-U}$, in Equation 6.24.

$$projClientSatisEx^t_{C-S-U} = \begin{cases} \frac{projEx^t_{S-U}}{tEx^t_{C-U}} & \text{if } (projEx^t_{S-U}) \leq tEx^t_{C-U} \\ 1 & \text{if } (projEx^t_{S-U}) > tEx^t_{C-U} \end{cases} \tag{6.24}$$

Where the projected execution duration, $projEx^t_{S-U} = \frac{setEx^t_S \cdot |S| + cEx^t_U \cdot |J^t|}{|S|+|J^t|}$.

6.3.2.5 Projected Client Satisfaction - Price

The community may want to admit Web services in order to handle a different price than the predicted price. One reason might be in order for the community to have a buffer in case the client eventually submits a lower requested price than what is predicted. Another reason might also be that the community would want to take a risk and require a higher price than what was predicted. We introduce the community price target modifier, $cPrTargetMod^t_U$ for community U at time t. This value starts from 0 and can exceed 1.0 if the community feels that it wants a higher price. The target price, tPr^t_{C-U}, at time t is defined in Equation 6.25.

$$tPr^t_{C-U} = cPrTargetMod^t_U \cdot pPr^t_C \tag{6.25}$$

We now define the projected client price satisfaction based on the price by admitting set of Web services S into the community, $projClientSatisPr^t_{C-S-U}$, in Equation 6.26.

$$projClientSatisPr^t_{C-S-U} = \begin{cases} \frac{projPr^t_{S-U}}{tPr^t_{C-U}} & \text{if } (projPr^t_{S-U}) \leq tPr^t_{C-U} \\ 1 & \text{if } (projPr^t_{S-U}) > tPr^t_{C-U} \end{cases} \tag{6.26}$$

Where the projected price, $projPr^t_{S-U} = \frac{setPr^t_S \cdot |S| + cPr^t_U \cdot |J^t|}{|S|+|J^t|}$.

6.3.2.6 Projected Client Satisfaction - Summary and Total Score

We summarize the projected client satisfaction components in Table 6.6. The total projected client satisfaction score is dependent on the values of these properties.

Table 6.6. Projected Admission Client Satisfaction Properties

Property	**Formula**
Availability	$projClientSatisAv^t_{C-S-U} = \begin{cases} \frac{projAv^t_{S-U}}{tAv^t_{C-U}} & \text{if } (projAv^t_{S-U}) \leq tAv^t_{C-U} \\ 1 & \text{if } (projAv^t_{S-U}) > tAv^t_{C-U} \end{cases}$
Reliability	$projClientSatisRl^t_{C-S-U} = \begin{cases} \frac{projRl^t_{S-U}}{tRl^t_{C-U}} & \text{if } (projRl^t_{S-U}) \leq tRl^t_{C-U} \\ 1 & \text{if } (projRl^t_{S-U}) > tRl^t_{C-U} \end{cases}$
Capacity	$projClientSatisCp^t_{C-S-U} = \begin{cases} \frac{setCp^t_S + cCp^t_U}{tCp^t_{C-U}} & \text{if } (setCp^t_S + cCp^t_U) \leq tCp^t_{C-U} \\ 1 & \text{if } (setCp^t_S + cCp^t_U) > tCp^t_{C-U} \end{cases}$
Execution Duration	$projClientSatisEx^t_{C-S-U} = \begin{cases} \frac{projEx^t_{S-U}}{tEx^t_{C-U}} & \text{if } (projEx^t_{S-U}) \leq tEx^t_{C-U} \\ 1 & \text{if } (projEx^t_{S-U}) > tEx^t_{C-U} \end{cases}$
Price	$projClientSatisPr^t_{C-S-U} = \begin{cases} \frac{projPr^t_{S-U}}{tPr^t_{C-U}} & \text{if } (projPr^t_{S-U}) \leq tPr^t_{C-U} \\ 1 & \text{if } (projPr^t_{S-U}) > tPr^t_{C-U} \end{cases}$

By giving each projected client satisfaction component a weight, we can now define the total projected client satisfaction score. We define the projected client satisfaction score in Equation 6.27.

$$\begin{aligned} projClientSatis^t_{C-S-U} &= w_{projCp} \cdot projClientSatisCp^t_{C-S-U} \\ &+ w_{projAv} \cdot projClientSatisAv^t_{C-S-U} \\ &+ w_{projRl} \cdot projClientSatisRl^t_{C-S-U} \\ &+ w_{projEx} \cdot projClientSatisEx^t_{C-S-U} \\ &+ w_{projPr} \cdot projClientSatisPr^t_{C-S-U} \end{aligned} \tag{6.27}$$

Where w_{projCp}, w_{projAv}, w_{projRl}, w_{projEx}, and w_{projPr} are the weights of the projected client capacity satisfaction, projected client availability satisfaction, projected client reliability satisfaction, projected client execution duration satisfaction, and projected client price satisfaction respectively. The exact values of these weights are determined by the system administrator of the community. This is intended in order to give the system administrator the flexibility to choose which properties are more important.

6.3.3 Projected Web Services Satisfaction

In this section, we define the projected satisfaction of SWSs for each potential set of Web services, S, that is admitted into the community. Since each Web service joins a community expecting a certain level of work, the satisfaction of SWSs in a community is based on its workload. We define the workload as the proportion of requests that the SWS is asked to handle and the capacity of the SWS. Before presenting the projected SWS Satisfaction, we define the target workload, tWl_U^t. The target workload is defined as the proportion of predicted number of requests against the potential total capacity of admitted Web services and the current capacity of the community. The target workload is defined by the system administrator of each community, and carries a value of between 0 and 1. The target workload has a maximum value of 1 because it makes no sense for the community to target the situation where there are more requests than the community can handle, the community would have to turn requests away.

The projected satisfaction can be calculated by comparing the predicted number of requests with the total capacities of the community and set of Web services, S. We define the projected Web service satisfaction of community, U, with set S at time t in Equation 6.28.

$$projSWSSatis_{C-S-U}^t = \begin{cases} \frac{pCR_C^t}{setCp_S^t + cCp_U^t} & \text{if } \frac{pCR_C^t}{setCp_S^t + cCp_U^t} \leq tWl_U^t \\ 1 & \text{if } \frac{pCR_C^t}{setCp_S^t + cCp_U^t} > tWl_U^t \end{cases} \tag{6.28}$$

6.3.4 Projected Community Satisfaction

In this section, we define the projected satisfaction of the community for each potential set of Web services, S, that is admitted into the community. We consider the community satisfaction to be based on the income of the overall community. This is based on the predicted number of requests, the QoC-price, as well as the price and capacities of all possible SWSs.

There are two main parts to calculate the projected satisfaction of the community. The first part consists of the minimum possible average price of all SWSs in the community. We define the minimum possible average price, $minPr^t_J$ as the lowest pCR^t_C for all SWSs. The minimal possible average price is calculated by executing algorithm 1.

In this algorithm, we assume that the predicted number of requests is always lower than the total capacities of all possible SWSs.

Algorithm 1 Minimum Possible Price

```
$minPr^t_J \leftarrow 0$
$i \leftarrow 0$
for all SWS $j \in J$ sorted by increasing price do
    $capacitySWS \leftarrow 0$
    while $capacitySWS \leq swsCp^t_j$ AND $(i < pCR^t_C)$ do
        $minPr^t_J \leftarrow minPr^t_J + swsPr^t_j$
        $i \leftarrow i + 1$
        if $i = pCR^t_C$ then Exit Forloop
        end if
        $capacitySWS \leftarrow capacitySWS + 1$
    end while
end for
$minPr^t_J \leftarrow minPr^t_J / i$
```

The second part consists of the projected community price, $projCommPr^t_{S-U}$ defined in Equation 6.29.

$$projCommPr^t_{S-U} = \frac{setPr^t_S \cdot setCp^t_S + cPr^t_U \cdot cCp^t_U}{setCp^t_S + cCp^t_U} \tag{6.29}$$

We can now define the projected community satisfaction of community, U, with set, S, at time ,t, in Equation 6.30.

$$projCommSatis^t_{C-S-U} = \frac{minPr^t_J}{projCommPr^t_{S-U}} \tag{6.30}$$

The community would always want to maximize the projected community satisfaction since it represents the highest amount of revenue for the community. The community satisfaction has a maximum value of 1.0.

6.3.5 WS Admission - Total Score

We can now define the projected satisfaction for set S by combining the projected client satisfaction, the projected Web services satisfaction and the projected community satisfaction. We define the total projected satisfaction for set S in Equation 6.31.

$$\begin{aligned} totalProjSatisC-S-U^t = {} & w_{projClient} \cdot projClientSatis^t_{C-S-U} \\ & + w_{projSWS} \cdot projSWSSatis^t_{C-S-U} \\ & + w_{projComm} \cdot projCommSatis^t_{C-S-U} \end{aligned} \tag{6.31}$$

Where $w_{projClient}$, $w_{projSWS}$ and $w_{projComm}$ are the weights for the projected client satisfaction, the projected SWS satisfaction and the projected community satisfaction respectively. The exact values of these weights are determined by the system administrator of the community. This is intended in order to give the system administrator the flexibility to choose which properties are more important. For example, if the system administrator would like to give more emphasis to the projected client satisfaction, it can give a higher value to $w_{projClient}$ relative to $w_{projSWS}$ and $w_{projComm}$.

6.4 Web service perspective

The goal of a Web service is to handle as many requests as it can handle (limited by its capacity), this maximizes the amount of revenue that the Web services receives since it receives an amount of currency for each request that it handles. It has the option of providing the service on its own, or joining a community in the hopes that the community will provide more requests to it. We assume that when the Web service receives an invitation from the community, it is guaranteed a spot in the community if the Web service decides to join the community. It is thus up to the Web service to evaluate whether it is better off being alone, or to join the community. We explained the supplier-side SLA and the model of a community earlier in Section 4.5.

In order to make this decision, the Web service has to estimate its workload if it were on its own, we notate this as $esWl^t_w$ for Web service, w, at time t.

Therefore, for Web service w, it will join the community if $gWl^t_j > esWl^t_w$. Otherwise it will prefer to be alone.

In the event that the guaranteed workload is not provided by the community in the SLA, the Web service has to make an estimate on whether it is better off being alone or to join the community. This decision could also be based on the community's reputation.

A community's reputation can be retrieved through a central repository, this could an additional service which the service registry has. How a community's reputation is calculated by the service registry is beyond the scope of the paper, but it can be assumed that the reputation of a community is proportional to the satisfaction of the clients and Web services. The Web services that are evaluating the reputation of the community will have to be already existing in the community or former SWSs of the community. Web services that are considering whether or not to join the community can use this reputation to decide whether or not to join the community.

6.5 Expulsion of SWSs

One reason why a SWS might be expelled from a community might be its inability to sustain its QoS at its promised levels. Such an expulsion is straightforward in the sense that the community does not need to choose which SWS to expel from the community. The community is able to expel the SWS in accordance with the terms and conditions set in the SLA.

Another reason for the expulsion of SWSs from the community is in order for the community to reach the targeted QoS values. Any expulsion of SWSs from the community has to be compliant with the SLA (See Section 4.5)which the community and SWS agreed to. We use the set notation that was described earlier in Table 6.5. However, in the case of expulsion, the set of Web services refers to a set of SWSs that already exist in the community. If we denote $P(J^t)$ as the set of all subsets of J^t, the set S refers to an element of $P(J^t)$.

We can now define the projected client satisfaction, the projected SWS satisfaction and the projected community satisfaction for each set of SWSs. In Section 6.5.1 we present our proposed formulas for the projected client satisfaction. Next in Section 6.5.2 we present our proposed formulas for the projected SWS satisfaction. In Section 6.5.3 we present our proposed formulas for the projected MWS satisfaction. Finally we present the projected expulsion satisfaction score in Section 6.5.4.

