A QoS-Aware Web Service Client Framework for GUISET

A dissertation submitted by:

Esau Laymon Mathonsi

(Reg No: 20040445)

To:

Department of Computer Science,

Faculty of Science and Agriculture,

University of Zululand,

Kwadlangezwa 3886, RSA

Supervisor: Prof MO Adigun

2011

DECLARATION

I, Esau Laymon Mathonsi, declare that this dissertation represents my own work and that this work has not been previously submitted at any university or other institution of tertiary education. All sources of information used in this workhave been acknowledged.

Signature_____

Esau L Mathonsi

DEDICATION

To Mandisa Mathonsi

ACKNOWLEDGEMENT

I would like to thank my supervisor, Prof. M.O. Adigun, for the opportunity and his guidance, advice and support.

I would also like to thank Edgar Jembere and Olatunji for their support and advice. I also would like to acknowledge my laboratory mates at the Centre for Mobile e-Services for their contribution. I'm grateful and thankful to thank my family for their understanding and support.

Finally, I thank Telkom for funding me throughout my research.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

Acronym	Definition
C-Broker	Classical QoS-aware broker
G-Broker	GUISTE QoS-aware broker
GUISET	Grid-based Utility Infrastructure for SMMEs Enabling Technologies
IT	Information Technology
SMMEs	Small, Medium and Micro Enterprises
SOA	Service Oriented Architecture
QoS	Quality of Service
OASIS	Organization for the Advancement of Structured Information Standards

ABSTRACT

Web Services have become one of promising technologies, and is getting widely adopted in business and also gradually deployed in real customer environment. The technology of Web Service is capable of providing a means to integrate different functional components over the *Internet* and enabling business entities to interact with one another through standard application program. As more and more Web Services become available on the service registry, selecting one Web Service among a group of Web Services which are offering similar functionality is a challenge. Quality of Service (QoS) is becoming an important criterion for differentiating Web services that offer similar functionality. However, the current SOA environment does not support QoS. This is because the discovery mechanism of the current SOA environment only considers the functionality (what the Web service does) of a Web service.

There are many research efforts that attempted to address these issues in QoS-aware service discovery and selection but they still leave a lot to be desired. In this research work we propose an approach that utilises both the functionality and the QoS information of a Web service during service discovery and selection. This is realizable through our proposed GUISET QoS-aware Web service Selection Broker (G-Broker). The G-Broker allows service consumers to express their QoS requirements and select the most appropriate Web service for the service consumer. It also measures the QoS information of the selected Web service to avoid false QoS guarantees. It provides support for QoS negotiation between service consumer and providers and forces Service Level Agreement (SLA) creation. Lastly, it allows monitoring of the agreed QoS between clients and providers, and therefore, detecting any QoS violation. The G-Broker provides more satisfaction to consumers and also scales well as the number of Web services published in the service registry increases.

CHAPTER ONE

INTRODUCTION

1.1 Overview

In view of the on-going IT trends and emerging technologies, our research centre has proposed a Grid-based Utility Infrastructure for SMMEs Enabling Technologies (GUISET) (Adigun *et al*., 2006). GUISET takes advantage of the affordability and popularity of mobile device technologies as well as the advancements in Service Oriented Architecture (SOA) to address the hardware and software acquisition problems for resource constrained enterprises such as the Small, Medium and Micro Enterprises (SMMEs) in rural areas particularly in Africa. To address these problems, GUISET implements a Service Oriented Architecture and hence the challenges towards realising SOA are also GUISET challenges.

The use of Web services on the World Wide Web (WWW) is expanding rapidly as the need for application-to-application communication and interoperability grows. The use of Web services provides a standard means of communication among different software applications, even if software applications are written in different programming languages and running on different operating systems. It also provides loose coupling which reduces dependencies between software applications and provide flexibility and agility.The increasing number of Web services on the web(or on a Web service provisioning infrastructure like our GUISET)implies that there will be an increasing number of Web serviceswith similar functionality. Discovering aWeb service that meets a service consumer's request among a large group of Web services that offer similar functionalities is a challenge (Ziqiang *et al*., 2007).