6.5.1 Projected Client Satisfaction - Expulsion

We follow the same reasoning when calculating the projected expulsion client satisfaction for the set of potential web services when calculating the projected client satisfaction for a set of existing SWSs. We first present the projected expulsion client satisfaction based on the number of requests in Section 6.5.1.1. Next in Section 6.5.1.2, we present the projected expulsion client satisfaction based on availability. In Section 6.5.1.3, we present the projected expulsion client satisfaction based on reliability. In Section 6.5.1.4, we present the projected expulsion client satisfaction based on execution duration. In Section 6.5.1.5, we present the projected expulsion client satisfaction based on price. Finally in Section 6.5.1.6, we summarize the projected expulsion client satisfaction properties and present the total projected expulsion client satisfaction score.

6.5.1.1 Projected Client Capacity Satisfaction - Expulsion

We define the projected expulsion client satisfaction based on the number of requests by considering the set of SWSs S for a community U at time t in Equation 6.32

$$projExClientSatisCp^t_{C-S-U} = \begin{cases} \frac{setCp^t_S}{tCp^t_{C-U}} & \text{if } setCp^t_S \leq tCp^t_{C-U} \\ 1 & \text{if } setCp^t_S > tCp^t_{C-U} \end{cases} \tag{6.32}$$

6.5.1.2 Projected Client Availability Satisfaction - Expulsion

We define the projected expulsion client satisfaction based on the availability by considering the set of SWSs S for a community U at time t in Equation 6.33

$$projExClientSatisAv^t_{C-S-U} = \begin{cases} \frac{setAv^t_S}{tAv^t_{C-U}} & \text{if } (setAv^t_S) \leq tAv^t_{C-U} \\ 1 & \text{if } (setAv^t_S) > tAv^t_{C-U} \end{cases} \tag{6.33}$$

6.5.1.3 Projected Client Reliability Satisfaction - Expulsion

We define the projected expulsion client satisfaction based on the reliability by considering the set of SWSs S for a community U at time t in Equation 6.34

$$projExClientSatisRl^t_{C-S-U} = \begin{cases} \frac{setRl^t_S}{tRl^t_{C-U}} & \text{if } (setRl^t_S) \leq tRl^t_{C-U} \\ 1 & \text{if } (setRl^t_S) > tRl^t_{C-U} \end{cases} \tag{6.34}$$

6.5.1.4 Projected Client Execution Duration Satisfaction - Expulsion

We define the projected expulsion client satisfaction based on the execution duration by considering the set of SWSs S for a community U at time t in Equation 6.35

$$projExClientSatisEx^t_{C-S-U} = \begin{cases} \frac{setEx^t_S}{tEx^t_{C-U}} & \text{if } (setEx^t_S) \leq tEx^t_{C-U} \\ 1 & \text{if } (setEx^t_S) > tEx^t_{C-U} \end{cases} \tag{6.35}$$

6.5.1.5 Projected Client Price Satisfaction - Expulsion

We define the projected expulsion client satisfaction based on the price by considering the set of SWSs S for a community U at time t in Equation 6.36

$$projExClientSatisPr^t_{C-S-U} = \begin{cases} \frac{setPr^t_S}{tPr^t_{C-U}} & \text{if } (setPr^t_S) \leq tPr^t_{C-U} \\ 1 & \text{if } (setPr^t_S) > tPr^t_{C-U} \end{cases} \tag{6.36}$$

6.5.1.6 Projected Client Satisfaction - Summary and Total Score

We summarize the projected client satisfaction components in Table 6.7. The total score of the projected expulsion client satisfaction is dependent on the values of these properties.

Table 6.7. Projected Expulsion Client Satisfaction Properties

Property	**Notation**
Availability	$projExClientSatisAv^t_{C-S-U} = \begin{cases} \frac{setAv^t_S}{tAv^t_{C-U}} & \text{if } (setAv^t_S) \leq tAv^t_{C-U} \\ 1 & \text{if } (setAv^t_S) > tAv^t_{C-U} \end{cases}$
Reliability	$projExClientSatisRl^t_{C-S-U} = \begin{cases} \frac{setRl^t_S}{tRl^t_{C-U}} & \text{if } (setRl^t_S) \leq tRl^t_{C-U} \\ 1 & \text{if } (setRl^t_S) > tRl^t_{C-U} \end{cases}$
Capacity	$projExClientSatisCp^t_{C-S-U} = \begin{cases} \frac{setCp^t_S}{tCp^t_{C-U}} & \text{if } setCp^t_S \leq tCp^t_{C-U} \\ 1 & \text{if } setCp^t_S > tCp^t_{C-U} \end{cases}$
Execution Duration	$projExClientSatisEx^t_{C-S-U} = \begin{cases} \frac{setEx^t_S}{tEx^t_{C-U}} & \text{if } (setEx^t_S) \leq tEx^t_{C-U} \\ 1 & \text{if } (setEx^t_S) > tEx^t_{C-U} \end{cases}$
Price	$projExClientSatisPr^t_{C-S-U} = \begin{cases} \frac{setPr^t_S}{tPr^t_{C-U}} & \text{if } (setPr^t_S) \leq tPr^t_{C-U} \\ 1 & \text{if } (setPr^t_S) > tPr^t_{C-U} \end{cases}$

By giving each projected expulsion client satisfaction component a weight, we can now define the total projected expulsion client satisfaction score. We define the projected client satisfaction score in Equation 6.37.

$$\begin{aligned} projExClientSatis^{t}_{C-S-U} = {} & w_{projExCp} \cdot projExClientSatisCp^{t}_{C-S-U} \\ & + w_{projExAv} \cdot projExClientSatisAv^{t}_{C-S-U} \\ & + w_{projExRl} \cdot projExClientSatisRl^{t}_{C-S-U} \\ & + w_{projExEx} \cdot projExClientSatisEx^{t}_{C-S-U} \\ & + w_{projExPr} \cdot projExClientSatisPr^{t}_{C-S-U} \end{aligned} \tag{6.37}$$

Where $w_{projExCp}$, $w_{projExAv}$, $w_{projExRl}$, $w_{projExEx}$, and $w_{projExPr}$ are the weights of the projected client capacity satisfaction, projected client availability satisfaction, projected client reliability satisfaction, projected client execution duration satisfaction, and projected client price satisfaction for expulsion respectively. The exact values of these weights are determined by the system administrator of the community. This is intended in order to give the system administrator the flexibility to choose which properties are more important. For example, if the system administrator would like to give more emphasis to the projected expulsion client satisfaction for Availability, it could give a higher weight value towards $w_{projExAv}$ relative to the other 4 weights.

6.5.2 Projected SWS Satisfaction - Expulsion

The projected expulsion satisfaction can be calculated by comparing the predicted number of requests with the capacities of the set of SWSs, S. We define the projected Web service satisfaction of community, U, with set S at time t in Equation 6.38.

$$projExSWSSatis^{t}_{C-S-U} = \begin{cases} \frac{pCR^{t}_{C}}{setCp^{t}_{S}} & \text{if } \frac{pCR^{t}_{C}}{setCp^{t}_{S}} \leq tWl^{t}_{U} \\ 1 & \text{if } \frac{pCR^{t}_{C}}{setCp^{t}_{S}} > tWl^{t}_{U} \end{cases} \tag{6.38}$$

6.5.3 Projected Community Satisfaction - Expulsion

We define the projected community satisfaction due to expulsion similar to the projected community satisfaction in admission defined in Section 6.3.4. We can now define the projected community satisfaction of community, U, with set, S, at time, t, in Equation 6.39.

$$projExCommSatis^{t}_{C-S-U} = \frac{minPr^{t}_{J}}{projExCommPr^{t}_{S-U}} \tag{6.39}$$

Where the projected Community price due to expulsion, $projExCommPr^{t}_{S-U}$ is defined in Equation 6.40.

$$projExCommPr^{t}_{S-U} = \frac{setPr^{t}_{S} \cdot setCp^{t}_{S} + cPr^{t}_{U} \cdot cCp^{t}_{U}}{setCp^{t}_{S} + cCp^{t}_{U}} \tag{6.40}$$

The community would always want to maximize the projected community satisfaction since it represents the highest amount of revenue for the community. The community satisfaction due to expulstion has a maximum value of 1.0.

6.5.4 WS Expulsion - Total Score

We can now define the projected expulsion satisfaction score for set S by combining the projected expulsion client satisfaction, the projected expulsion Web services satisfaction and the projected expulsion community satisfaction. We define the total projected expulsion satisfaction score for set S in Equation 6.41.

$$\begin{aligned} totalProjExSatis^{t}_{C-S-U} &= w_{projExClient} \cdot projExClientSatis^{t}_{C-S-U} \\ &+ w_{projExSWS} \cdot projExSWSSatis^{t}_{C-S-U} \\ &+ w_{projExComm} \cdot projExCommSatis^{t}_{C-S-U} \end{aligned} \tag{6.41}$$

Where $w_{projExClient}$, $w_{projExSWS}$ and $w_{projExComm}$ are the weights for the projected client satisfaction, the projected SWS satisfaction and the projected community satisfaction for expulsion respectively. The exact values of these weights are determined by the system administrator of the community. This is intended in order to give the system administrator the flexibility to choose which properties are more important.

Using the total projected expulsion satisfaction for each set, the community can decide which SWSs to keep in the community and expel the rest. The set of SWSs that returns the highest score should remain in the community while the MWS expels the rest. The community has to keep in mind that it has to fulfil the terms and conditions stated in the SLA when expelling SWSs from the community.

6.6 Experiments

In this section we demonstrate the mechanisms presented in this chapter by conducting experiments. The goal of the experiments twofold:

- To illustrate the different approaches a community may undertake in its admission and expulsion in order to better sustain the quality of Web services
- Compare these approaches with the situation where a community does not admit or expel Web services

Our measurable properties include the number of handled requests as well as the satisfaction of the client, SWSs and MWS. We assume that the sustainability of these values is a measurement on the community's ability to sustain the quality of Web services.

To be thorough for these experiments, we consider different types of clients. We consider a normal client to behave normally if it there is a minimal predictable change in the number of requests and the QoS requirements of each request that is sent to the community. The amount of change is minimal and is assumed to be small enough such that a normal community is given enough time to make the changes in order to handle the requests at a similar level from earlier. The exact amount of change of what defines *normal* is subjective, relative, and out of the scope of this paper.

Other than this normal client, in our experiments, we consider 4 other types.

- Request count sudden decrease - this type of client behaves normally for a period of time before suddenly decreasing the number of requests it sends to the community The sudden decrease only occurs for one time unit.
- Request count sudden increase - this type of client behaves normally for a period of time before suddenly increasing the number of requests it sends to the community. The sudden increase only occurs for one time unit.
- Request QoS sudden decrease - this type of client behaves normally for a period of time before suddenly decreasing the Qos requirements of the requests it sends to the community. The sudden decrease only occurs for one time unit.
- Request QoS sudden increase - this type of client behaves normally for a period of time before suddenly increasing the Qos requirements of the requests it sends to the community. The sudden increase only occurs for one time unit.

From the community's perspective, we consider 3 types of communities:

- Conservative - this community sets the QoS target modifiers in order to be able handle a big change in the projected QoS requirements. In our experiments, we set this to a 20% change depending on which property it is. This affects the target capacity of the community as well.
- Normal - this community sets the QoS target modifiers in order to be able handle a normal change in the projected QoS requirements. In our

experiments, we set this to a 10% change depending on which property it is. This affects the target capacity of the community as well.

- Risky - this community sets the QoS target modifiers in order to be able handle a small change in the projected QoS requirements. In our experiments, we set this to a 0% change depending on which property it is. This affects the target capacity of the community as well.

It is assumed that in general, a service provider that is able to handle a higher QoS is going to charge more. Thus a risky community takes the gamble that there is little change in the QoS requirements of future requirements and in return can potentially get more profit.