The literaturesuggeststhat quality QoS is a key aspect of the competitive Web services market since service consumers are faced with the inevitability of selecting the right service provider among competitors (Gang *et al*., 2009; Rafael and Carlos, 2006; Serhani *et al*.,(2005) and many more). Hence, the QoS is an essential mechanism that may be used to differentiate between those Web services which offer similar functions. Unfortunately, the current Service Oriented Architecture (SOA) environment of Web service, which is based on*register—find—bind* model, has no provision for QoS. This is so because its discovery mechanism is only based on what the Web service does (functionality)and does not consider the QoS guarantees(Ran, 2004).To address the above issues, some researchers(Eyhab and Qusay, 2007;Kambiz *et al*.,2006; Rajendran and Balasubramanie, 2010 and others) extended the current *register-find-bind* model SOA environment by introducing a middle tier component that mediates between the service consumer, the service provider, and the UDDI registry.This middle tier component is normally called a QoS-aware broker because its discovery mechanism is not only based on the functionality of a Web service but it also considers the QoS requirements specified by the service consumer.

The QoS-aware broker selects a Web service from the UDDI registry that meets the service consumer's functional and QoS requirement.The QoS-aware broker may implement various capabilities depending on each organisation's requirements. In any Web service provisioning infrastructure, there are other issues that need to be considered as Dobson (2006) suggested. These issues include QoS measurement, this is because the QoS information of a Web serviceadvertised by the service provider cannot be always trusted, therefore the QoS information of any advertised Web service needs to be measured before the service consumer binds to that service. Another issue is to provide means for enabling the service consumer to negotiate for some QoS guarantees with a service provider and sign a Service Level

Agreement (SLA) about the negotiated QoS guarantees. The last issues that need to be considered areQoS monitoring and detection of SLA violations. The QoS monitor is made to detect the real behavior of the service. Then a monitored profile is constructed to be compared to the agreements included in the SLA. However, from the literature we discovered that there is no existing QoS-aware broker that combines all of these capabilities. In this work, we propose a QoS-aware broker that integrates these capabilities: QoS measurement, QoS negotiation, SLA, and QoS monitoring.Our proposed QoS-aware broker in GUISET to provide a QoS-aware Web service infrastructure; it can also be adopted by any QoS-aware Web service infrastructure.

The rest of this chapter is organized as follows: In Section 1.2, the backgroundof the study is described. The statement of the problemthat this work addresses is described in Section 1.3. Section 1.4 gives the rationale of the study. The research questions that this work answers are listed in Section 1.5.Section 1.6 presents the research goal and objectivesof this study. The research methodologies are discussed in Section 1.7. The overall structure of this dissertation is presented in Section 1.8.

1.2. Background

The Web service technology facilitates the development of software (i.e. web applications) by allowing the integration of independently published Web services as components of a new business solution. Lee *et al*.,(2005) defined a Web service as a service module that delivers to a user a desired service through the *Internet* regardless of time, place, and platform. Web services have three enabling technologies. The first one is Simple Object Access Protocol (SOAP) that allows the exchange of messages between Web services. SOAP is used during service communication over either HTTP or HTTPS. The second is Web Service Description

Language (WSDL) that exposes the operation or the functionality of the Web service to the service client. It can be just an ordinary standalone application or any other Web service. The third enabling technology is the Universal Description Discovery and Integration (UDDI) registry. This is a registry or repository, which enables service providers to register their Web services, and clients to search for Web services that will help them to accomplish their business objectives.

GUISET adopts the Service Oriented Architecture in a way that could address some of the digital divide issues in Africa. In Africa a typical small, micro, and medium-sized enterprises (SMMEs) are facing a number of challenges namely:

1. Lack of sufficient and good quality information to inform decision-making on used of technology and market characteristics restricting deployment of technology.
2. Non-use of appropriate business strategy.
3. Limited knowledge of e-commerce technology.
4. Lack of appropriate infrastructure for the deployment of some technology.

GUISET serves as an infrastructure for addressing some of these challenges. GUISET is an infrastructure for deploying centrally owned resources such as software services for a subscription. The conceptualization of GUISET is based on the idea that affordable technology is there for these SMMEs. The GUISET infrastructure provides a platform where service providers can advertise their Web services. Service providers in our GUISET can be a single SMME or a group of SMMEs, as well as the service consumers. The proposed GUISET infrastructure is composed of three layers, namely (i) Multi-Modal Interface, (ii) Middleware layer, (iii) Grid infrastructure layer. Our work focuseson the Middleware layer which deals with managing and sharing of resources and selecting requested Web services. The full details of GUISET's layers (Multi-Modal Interface,Middleware layer and Grid

infrastructure layer) are presented in Chapter Two and not discussed any further in this Chapter.