Our experiments involve 15 different simulations by comparing each different client type (5 types) to each different community type (3 types). We do these experiments for both the admission and expulsion approaches. The proposed admissions and expulsion approaches are compared to the approach where nothing is done (no admission nor expulsion).

The increase in QoS requirements from the clients demonstrates a community's ability to admit service providers, while the decrease in QoS requirements from the clients demonstrates a community's ability to expel service providers.

In Section 6.6.1 we first look at these measurable properties for the normal client and normal community and observe what is happening. Next in Section 6.6.2, we demonstrate the benefits of our approach by including a sudden change in QoS requirements after a period of time. In Section 6.6.3, we demonstrate the benefits of our approach by including a sudden change in number of requests from the client after a period of time. Finally in Section 6.6.4 we conclude the member management experiment section.

6.6.1 Normal Client - Normal Community

In these experiments we simulate 50 SWSs over 10 time units. At the end of each time unit, if the client satisfaction was sufficient (80%) the number of

Table 6.8. Member Management Experiment Properties

Member Management Experiment Properties	**Mean**	**Min**	**Max**
SWS Price	12	10	14
SWS Failure Probability	10	0	20
Requested Budget	15	10	20
Requested Availability	57.5	30	85
Requested Reliability	52.5	20	85
Requested Execution Duration	4250	3500	5000

requests per unit time was increased by 10% while if the client satisfaction was low enough (20%), then the number of requests per time unit was decreased by 10%.

In Table 6.8, we list other values of the properties that were used in the experiment.

SWS Failure Probability refers to the probability (in percentage) of how likely it is for the SWS to fail. For the member management experiments, the fixed price auction was used in the community for the selection and substitution processes.

Our first experiment demonstrates the normal situation comparing the situation where there was no admission with the WS admission mechanisms that is proposed in Section 6.3. In our experiments we consider this admission mechanism in 3 forms - a risky, normal and conservative approach. These 3 forms differ in the value of the target modifier, we list the values of the target modifiers for all 3 approaches in Table 6.9. In a risky approach, the community prepares the least for potential changes from the client in both QoS and the number of requests, whereas in the conservative approach, the community prepares the most for potential such changes.

We look at the number of handled requests in Figure 6.1. We notice here that when the community is not allowed to admit any WSs into the community, the number of requests stays static. Whereas if the WS admission

Table 6.9. Target Modifier for the different admission types

Target Modifier	**Risky**	**Normal**	**Conservative**
$cCpTargetMod_U^t$	1.0	1.1	1.2
$cAvTargetMod_U^t$	1.0	1.1	1.2
$cRlTargetMod_U^t$	1.0	1.1	1.2
$cExTargetMod_U^t$	1.0	0.9	0.8
$cPrTargetMod_U^t$	1.0	0.9	0.8

mechanism was used, the community admits WSs into the community and is thus able to increase the number of requests that are handled. Furthermore, a conservative approach implies that the community prepares more for changes, and is thus able to handle more requests.

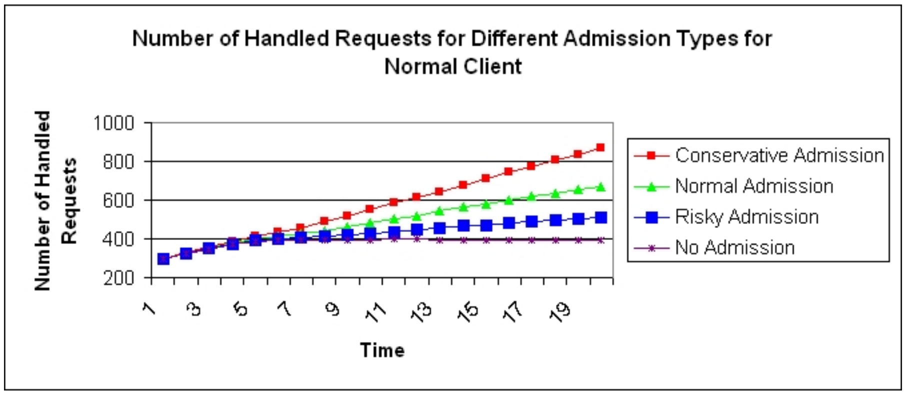
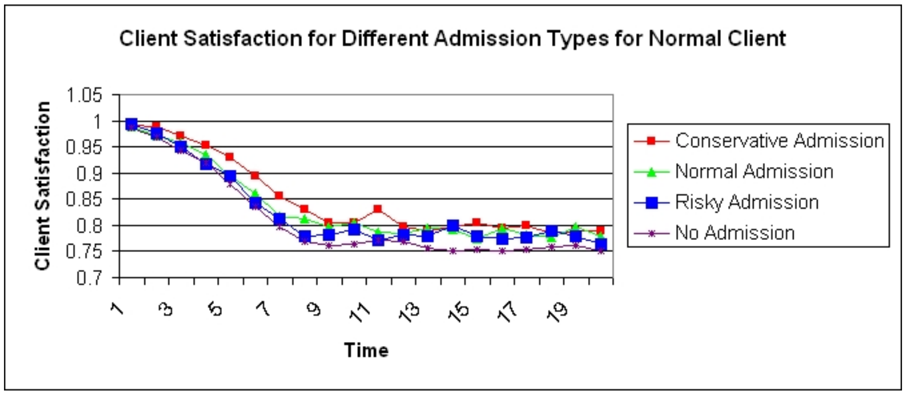
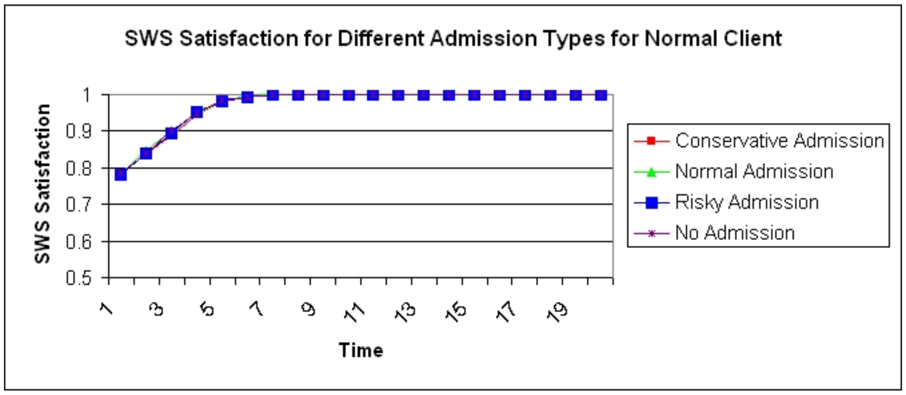


Fig. 6.1. Number of Handled Requests for WS Admission and No WS Admission

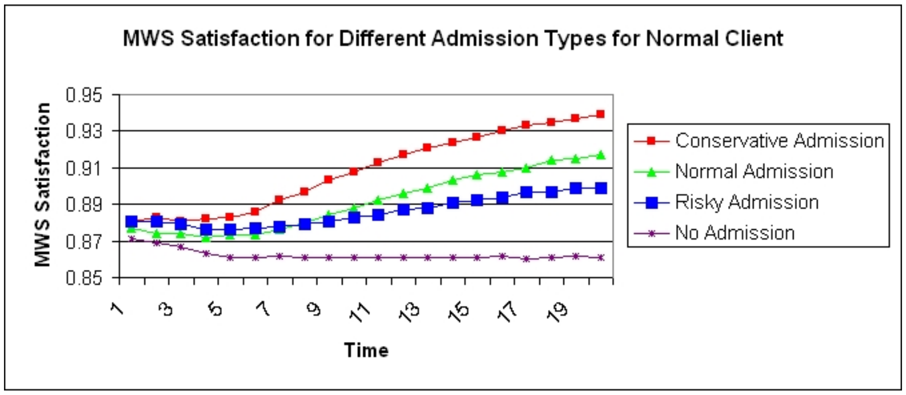
We now look at how the client satisfaction comparing the WS admission mechanism with a non-admission approach. Similarly, we see the 3 different WS admission mechanism types. We see this in Figure 6.2-(a). We notice that the client satisfaction is higher for a conservative approach, then the normal approach, then the risky approach. Finally, when the community does



(a) Client Satisfaction for WS Admission and No WS Admission



(b) SWS Satisfaction for WS Admission and No WS Admission



(c) MWS Satisfaction for WS Admission and No WS Admission

Fig. 6.2. Satisfactions for Normal Requests from the Client

not accept any new WSs into the community, the client satisfaction is at its lowest.

The SWS satisfaction is shown in Figure 6.2-(b), however we see here that because the number of requests outweigh the total capacities of SWSs in the community, the SWS satisfaction for all admission types approach and remain 1.0 after a period of time.

The MWS satisfaction depends on the price difference between the requested price and the price that is offered by the SWSs in the community. The bigger this disparity, the higher the MWS satisfaction. We observe the MWS in Figure 6.2-(c). We observe here that the MWS satisfaction increases from the non-admission approach, to the risky admission, to the normal admission, to the highest MWS using the conservative admission approach.

6.6.2 Sudden Change in Client QoS

If an unpredictable client changes the QoS requirements of its requests, the community may not be ready for the changes. In this section we observe the results of both changes where the QoS requirements are suddenly increased as well as where the QoS requirements are suddenly decreased. We first look at the number of handled request when there is a sudden increase in the QoS requirements in Figure 6.3. We notice that the conservative approach undertaken by the community allows the most number of requests to be handled by the community. When there are no admissions allowed, the number of handled requests is the least.

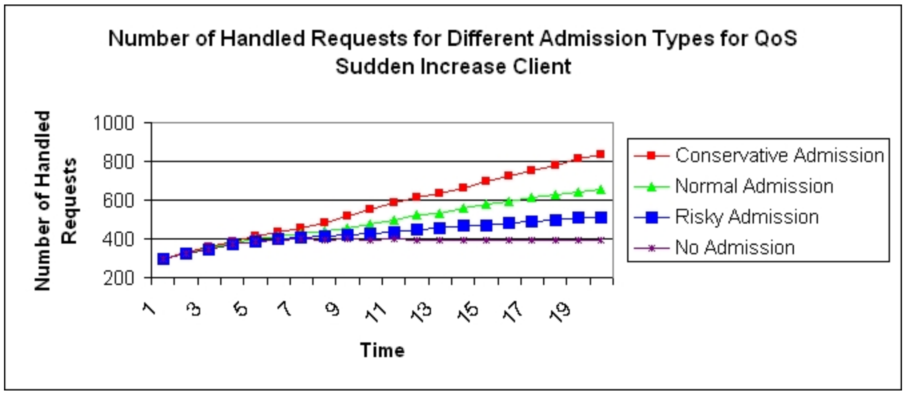
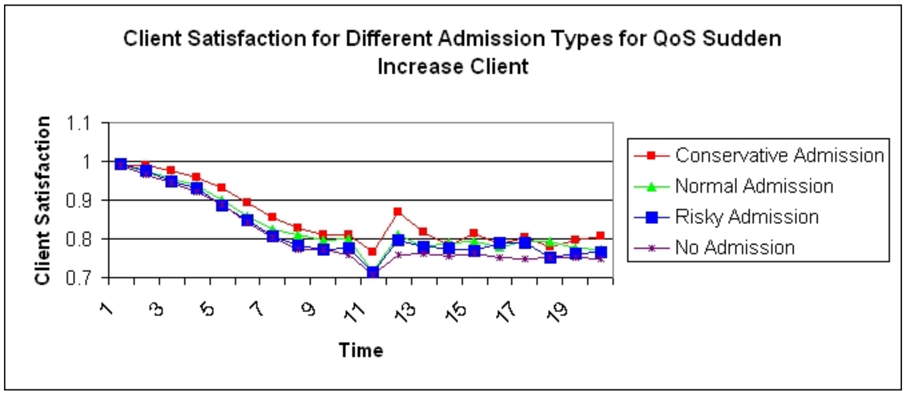


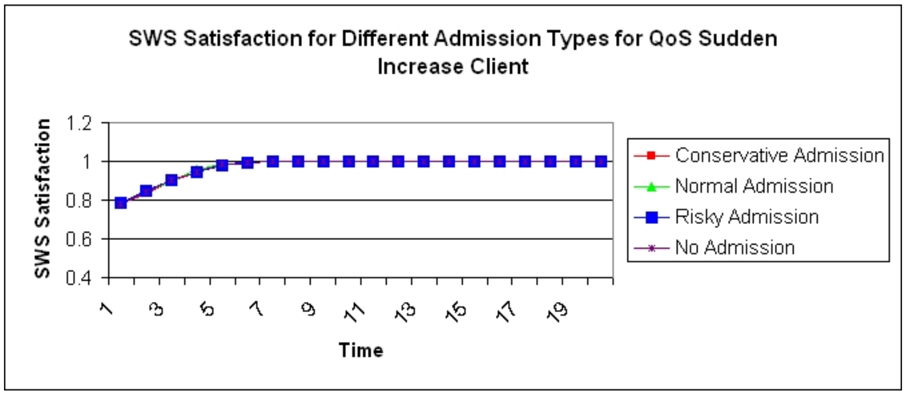
Fig. 6.3. Number of Handled Request for sudden increase in QoS requirements

Next we look at the satisfactions for a sudden increase in QoS requirements in Figure 6.4. We immediately notice the advantage of the conservative approach in Figure 6.4-(a). The drop in client satisfaction is the least if the community undertakes the conservative approach. When the QoS requirements increase suddenly, we notice that the client satisfaction for the normal and risky approaches yield the same amount of client satisfaction as the no admission approach when the sudden increase occurs. There is no significant change in the SWS and MWS satisfactions depicted in Figure 6.4-(b) and Figure 6.4-(c) respectively.

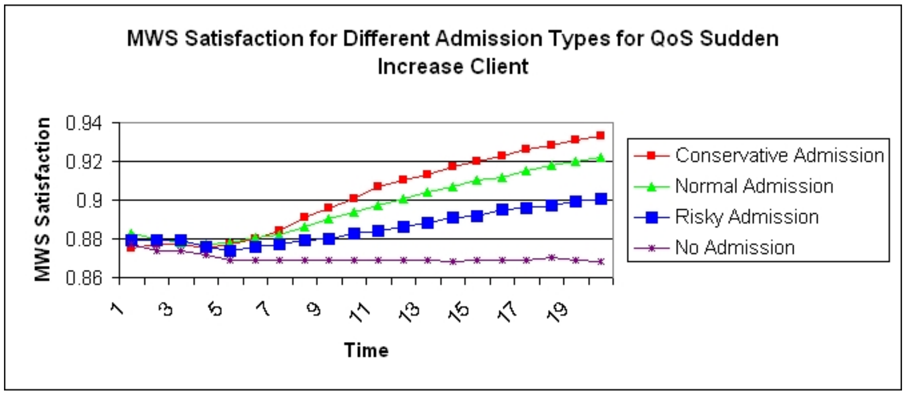
We now look at the number of handled request when there is a sudden decrease in the QoS requirements in Figure 6.5. We notice here that the sudden decrease in QoS requirements does not have much effect on the number of handled requests. The conservative approach still yields more requests being handled compared to the other admission approaches.



(a) Client Satisfaction for Sudden Increase in QoS Requirements



(b) SWS Satisfaction for Sudden Increase in QoS Requirements



(c) MWS Satisfaction for Sudden Increase in QoS Requirements

Fig. 6.4. Satisfactions for Sudden Increase in QoS Requirements

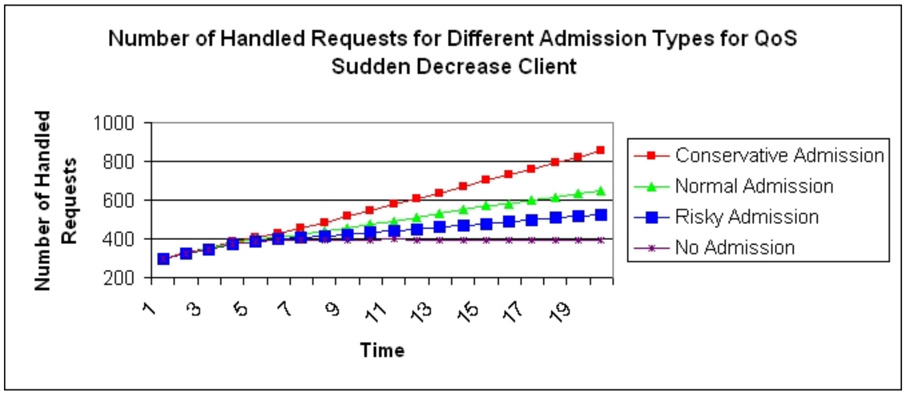


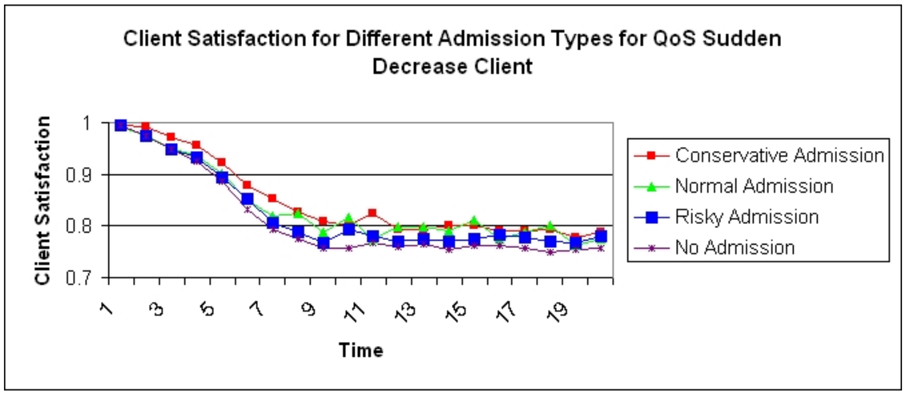
Fig. 6.5. Number of Handled Request for sudden decrease in QoS requirements

Next we look at the satisfactions for a sudden decrease in QoS requirements in figure 6.6. When there is a sudden decrease in the QoS requirements, the client satisfaction stays consistently high as seen in figure 6.6-(a). The SWS and MWS satisfactions in figure 6.6-(b) and figure 6.6-(c) respectively remain consistent with previous experiments.

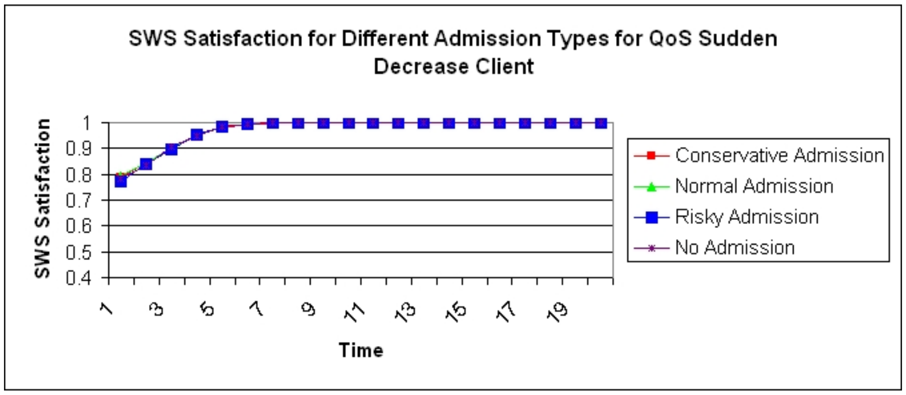
6.6.3 Sudden Change in Number of Requests

In this section we observe the effects of the client suddenly changing the number of requests to the community. We first observe the effects of having a sudden increase in the number of requests and then observe the effects of having a sudden decrease in the number of requests. The average QoS for the requests do not change. In the experiments we observe the number of handled requests for the different admission types as well as the levels of satisfaction of all 3 parties for the different admission types.

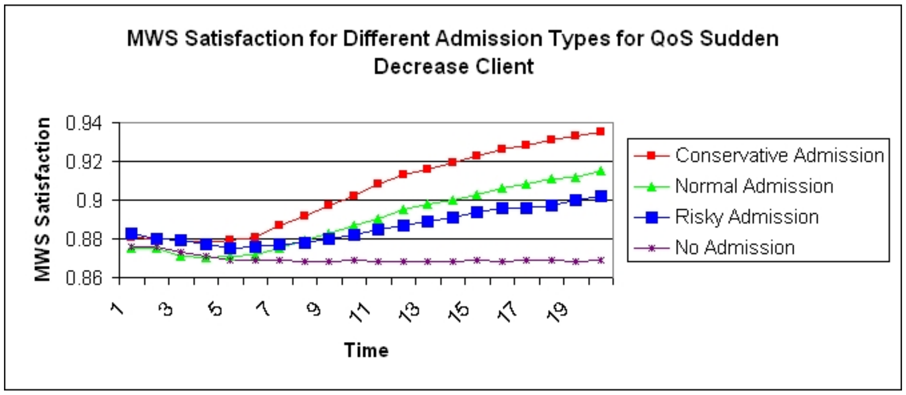
A sudden increase in the number of requests to the community can catch the community offguard. It may not be ready to handle the sudden increase of requests. In figure 6.7 we observe the number of handled requests for the 3 admission types as well as the situation where the community does not admit



(a) Client Satisfaction for Sudden Decrease in QoS Requirements



(b) SWS Satisfaction for Sudden Decrease in QoS Requirements



(c) MWS Satisfaction for Sudden Decrease in QoS Requirements

Fig. 6.6. Satisfactions for Sudden Decrease in QoS Requirements

any Web services. We notice in figure 6.7 that as expected, the conservative approach allows the community to handle more requests in the event that there is a sudden change in the number of requests.

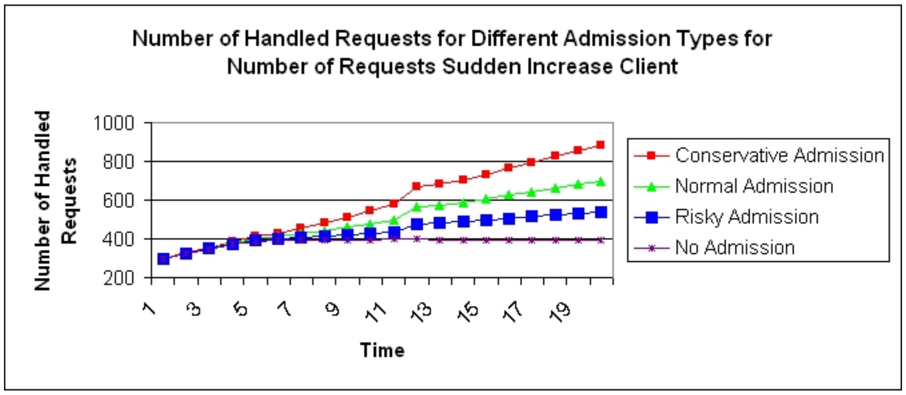
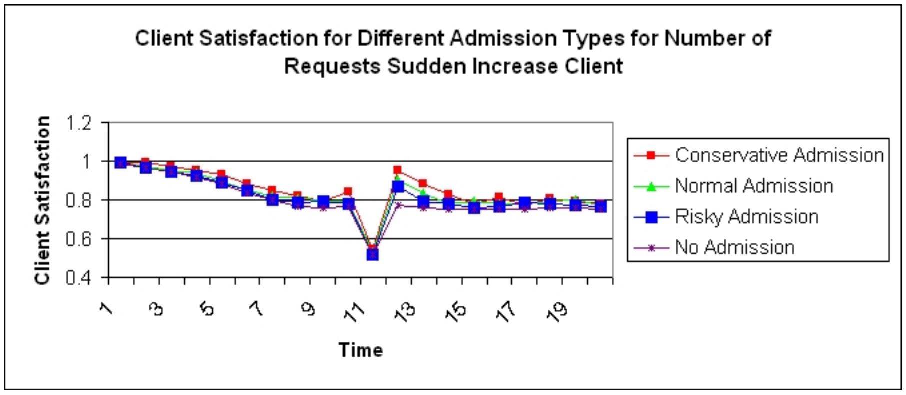