Despite the major success of Service Oriented Architecture with regard to adoption, it is still facing some challenges. These include the fact that, in a dynamic and service-rich environment like GUISET, the potential number of Web services with closely related functionalities can be extremely large. It would, therefore, be a challenge to select a Web service among that number of Web service which exhibit similar functionality. This challenge, therefore, exists for the Middleware layer of the GUISET infrastructure. Differentiating between Web services that share similar functionality is significantly achieved by examining quality of service Web service. Unfortunately, the current SOA does not support Web services based on non-functional (QoS) requirements of a service consumerZigiang*et al*.,(2007). The Web service discovery of the current SOA environment selects Web services based on their functionalities only.To address this limitation Tain*et al*.,(2003) proposed a middle tier component that will mediate between the client, service provider and the UDDI registery. This middle tier component is also referred to as broker/dispatcher.The introduction of the broker that considers QoS requirements during Web service discovery helps in providing a finer search where there are multiple Web services providing similar functionality.

A typical QoS broker, like the one in Figure 1.1, is a service capable of performing the following functions:

- Accepting incoming requests from customers.
- Maintaining information about the resources and services rendered by Web services' providers.

- Finding, through the UDDI registry, the set of QoS-aware Web services able to deliver the customer required service.

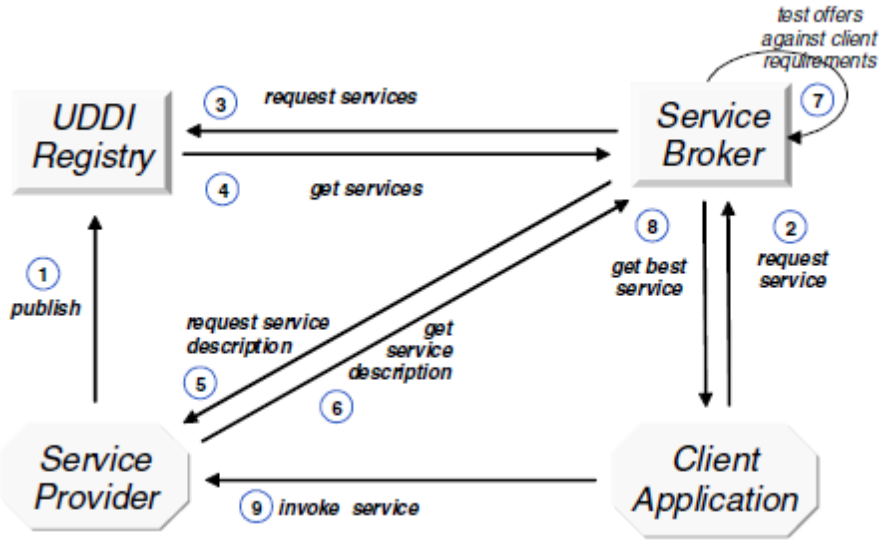

Figure 1.1 QoS Broker Tian et al., (2003)

- Selecting the most appropriate Web services able to satisfy the customer required QoS.
- Notifying customers of the approval or rejection of their request.

On top of those listed functionalities of the QoS-aware broker, there are other capabilities that a QoS-aware broker needs to implement in QoS-aware Web service selection infrastructure.According to Dobson (2006), QoS measurement, QoS negotiation, QoS monitoring and SLA need to be considered in any QoS-aware Web service selection. This study investigated some of proposed QoS-aware brokers (Ziqiang*et al.*, 2007; Rajendran and Palassubramanie, (2010); Serhani *et al.*, 2005;Rafael and Carlos, 2006and many more); none of them combine the four capabilities. Therefore, the main focus of the study is to examine the feasibility of a QoS-aware broker that can combine these four capabilities(QoS measurement, QoS negotiation, QoS monitoring and SLA), thereby enabling a selection QoS-

aware Web services that will satisfy the consumer.This proposed QoS-aware broker would be used in our own Web service provisioning infrastructure GUISET (Adigun*et al*., 2006).

1.3. Statement of the problem

Currently, the Middleware layer in the SOA environment lacks the capacity to select the most appropriate Web service from two or more equally likely candidates(Ziqiang*et al*., 2007). Existing literature suggests that Web services differentiation can be achieved by considering the quality of service propertiessuch as Availability, Accessibility, Integrity, Performance, Reliability, Regulatory Compliance, Security, Capacity, Response Time, Accuracy, Scalability and Reputation. Therefore, this work will explore the possibility of providing a middleware support service for selecting a Web service based on a QoS supplied by the consumer.