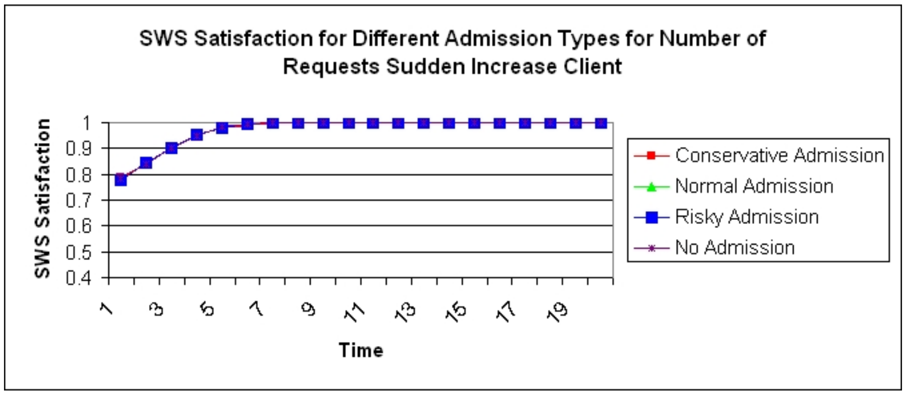
Fig. 6.7. Number of Handled Request for Sudden Increase in Number of Requests From Client

In figure 6.8 we observe the satisfactions for the 3 admission types as well as the situation where the community does not admit any Web services. We notice a sharp drop in the client satisfaction in figure 6.8-(a) when there is a jump in the number of requests. This is due to the large number of requests that were not handled. However, immediately after the sudden drop, we notice that the conservative approach allows for the biggest client satisfaction. The SWS and MWS satisfactions in figure 6.6-(b) and figure 6.6-(c) respectively remain consistent with previous experiments.

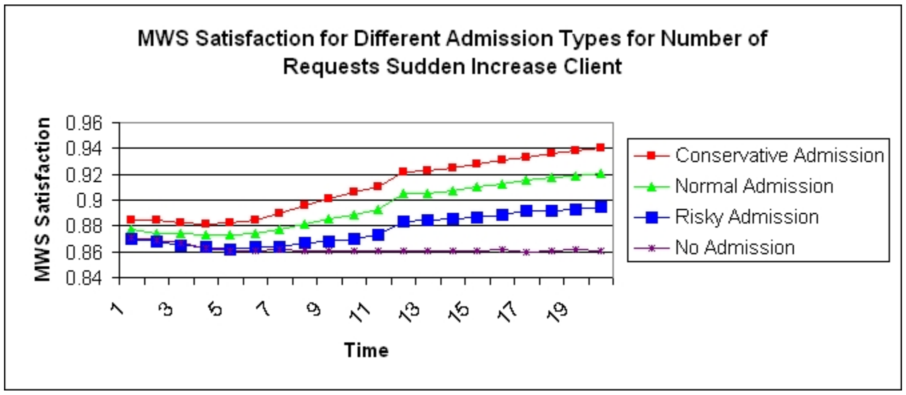
A sudden decrease in the number of requests to the community has a direct decreasing effect on the number of handled requests. This is because the community has less requests to handle. This causes to changes in the satisfactions as well. In Figure 6.9 we observe the number of handled requests for the 3 admission types as well as the situation where the community does not admit any Web services. We notice that the number of handled requests



(a) Client Satisfaction for Sudden Increase in Number of Requests From Client



(b) SWS Satisfaction for Sudden Increase in Number of Requests From Client



(c) MWS Satisfaction for Sudden Increase in Number of Requests From Client

Fig. 6.8. Satisfactions for Sudden Increase in Number of Requests From Client

drops when there is a drop in the number of requests from the client, this is natural as there are less requests that can be handled by the community.

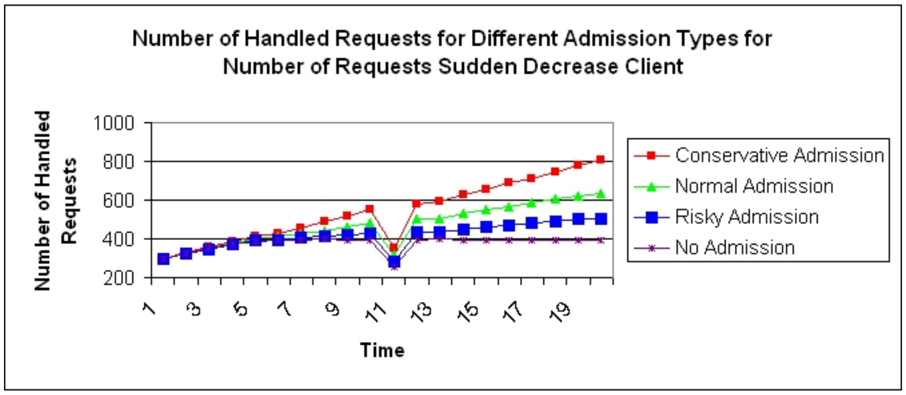


Fig. 6.9. Number of Handled Request for Sudden Decrease in Number of Requests From Client

In figure 6.10 we observe the satisfactions for the 3 admission types as well as the situation where the community does not admit any Web services. We notice a sharp increase in client satisfaction in figure 6.10-(a) when the drop in number of requests occurs. This increase in client satisfaction is due to the high chance that all requests were handled and that the community was able to use the SWSs with the highest QoS properties to handle all the requests. This would mean that the capacities of SWSs were not full, as is depicted in figure 6.10-(b). We see in this figure that the SWS satisfaction drops when there is a sharp decrease in the number of requests. The conservative approach has the lowest SWS satisfaction during this drop since it has the highest number of SWS count in the community during that time.

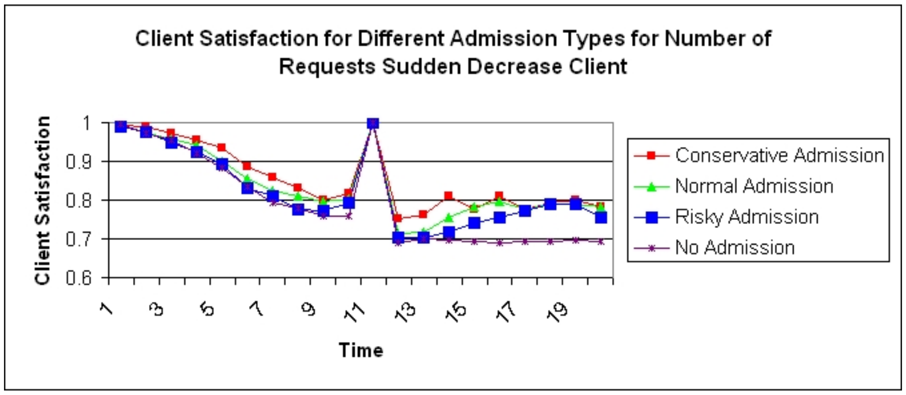
We notice an increase in the MWS satisfaction when there is a sharp decrease in the number of requests from the client in figure 6.10-(c). An interesting observation here is that the increase is normal when the community takes a conservative approach in the admission. This can be due to the fact since there is a higher number of SWSs in the community, these SWSs already provide the highest possible income to the community.

6.6.4 Experiments Conclusion

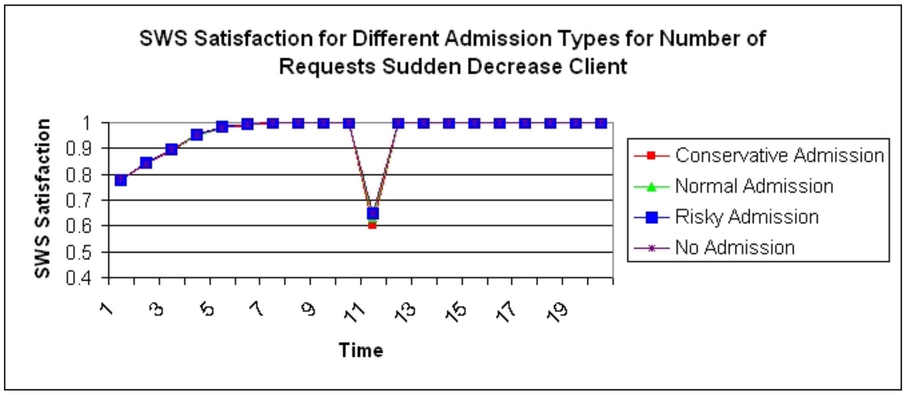
The experiments show how the number of handled requests, client satisfaction, SWS satisfaction, and MWS satisfaction can be affected by the MWS's approach on how to admit and expel Web services. The results show the flexibility that can be given to a MWS and its corresponding effects.

By modelling different types of clients, the experiments also show the importance of the MWS's approach. Having a large buffer gives the MWS the flexibility to not incur a huge penalty in the event of sudden changes by the client.

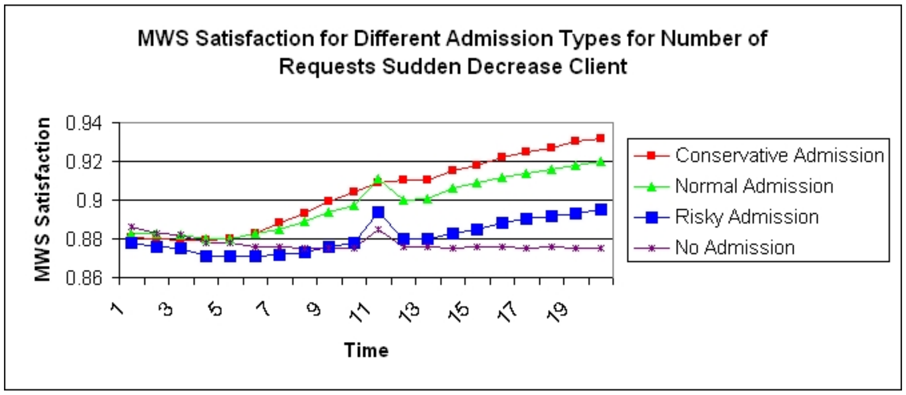
The experimental results also show the benefits of admission and expulsion compared to the situation where no admission nor expulsion is adopted by the community. They show that the satisfactions of all 3 parties, and therefore, the sustained quality of Web services is better if a community adopts admission and expulsion.



(a) Client Satisfaction for Sudden Decrease in Number of Requests From Client



(b) SWS Satisfaction for Sudden Decrease in Number of Requests From Client



(c) MWS Satisfaction for Sudden Decrease in Number of Requests From Client

Fig. 6.10. Satisfactions for Sudden Decrease in Number of Requests From Client

7 Discussion

In this chapter we perform a comparison between the proposed approach in this thesis with other solutions proposed by other authors in the domain of Web services. This includes the limitations of the proposed mechanisms in this thesis. The comparison has to be discussed at two levels, at the user request management level when individual requests are sent to the community, and at the member management level where the community has the opportunity to admit and expel Web services in the community as it feels necessary.

The comparisons are done qualitatively. This means that our focus is on the features between the different approaches. For example, the 3 way satisfaction in our approach which considers the satisfaction of all three parties is a feature that, to the best of our knowledge, has not been proposed by other authors.

In Section 7.1 we compare our proposed approach with other solutions at the user request management level. Next, in Section 7.2 we compare our proposed approach with other solutions at the member management level.