Recently, researchers have been paying attention to the importance of QoS Measurement, QoS Negotiation, QoS Monitoring and Service Level Agreement. A number of QoS-aware services selection frameworks already include some of these capabilities based on their framework requirement.However, none of the existing frameworks incorporates all of these capabilities. A mechanism to measure, negotiate, and monitor QoS in the context of a Service Level Agreement is just what GUISET needs. First, a GUISET client will rely on the mechanism to extend its functionality. In other words the middleware layer provides for Clients to be decorated with capabilities encapsulated in the mechanism. Second, this act of decorating a client dynamically with QoS measurement, QoS negotiation, QoS monitoring and SLA makes GUISET unique.

1.4. Rationale with the study

This research work contributes to GUISET and also to Service Oriented Architecture. GUISET is a Grid based infrastructure to enable on-demand service provision to SMMEs. The increasing number of Web services which exhibit similar functional characteristics in any Web service provisioning infrastructure, makes it challenging to select one Web service that meets the service consumer's request. To overcome this challenge quality of service is used as a mechanism to differentiate those Web services, which exhibit similar functionality. But the current SOA discovery mechanism does not support QoS. Many researchers propose a QoS-aware broker which its discovery mechanism does support QoS. In any QoS-aware Web service selection environment, QoS measurement, QoS negotiation, QoS monitoring and SLA areimportant (Dobson, 2006). However, none of the existing QoS-aware brokers combine all these four capabilities. Therefore, this research work was an effort to incorporate these capabilities in to a single QoS-aware broker to ensure that the service consumers bind to Web services that will meet their QoS requirements as requested.

1.5. Research questions

An investigation of the existing approaches identified the following issues which this research addresses:

1. How can we create a consumer-centric framework that selects services based on the interest of the consumer?

2. Which capabilities must the GUISET client have in order to perform QoS-aware service selection?

3. How much effect does our consumer-centric framework has towards performance.

1.6. Research Goal and Objectives

a. Research goal

The goal of this project was to design a GUISET middleware layer capability as a generic framework service for QoS-driven Clients.

c. Objectives

The above goal was formulated as an equivalent of some lower-level objectives, which wereto :

i. find aconceptual technique that is used to express QoS properties,

ii. design a Middleware layer capability (Service) for QoS-driven requests for services , and

iii. craft and evaluate a Web service that finds and binds to the most appropriate GUISET Service or Resource.

1.7. Research methodology

This study employed the design and creation research approach. Therefore, the primary method used was model formulation and this was assisted by literature analysis and model prototyping.

a. Literature Survey

Literature survey was a useful secondary method. This theoretical aspect of our research work included conducting an extensive literature search for existing QoS-aware Web service selection frameworks. During the survey, clarity was sought from existing proposed approaches to implementing a QoS-aware Web service selection framework using a Service

Oriented Architecture (SOA) approach. Knowledge gained from this survey was utilized in the evaluation of QoS-aware Web service selection approaches, and also for the formulation/identification of metrics to use during evaluation of the proposed model.

b. **Model Formulation**

This being a design and creation research, requires that the study uses some form of model formulation. This method involved an analytical approach to evaluating what has already been achieved in Web service provisioning environment research. The analysis outcome provided the baseline wisdom needed for model formulation through implementation of a prototype. The formulated model was then used as a solution approach to fill the gaps which were discovered from the existing QoS-aware Web service selection frameworks

c. Prototyping

Another secondary method used in this study was prototyping. A prototype was developed based on the formulated model. The developed prototype performs QoS-aware Web service selection and performs QoS Negotiation, QoS Measurement, QoS Monitoring and Service Level Agreement (SLA) as it is required in a QoS-aware Web service provisioning infrastructure. The implemented prototype was evaluated as a proof of concept.The prototype was compared with a QoS-aware Web service framework that does not have the four capabilities mentioned above, using appropriate performance parameters as a way of evaluating the two frameworks.

1.8. Dissertation Outline

The remainder of the dissertation is organized as follows: In Chapter Two, we explained foundation concepts of distributed systems that this research work builds upon. We were particularly interested in the Web service provisioning infrastructure. Chapter Three reviewed literature related to the key issues addressed in this study.These include QoS-aware Web service provisioning and related frameworks that enable the selection of Web services by not considering only functional aspects of Web services but also considering the QoS constrains. In Chapter Four, we presented the QoS-aware broker architecture proposed in this study. Next in Chapter Five, we presented the simulation and results analysis of the system architecture then in Chapter Six we provided a summary of our research and give suggestions for future work.

CHAPTER TWO

CONCEPTUAL OVERVIEW

*2.1.*Introduction

This chapter presents a background of key concepts in Service Oriented Architecture and QoS-aware Web service discovery. Web servicesplay asignificant role in building distributed applications on the *Internet* today. But the increasing number of the published Web services makes discovering one Web service to be considerably difficult. The existence of Web services which are offering similar functionality makes the selection of a given Web service challenging. A mechanism toenable Wed service discovery and selection based on the service consumer's QoS requirement is needed.