7.1 Discussion - User Request Management

The features that we compare in the user request management level are:

- Selection responsibility on broker or client - whether the selection is done by the service broker or the service client (Section 7.1.1)

- QoS based - whether the level of QoS provided by the service providers is considered in the selection process (Section 7.1.2)
- Three way satisfaction - whether the selection process takes into account the satisfaction of the service provider and service broker (if applicable) and the service client (Section 7.1.3)
- Submission of price only - is it possible for the client to submit only the price that it is willing to pay (Section 7.1.4)
- Fully automated - can the selection be done automatically without manual interference from a system administrator or client (Section 7.1.5)
- Single point of failure - whether a single point of failure exists in the solution (Section 7.1.6)
- Substitution Mechanism - whether a method of substitution is proposed in the event that the service provider fails (Section 7.1.7)
- Composition - whether an aggregation of Web services is proposed in order to create a composition of Web services to provide a more comprehensive service (Section 7.1.8)
- Specific QoS properties - whether a specific set of QoS properties or a general formula was presented (Section 7.1.9)

Finally, we summarize the discussion on the user request management in Section 7.1.8.

7.1.1 Selection Responsibility

In some cases where the selection of service provider is done by the client [79, 85, 98, 110], the service broker plays a part in service discovery and collection of QoS levels data. [98] calls the service broker a certifier but does not do the selection itself. [85] allows the service broker to trim the list of suitable service providers, but the eventual selection is still under the responsibility of the service client. In our approach, this selection is performed by the service broker (MWS).

In cases where selection is done by the client, while the client retains more control, it also means that the responsibility for any kind of substitution falls on the client. This means that more work has to be done if the selected service provider were to fail. However, in cases where the selection is done by the broker, any substitution is carried out by the broker, unknown to the client.

7.1.2 QoS Based Analysis

All the literature that we came across considered the QoS of the service providers as part of the selection process. The difference between the models in the literature depend on the number of QoS properties modelled. In our thesis we used 4 QoS properties (Availability, Reliability, Execution Duration, and Price) whereas other literature used different numbers of QoS properties. For example, in [79] 3 were used (price, execution duration and reputation), while in other literature no specific properties were mentioned but formulas were presented [4].

7.1.3 Three-way Satisfaction

[128] proposes two selection algorithms that is to be conducted by a service broker. One is a homogeneous algorithm where the amount of resources is shared by all clients, and a non-homogeneous algorithm where clients with a higher requirement gets more resources. Their approach assumes that resources provided by service providers are divisible, whereas in this thesis, we assume that each request can only be handled by a single service provider.

The other approaches in Table 7.1 only look at satisfying the client by fulfilling the QoS requirements of the request. These literatures do not take into account the satisfaction of the service provider nor the service broker. Our approach takes into consideration the satisfaction of three parties - the service client, the service provider and the service broker. This approach is novel and a key contribution to the research community.

7.1.4 Submission of price only

To the best of our knowledge, within the domain of Web services, there is no literature that proposes submitting one QoS property as a method of selection. In this thesis we have proposed a selection mechanism that allows clients to submit only the price that it is willing to pay. In this modern age where budgets are getting increasingly tight, such a model is becoming more relevant.

7.1.5 Fully automated

The main advantage of using Web services is its ability to be automated. A fully automated selection allows the service provider to be selected automatically without any interference from the system administrator or the client. In our solution we have specified a selection mechanism that is capable of being fully automated. One assumption that we have made is that the communication methods and syntax language has been determined beforehand among the parties (See Section 2.3.2).

Solutions have been proposed in order to better enable full automation among Web services (See Section 2.3.2). For example, Protocols such as SOAP and WSDL allow the three parties to communicate with each other. These solutions can be implemented alongside the approaches in Table 7.1.

7.1.6 Single point of failure

In some literature within the domain of Web services, there exists a single point of failure in the service broker. This applies for approaches where the selection is done by the service client [79,85,98,110], this is because the service broker is responsible for gathering the QoS information and if this fails, then the service client does not have any information to make the selection. The one exception [87] invokes a new agent for each request. This agent collects QoS information on behalf of the client. If one agent were to fail, it does not affect the other agents of other requests. This approach is more robust than our approach of a community of Web services since if the MWS were to fail, then no requests can be handled.

7.1.7 Substitution Mechanism

[115] mentioned the need to substitute or replace a service provider if it fails but did not propose a mechanism for doing so. In our thesis we have provided a full mechanism for the community to substitute service providers in case of failure. The substitution mechanism allows the service client to submit the request once and ignore any possible failures that might occur on the service providers. This is because, in our approach, the MWS is responsible for finding a service provider to substitute the failed service provider and send the eventual reply back to the service client.

7.1.8 Composition

Web services of different functionality can be aggregated together in order to provide a service that has more functionality than each individual Web service. Such an aggregation is called a composition. In our thesis we do not deal with composition of Web services since a community of Web services consists of Web services with the same functionality. However, in other literature, the aggregation of Web services is heavily discussed [4, 5, 79, 85, 115, 132] which result in a composition.

7.1.9 Modelled with Specific QoS properties

In some literature, including this thesis, the models presented include specific QoS properties. In this thesis we use 5 QoS properties (price, availability, reliability, execution duration and capacity). This approach is similar to other literature [5, 47, 110, 115, 132] which may use different QoS properties. Some literature do not list specific QoS properties [4,40,85,87,128]. Some approaches provide both [79].

Although we have specific QoS properties in this thesis, it is trivial to extend our formulas to a more general formula to encompass different QoS properties. Our proposed weighted sum formulas can be extended to include more than 5 QoS properties. The use of both probabilistic (availability and reliability) and deterministic values (price, execution duration, and capacity)for the QoS properties in our thesis make this extension trivial.

7.1.10 Summary

We summarize these features across different approaches in Table 7.1 where m is a constant and n is the number of service providers to choose from.

Table 7.1. Comparison of qualitative properties

Qualitative Property	**This Thesis**	[79, 85, 98]	[87]	[4, 5, 40, 47, 79, 110, 128, 132]	[115]
Selection Responsibility	Broker	Client	Broker (Agent)	Broker	Broker
QoS based	Yes	Yes	Yes	Yes	Yes
Three-way Satisfaction	Yes	No	No	No	No
Submission of price only	Yes	No	No	No	No
Fully automated	Yes	Yes	Yes	Yes	Yes
Single point of failure	Yes	Yes	No	Yes	Yes
Substitution Mechanism	Yes	No	No	No	Yes
Composition	No	Yes	No	Yes (No for [40, 47, 79, 110, 128])	Yes
Specific QoS Properties	Yes	Yes (No for [85])	No	Yes (No for [4, 40, 128])	Yes

7.2 Discussion - Member Management

The features that we compare in the member management level are:

- Three way satisfaction - the decision on whether a service provider joins or leaves a group of Web services is done considering the satisfaction of the service provider, service client and service broker (Section 7.2.1)

- Auto-updating of QoS Level - whether the mechanism allows an automated update of the QoS of service providers (Section 7.2.2)
- QoS of a group of Web services - whether it is possible to calculate the QoS level of a group of Web services (Section 7.2.3)
- Broker and Provider SLA negotiation per request - whether the service broker and service provider has to negotiate a new SLA for each request that the service client submits (Section 7.2.4)

7.2.1 Three way satisfaction

Our approach presents a mechanism of deciding which Web services to join the community that takes into consideration the satisfaction of the service client, service provider and service broker. In the domain of composition of Web services, [4,5,50,58,59,96,130,132] consider which Web services to admit into the composition by considering the level of QoS provided to the service client but disregard the satisfaction of the service providers and the composition as a whole. By taking into account the satisfaction of the service providers and service broker, our approach can better sustain the long-term QoS as shown in the experiments in Section 6.6.

7.2.2 Auto-updating of QoS Level

Ensuring the QoS of service providers remain a challenging task. Certain solutions include having an external party continuously probe the QoS level of the service providers [19,133], other solutions consider creating the history of the service providers and updating the *reputation* property of the service provider. [110] takes the approach of having a service broker certify the QoS level of the service providers before they are published on the registry.

The updating of QoS level occurs naturally in our approach due to the proposed auction system within a community of Web services. Since all traffic flows between the service client and SWS, the MWS is very clear on the level of QoS that is provided to the service client. In addition, the proposed auction system ensures that the SWSs truthfully reveal the level of QoS that they can provide.

7.2.3 QoS of a group of Web services

There exists literature on the end-to-end QoS provided by a composition of Web services [59, 130], but this is not applicable when determining the QoS of a group of similar Web services. We published in an earlier paper [77] that explains how this can be done. One benefit of being able to determine the QoS of a group of similar Web services is to publish such information on the service registry. This allows the service client to accurately determine its expectations when sending a request to the community.

7.2.4 Broker and Provider SLA negotiation per request

In a community of Web services, the relationship between the service broker and service provider is determined when the service provider joins the community. However, in a composition of Web services [61], the relationship between service provider and service broker is further and negotiation of a SLA is required each time the service broker requests for the services of the service provider. While this negotiation can be automated [104], it provides an extra layer of complication and takes more time each time the service client submits a request.

8 Conclusion

In this thesis we have shown our approach on how to sustain the quality of Web services. We first introduced the problem and the complexity behind it. We then presented the related work in other literature on how to sustain the quality of Web services from fields ranging from the SOA domain to cloud computing.

We proposed to use the framework of a community of Web services in order to sustain the quality of Web services. We expanded on the CNP by introducing auction theory into the community of Web services. Next, we presented our proposed mechanisms into 2 main categories, the first involves the selection process of a Web service when a request is received by the community which we call the user request management. The second category involves the overall management of Web services in the community in order to better sustain the quality of Web services which we call the member management.

Fixed-price auction was proposed in order to allow service clients to submit requests based on a fixed price. We showed the properties of using a fixed-price auction and how it can be incorporated into the framework of a community of Web services. This auction mechanism was used as part of a 3-way satisfaction selection approach that takes into account the satisfaction of the service client, the service provider and the service broker. In addition, QoS monitoring approaches were proposed in the context of a community in order to best determine whether a substitution needs to occur if an originally selected service provider were to fail. A substitution mechanism was proposed using a similar 3-way satisfaction approach in order to best select the service provider to handle the substitution process.

In addition, we performed experiments using real-world data to show the benefits of using the fixed-price auction system. We compared the fixed-price auction system with the basic method of selecting the SWS to handle the request and an auction based selection where the MWS sends the SWSs all QoS requirements except the price. In our conclusion for the fixed-price auction experiments, we showed that the auction system gives more flexibility to the community. The fixed-price auction system gives the client the flexibility of submitting only the price to the community. Such a feature is important in today's context where budget is a major constraint. Although in our experiments and proposal we have used price as a fixed value to conduct the fixed-price auction system, another QoS property other than price could be fixed if the community prefers it. This can be done according to the design of the community.

The 3-way satisfaction approach for user request management section were simulated using experiments that compared both the selection and substitution processes. These experiments were conducted with real-world values in order to improve the credibility of the experiments. We showed in the experiments how the 3-way satisfaction approach can be implemented within the context of a community of Web services. In the experiments for both the selection and substitution processes, the 3-way satisfaction approach was compared with the traditional method of using only the satisfaction of the client to select or substitute the Web service. We showed in the experiments that using the 3-way satisfaction approach gives a higher total satisfaction over time compared to the traditional method. This higher total satisfaction is eventually translated into a higher income for the community as well as a higher number of handled requests in total.

Next in the member management chapter, we proposed mechanisms to manage the components of a community of Web services. Before any admission or expulsion can occur, we proposed a method to best predict the QoS that a community can provide as a whole, as well as methods to project the client requirements. With the projected client requirements, the community can then better manage the number as well as the type of service providers in the community. This involves both the admission and the expulsion of service providers from the community. We proposed a 3-way satisfaction approach in order to best select service providers to join the community. A similar 3-way satisfaction approach was proposed to decide which service providers to expel from the community.

We demonstrated how the 3-way satisfaction approach can be used in the admission and expulsion of Web services in a community. We considered different types of client that behave differently and see a conservative, normal or risky approach by the community reacts to these different clients. We demonstrated how different property values can affect the satisfaction of the different parties. Using the admission and expulsion mechanisms earlier, we show how the MWS can make its own decisions on how it would like to administer the community.

We compared our approaches in the discussion chapter. We looked at the qualitative features of our approach with other approaches within the domain of Web services. Similar to the rest of the thesis, the comparison was divided into the user request management section, and the member management section.

Through our proposed mechanisms, we have shown how our unique approach of 3-way satisfaction using the framework of a community of Web services can sustain the quality of Web services.

9

Future plans

In this chapter, we look at possible further research topics that might be developed from this thesis.

In this thesis we have assumed that the selection of the MWS is a conceptual one. We acknowledge that the MWS is a single point of failure within the framework of a community of Web services. One potential topic could include a mechanism on how a SWS might promote itself to become a MWS in the event that the MWS were to fail.

We introduced auction theory into the framework of a community of Web services. One area that might be lacking is research into the collaboration among service providers. Although in our current framework the service providers do not theoretically know the identity of each other, the realistic possibility of this cannot be ruled out. When service providers discover the identity of each other, it would be interesting to consider the effects of potential collaboration among each other. It would be interesting to know how such collaboration might affect the fixed-price second QoS auction. More importantly, further research could go into what additional steps the community can do to negate or at least minimize the effects of such collaboration.

It might also be interesting to see how auction theory would be implemented within the context of cloud computing. In the domain of cloud computing, a selection is also done across different levels to determine which service at a lower layer is selected to handle the request. It seems entirely plausible to implement auction theory into this selection process, however, more research needs to be done on exactly how this can be implemented and the implications for doing so.

The 3-way satisfaction approach could also possibly be extended into the domain of cloud computing. The similar selection process could take into account the satisfaction of multiple parties and not just the satisfaction of the service client. The exact implementation needs more research and the implications within the domain of cloud computing need to be discussed as well.

In our thesis, although we used some real-world values, we feel that a running working example could best be used in order to illustrate the benefits of our approach. This would allow a better and more accurate analysis of how well the proposed mechanisms work.

This concludes the future possible research topics on the PhD thesis titled *Sustaining the Quality of Web Services.*

10

Acknowledgements

I would like to thank many people who have helped me through the completion of this thesis. The first is my PhD promoter, Prof. Philippe Thiran, who is encouraging, honest, and a true mentor throughout the past 4 years. I am also grateful towards Prof. Zakaria Maamar with his kind advice and helpful criticism with several of my papers. He and Prof. Thiran are part of the *comite d'accompagnement* consisting also of Prof. Vincent Englebert and Prof. Djamal Benslimane. I would also like to thank the computer science department at the University of Namur and the PReCISE Research Centre for giving me the resources as well as La Wallonie and the European Regional Development Fund (ERDF) for funding me the past 4 years.

I am blessed to have worked with Prof. Jamal Bentahar who took me under his care at Concordia University for two weeks. I am thankful towards the assistance provided by CETIC with regards to the E-Health project, including Mr. Fabian Steels, Mr. Sbastien Rousseaux and Mr Gautier Dallons.

My colleagues in the office have been helpful and encouraging. Including Dr. Mohamed Boukhebouze, Dr. Mohanad Al-jabari, Mr. Amanuel Alemayehu, and Mr. Neto Waldemar. I am grateful to have worked with Dr. Hamdi Yahyaoui.

I am thankful for and would like to acknowledge many others who helped me along the way. This includes, but is not limited to Babak Khosravifar, Cedric Aerts, Babette Di Guardia, Anthony Cleve, Anas Laurent, and Nikolaos Matskanis.

I would also like to thank my mum for being such a source of inspiration, my family back home who has tolerated my absence for so many years. Family and friends have helped calmed me down and definitely helped me through of process of attaining the PhD. I thank you all.

References

1. Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1257–1258. ACM, 2007.
2. Rashid J Al Ali, Omer F Rana, David W Walker, Sanjay Jha, and Shaleeza Sohail. G-qosm: Grid service discovery using qos properties. *Computing and Informatics*, 21(4):363–382, 2012.
3. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications.* Springer Publishing Company, Incorporated, December 2010.
4. Mohammad Alrifai and Thomass Risse. Combining global optimization with local selection for efficient qos-aware service composition. *Proc. of the 18th international conference on World wide web*, pages 881–890, 2009.
5. Danilo Ardagna and Barbara Pernici. Global and local qos guarantee in web service selection. *Business Process Management Workshops*, 3812:32–46, 2006.
6. Alvin AuYoung, Brent Chun, Alex Snoeren, and Amin Vahdat. Resource allocation in federated distributed computing infrastructures. *1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure*, 9, October 2004.
7. Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-time monitoring of instances and classes of web service compositions. *IEEE International Conference on Web Services*, pages 63–71, 2006.

8. B. Benatallah, Q. Z. Sheng, and M. Duman. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40–48, January - Febuary 2003.
9. Jamal Bentahar, Zakaria Maamar, Djamal Benslimane, and Philippe Thiran. An argumentation framework for communities of web services. *Intelligent Systems 2007*, 22(6):75–83, November/December 2007.
10. Francine Berman and Richard Wolski. The apples project: A status report. *Proceedings of the 8th NEC Research Symposium*, 16, 1997.
11. Ken Binmore. Fun and games, a text on game theory. 1992.
12. Ken Birman, Robbert Van Renesse, and Werner Vogels. Adding high availability and autonomic behavior to web services. *26th International Conference on Software Engineering*, pages 17–26, 2004.
13. Kenneth Birman. Building secure and reliable network applications. *Worldwide Computing and Its Applications*, 1274:15–28, 1997.
14. Gregor V. Bochmann, Brigette Kerherve, Hanan Litfiyya, Mohammed-Vall M. Salem, and Haiwei Ye. Introducing qos to electronic commerce applications. *Second International Symposium on Topics in Electronic Commerce*, pages 138–147, 2001.
15. Craig Boutilier. Sequential optimality and coordination in multiagent systems. *IJCAI*, 99:478–485, 1999.
16. Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. *The primary-backup approach*, pages 199–216. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.
17. Rajkumar Buyya, David Abramson, Jonathan Giddy, et al. A case for economy grid architecture for service-oriented grid computing. *Parallel and Distributed Processing Symposium, International*, 1:20083–1, 2001.
18. Rajkumar Buyya, David Abramson, and Srikumar Venugopal. The grid economy. *Proc. of the IEEE*, 93(3):698–714, March 2005.
19. Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. Qos-aware replanning of composite web services. *IEEE International Conference on Web Services*, pages 121–129, 2005.

20. Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposite, and Maria Luisa Villani. A lightweight approach for qos-aware service composition. *2nd International Conference on Service Oriented Computing(ICSOC'04)*, 2004.
21. Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281 – 308, 2004.
22. Henri Casanova, Francine Berman, Graziano Obertelli, and Richard Wolski. The apples parameter sweep template: User-level middleware for the grid. *SC Conference*, 0:60, 2000.
23. Henri Casanova and Jack Dongarra. Netsolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11:212–223, 2000.
24. Gilles Fedak Cecile, Gilles Fedak, Cecile Germain, and Vincent Neri. Xtremweb : A generic global computing system. *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID01)*, pages 582–587, 2001.
25. Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. All learning is local: Multi-agent learning in global reward games. *Neural Information Processing Systems*, 2004.
26. Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew S. Grimshaw. The legion resource management system. *Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 162–178, 1999.
27. Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys (CSUR)*, 33:427–469, December 2001.
28. Wesley W Chu, Leslie J Holloway, Min-Tsung Lan, and Kemal Efe. Task allocation in distributed data processing. *Computer*, 13(11):57–69, 1980.
29. Brent N. Chun, Philip Buonadonna, Alvin Auyoung, Chaki Ng, David C. Parkes, Jeffrey Shneidman, Alex C. Snoeren, and Amin Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. *Proc. of the 2nd IEEE Workshop on Embedded Networked Sensors*, pages 19–28, May 2005.
30. Brent N. Chun and David E. Culler. User-centric performance analysis of market-based cluster batch schedulers. *2nd IEEE International Symposium on Cluster Computing and the Grid*, pages 30–38, 2002.

31. P Emerald Chung, Yennun Huang, Shalini Yajnik, Deron Liang, and Joanne Shih. Doors: Providing fault tolerance for corba applications. *IFIP International Conference on Distributed System Platforms and Open Distributed Processing (Middleware98)*, 1998.

32. ASP Industry Consortium. White paper on service level agreements, 2000. `http://www.cisco.com/warp/public/765/emea/aip_asp/documents/SLA_White_Paper.pdf`, June 2011.

33. World Wide Web Consortium. `http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/`, July 2013.

34. World Wide Web Consortium. `http://www.w3.org/`, July 2013.

35. Antonio Carlos da Rocha Costa and Yves Demazeau. Toward a formal model of multi-agent systems with dynamic organizations. *2nd. International Conference on Multi-agent Systems, ICMAS*, 96:431, 1996.

36. A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, and A. Youssef M. Spreitzer. Web services on demand: Wsla-driven automated management. *IBM Systems Journal*, 43(1):136–158, 2004.

37. Amir Vahid Dastjerdi. Qos-aware and semantic-based service coordination for multi-cloud environments. 2013.

38. Amir Vahid Dastjerdi, Saurabh Kumar Garg, and Rajkumar Buyya. Qos-aware deployment of network of virtual appliances across multiple clouds. *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 415–423, 2011.

39. Cary A. Deck and Bart J. Wilson. Fixed revenue auctions: Theory and behavior. *Economic Inquiry*, 46:342–354, 2008.

40. Vassiliki Diamadopoulou, Christos Makris, Yannis Panagis, and Evangelos Sakkopoulos. Techniques to support web service selection and consumption with qos characteristics. *Journal of Network and Computer Applications*, 31(2):108–130, 2008.

41. Glen Dobson, Russell Lock, and Ian Sommerville. Qosont: a qos ontology for service-centric systems. *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 80–87, 2005.

42. Jim E Doran, S Franklin, Nicholas R Jennings, and Timothy J Norman. On cooperation in multi-agent systems. *Knowledge Engineering Review*, 12(3):309–

314, 1997.

43. Stuart E. and Thiel. Multidimensional auctions. *Economics Letters*, 28(1):37 – 40, 1988.

44. Kemal Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, 1982.

45. The Free Online Encyclopedia. The free online encyclopedia. `http://www.thefreedictionary.com/`, June 2011.

46. Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, August 2005.

47. Peter Farkas and Hassan Charaf. Web services planning concepts. *Journal of WSCG*, 11(1):23–35, 2003.

48. Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. *International Conference on Multi Agent Systems*, pages 128–135, 1998.

49. Les Gasser, Carl Braganza, and Nava Herman. Mace: A flexible testbedfor distributed ai research. *Distributed artificial intelligence*, page 119, 1987.

50. M. Gillmann, G. Weikum, and W. Konner. Workflow management with service quality guarantees. *ACM SIGMOD International Conference on Management of Data*, pages 228–239, 2002.

51. Dani Goldberg and Maja J Matarić. Coordinating mobile robot group behavior using a model of interaction dynamics. *Third annual conference on Autonomous Agents*, pages 100–107, 1999.

52. Claudia V Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. *Second international joint conference on Autonomous agents and multiagent systems*, pages 137–144, 2003.

53. Steve Graham, Glen Daniels, Doug Davis, Yuichi Nakamura, Simeon Simeonov, Peter Brittenham, Paul Fremantle, Dieter Koenig, and Claudia Zentner. *Building Web services with Java: making sense of XML, SOAP, WSDL, and UDDI*. SAMS publishing, 2004.

54. Andreas Hanemann and Martin Sailer. A framework for service quality assurance using event correlation techniques. *Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Re-*

sources Conference/E-Learning on Telecommunications Workshop, pages 428–433, 2005.

55. W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
56. Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2004.
57. San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Information Sciences*, 177:5484–5503, December 2007.
58. Michael C. Jaeger, Gregor Rojec-Goldmann, , and Gero Muhl. Qos aggregation for web service composition using workflow patterns. *8th IEEE International Enterprise Distributed Object Computer Conference*, 0:149–159, 2004.
59. Michael C Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. Qos aggregation for web service composition using workflow patterns. *IEEE 8th International Conference on Enterprise Distributed Object Computing*, pages 149–159, 2004.
60. Srikumar Venugopal James Broberg and Rajkumar Buyya. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.
61. Li jie Jin, Li jie Jin, Vijay Machiraju, Vijay Machiraju, Akhil Sahai, and Akhil Sahai. Analysis on service level agreement of web services. *Technical Report HPL-2002-180, HP Laboratories*, 2002.
62. Radu Jurca, Boi Faltings, and Walter Binder. Reliable qos monitoring based on client feedback. *16th international conference on World Wide Web*, pages 1003–1012, 2007.
63. Lazowski J. Juszczyk, L. and S. Dustdar. Web service discovery, replication, and synchronization in ad-hoc networks. *1st International Conference on Availability, Reliability, and Security (ARES'2006')*, pages 847–854, 2006.
64. Nirav H. Kapadia and Jos A.B. Fortes. Punch: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, 2:153–164, 1999.
65. Alexander Keller and Heiko Ludwig IBM T. J. Watson Research Center. Defining and monitoring service level agreements for dynamic e-business. *LISA '02: Sixteenth Systems Administration Conference*, pages 189–204, 2002.

66. Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11:57–81, 2003. 10.1023/A:1022445108617.

67. Taha Khedro and Michael R Genesereth. Facilitators: A networked computing infrastructure for distributed software interoperation. *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence in Distributed Information Networks*, 1995.

68. Babak Khosravifar, Jamal Bentahar, and Ahmad Moazin. Analyzing the relationships between some parameters of web services reputation. *IEEE International Conference on Web Services (ICWS)*, pages 329–336, 2010.

69. Ryszard Kowalczyk, Peter Braun, et al. Towards agent-based coalition formation for service composition. *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 73–80, 2006.

70. Sarit Kraus. Beliefs, time and incomplete information in multiple encounter negotiations among autonomous agents. *Annals of Mathematics and Artificial Intelligence*, 20(1-4):111–159, 1997.

71. Sarit Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1):79–97, 1997.

72. D. Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang: A language for defining service level agreements. *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 100–106, 2003.

73. Hoong Chuin Lau and Lei Zhang. Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. *15th IEEE International Conference on Tools with Artificial Intelligence*, pages 346–350, 2003.

74. Lundy Lewis. *Managing business and service networks.* Springer; 1st Edition (April 30, 2001).

75. Baochun Li, Dongyan Xu, and Klara Nahrstedt. An integrated runtime qos-aware middleware framework for distributed multimedia applications. *Multimedia Syst.*, 8:420–430, December 2002.

76. Deron Liang, Chen-Liang Fang, Chyouhwa Chen, and Fengyi Lin. Fault tolerant web service. *Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference*, pages 310–319, 2003.

77. Erbin Lim, Philippe Thiran, Zakaria Maamar, and Jamal Bentahar. On the analysis of satisfaction for web services selection. *IEEE Ninth International Conference on Services Computing (SCC)*, pages 122–129, 2012.
78. M.; Mutka M.W. Litzkow, M.J.; Livny. Condor-a hunter of idle workstations. *8th International Conference on Distributed Computing Systems, 1988*, pages 104–111, 1988.
79. Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. *13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, 2004.
80. Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona: an architecture and library for creation and monitoring of ws-agreents. *2nd international conference on Service oriented computing*, pages 65–74, 2004.
81. Heiko Ludwig, Alexander Keller, Asit Dan, Richard King, and Richard Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3:43–59, 2003. 10.1023/A:1021525310424.
82. Z. Maamar, S. Subramanian, P. Thiran, D. Benslimane, and J. Bentahar. An Approach to Engineer Communities of Web Services - Concepts, Architecture, Operation, and Deployment -. *International Journal of E-Business Research, IGI Global Publishing*, 5(4), 2009.
83. Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and distributed Computing*, 59(2):107–131, 1999.
84. A. Mani and A. Nagarajan. Understanding quality of service for web services. *IBM developerWorks*, Jan 2002.
85. Umardand Shripad Manikrao and TV Prabhakar. Dynamic selection of web services with recommendation system. *International Conference on Next Generation Web Services Practices*, pages 5–pp, 2005.
86. Francisco Maturana, Weiming Shen, and Douglas H Norrie. Metamorph: an adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, 37(10):2159–2173, 1999.
87. E Michael Maximilien and Munindar P Singh. A framework and ontology for dynamic web services selection. *Internet Computing, IEEE*, 8(5):84–93, 2004.

88. E Michael Maximilien and Munindar P Singh. Multiagent system for dynamic web services selection. *First Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE at AAMAS)*, pages 25–29, 2005.

89. R. Preston Mcafee, Wendy Takacs, and Daniel R. Vincent. Auctions and bidding. *Journal of Economic Theory*, 25:158–179, 1987.

90. Lee W. McKnight and Jahangir Boroumand. Pricing internet services: Approaches and challenges. *Computer*, 33:128–129, February 2000.

91. D. A. Menasce. Qos issues in web services. *IEEE Internet Computing*, 6:72–75, November 2002.

92. Michael Menzel and Rajiv Ranjan. Cloudgenius: decision support for web server cloud migration. *21st international conference on World Wide Web*, pages 979–988, 2012.

93. Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Comprehensive qos monitoring of web services and event-based sla violation detection. *4th international workshop on middleware for service oriented computing*, pages 1–6, 2009.

94. Luis Nogueira and Luis Miguel Pinho. Iterative refinement approach for qos-aware service configuration. *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pages 155–164, 2006.

95. Longman English Dictionary Online. Longman english dictionary online. `http://www.ldoceonline.com/`, June 2011.

96. Yannis Panagis, Konstantinos Papakonstantinou, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Web service workflow selection using system and network qos constraints. *International Journal of Web Engineering and Technology*, 4(1):114–134, 2008.

97. Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. *Sixteenth conference on Uncertainty in artificial intelligence*, pages 489–496, 2000.

98. Shuping Ran. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.

99. Ori Regev and Noam Nisan. The popcorn market. online markets for computational resources. *Decision Support Systems*, 28(1):177–189, 2000.

100. Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic qos and soft contracts for transaction-based web services orchestrations. *IEEE Transactions on Services Computing*, 1:187–200, 2008.

101. Florian Rosenberg, Christian Platzer, and Schahram Dustdar. Bootstrapping performance and dependability attributes of web services. *IEEE International Conference on Web Services*, pages 205–212, 2006.

102. Hemant G Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings-Computers and Digital Techniques*, 141(1):1–10, 1994.

103. Thomas L Saaty. How to make a decision: the analytic hierarchy process. *European journal of operational research*, 48(1):9–26, 1990.

104. Akhil Sahai, Anna Durante, and Vijay Machiraju. Towards automated sla management for web services. *Hewlett-Packard Research Report HPL-2001-310 (R. 1)*, 2002.

105. Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad van Moorsel, and Fabio Casati. Automated sla monitoring for web services. *Management Technologies for E-Commerce and E-Business Applications*, 2506:28–41, 2002.

106. Akhil Sahai, Jinsong Ouyang, Vijay Machiraju, and Klaus Wurster. Specifying and guaranteeing quality of service for web services through real time measurement and adaptive control. Technical report, Management Project, E-Services Software Research Department, HP Laborataries, 1501 Page Mill Road, Palo Alto, CA 94034, 2001.

107. Perez-Sorrosal F. Patino-Martinez M. Salas, J. and R. Jimenez-Peris. Ws-replication: A framework for highly available web services. *15th International World Wide Web Conference*, pages 357–366, 2006.

108. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys (CSUR)*, 22:299–319, December 1990.

109. Fred B. Schneider. *Replication management using the state-machine approach*, pages 169–197. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

110. Mohamed Adel Serhani, Rachida Dssouli, Abdelhakim Hafid, and Houari Sahraoui. A qos broker based architecture for efficient web services selection.

IEEE International Conference on Web Services, pages 113–120, 2005.

111. Onn Shehory and Sarit Kraus. Task allocation via coalition formation among autonomous agents. *IJCAI (1)*, pages 655–661, 1995.
112. Onn Shehory and Sarit Kraus. Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. *ICMAS*, pages 330–337, 1996.
113. Onn Shehory and Sarit Kraus. A kernel-oriented model for coalition-formation in general environments: Implementation and results. *AAAI/IAAI, Vol. 1*, pages 134–140, 1996.
114. Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software: Practice and Experience*, 34:573–590, May 2004.
115. Amit Sheth, Jorge Cardoso, John Miller, Krys Kochut, and Myong Kang. Qos for service-oriented middleware. *Conference on Systemics, Cybernetics and Informatics*, pages 130–141, 2002.
116. Nan Feng Siun-Chuon, Nan Feng, Siun chuon Mau, and Narayan B. M. Pricing and power control for joint network-centric and user-centric radio resource management. *IEEE Trans. Commun*, 52:1547–1557, 2001.
117. R.G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29 Issue:12, December 1980.
118. Toshiharu Sugawara, Kensuke Fukuda, Toshio Hirotsu, Shin-ya Sato, and Satoshi Kurihara. Adaptive agent selection in large-scale multi-agent systems. *PRICAI 2006: Trends in Artificial Intelligence*, pages 818–822, 2006.
119. Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. *Tenth international conference on machine learning*, 337, 1993.
120. New York Times. `http://bits.blogs.nytimes.com/2011/04/21/amazon-cloud-failure-takes-down-web-sites/`, April 2011.
121. Hong-Linh Truong, Robert Samborski, and Thomas Fahringer. Towards a framework for monitoring and analyzing qos metrics of grid services. *Second IEEE International Conference on e-Science and Grid Computing*, pages 62–65, December 2006.

122. Fernando Vega-Redondo. *Economics and the Theory of Games*. Cambridge University Press, 2003.

123. C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: a distributed computational economy. *IEEE Transactions on Software Engineering*, pages 103–117, Feb 1992.

124. Karl E. Wiegers. *Software Requirements, 2nd Edition*. MICROSOFT PRESS, February 2003.

125. Jeffrey Wooldridge. *Introductory Econometrics*. South Western College; International ed of 4th revised ed edition (3 Oct 2008).

126. Ping Xuan, Victor Lesser, and Shlomo Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. *Proceedings of the fifth international conference on Autonomous agents*, pages 616–623, 2001.

127. Xinfeng Ye. Providing reliable web services through active replication. *6th IEEE ACIS International Conference on Computer and Information Science*, 0:1111–1116, July 2007.

128. Tao Yu and K-J Lin. The design of qos broker algorithms for qos-capable web services. *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 17–24, 2004.

129. Tao Yu and Kwei-Jay Lin. A broker-based framework for qos-aware web service composition. *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 22–29, 2005.

130. Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1:6, May 2007.

131. Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(03):303–328, 2001.

132. Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. *12th international conference on World Wide Web*, pages 411–421, 2003.

133. Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services com-

position. *IEEE Transactions on Software Engineering*, 30:311–327, May 2004.

134. Liangzhao Zeng, Hui Lei, and Henry Chang. Monitoring the qos for web services. *Service-Oriented Computing–ICSOC*, pages 132–144, 2007.
135. Wenying Zeng, Yuelong Zhao, and Junwei Zeng. Cloud service and service selection algorithm research. *First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 1045–1048, 2009.
136. Zibin Zheng and Michael R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. *IEEE International Conference on Web Services*, pages 145–152, 2008.
137. Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software: Practice and Experience*, 23(12):1305–1336, 1993.