Section 2.2, presentsa brief overview of the Service Oriented Architecture (SOA); while Section 2.3 discusses the Web service discovery process. QoS with Web service discovery and a technique for storing QoS information in a UDDI registry are discussed in Section 2.4. An analysis of brokers in an SOA environment towards achieving a better QoS-aware Web service provisioning environment is given in Section 2.5. Section 2.6 discusses the basic concepts of our proposed GUISET infrastructure and its challenges. Section 2.7 concludes this Chapter.

2.2 Service Oriented Architecture (SOA)

The growing demand for increasing the flexibility of the IT infrastructure to support rapidly evolving business needs, has led to a rising interest in Service Oriented Architecture (SOA) (Egambaram *et al*., 2008). SOA can be defined as an architecture that separatesfunctions into differentcomponents called Web services, these separated components can be combined to

achieve a particular business application. Web services can be implemented using any technology or any level of complexity and are therefore autonomous. Web Services are a solution to the heterogeneity and the platform dependence problem of distributed computing.SOA enables Web Services to be published by Web Service providers and to be discovered by Web Service consumers. A Web services application uses these open protocols namely:

1. Simple Object Access Protocol (SOAP)
2. Hyper Text Transfer Protocol (HTTP)
3. Extensible Makeup Language (XML)

These protocols are built to enable machine to machine interaction over a network.

The current Web Services architecture encompasses three roles: (Serhani *et al*., 2005):

(i) **Web Service provider**is an entity that builds and publishes interfaces for the Web services they offer and also reacts to requests for utilizing its Web services.

(ii) **Web Service consumer**is an entity that finds and utilizes the Web services published by the service providers for their business needs.

(iii) **Universal Description, Discovery and Integration** (UDDI) registry it is a repository that stores the interfaces of the published Web services.

Figure 2.1 depicts the interaction between a service consumer, service provider and registry. The service provider creates a Web service which offers a particular functionality or business operations. These service providers can be organizations or companies.

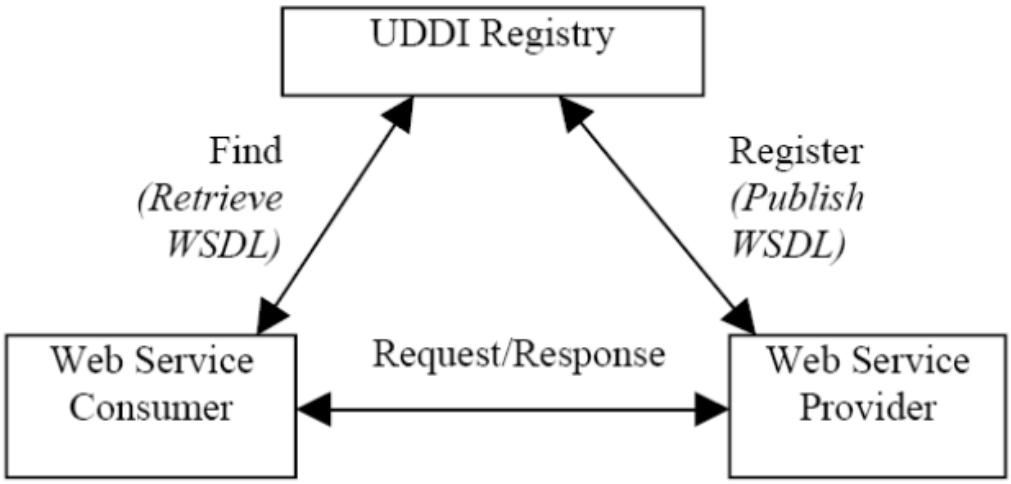

Figure 2.1: The current Service Oriented Architecture Zigiang et al.,(2007).

The service provider creates an interface that describes the Web service, which facilitates the Service Discovery and composition of the Web service. The interface of the Web service is described in a Web Services Description Language (WSDL) file. Service consumersdescribe their functional requirements first, if they need a Web service. Thenthe Web service requests which are sent by the service consumersincludethe functional requirement of the Web service. The Web service functional requirements are mostly contained in a WSDL. The service registry (UDDI) is used by service providers to publish/advertise their information about the Web services they offer. The service consumer searches the service registry to find the kind of a Web service they need. The UDDI is a registry standard for storing the interface description of the Web Services. The implementation details of the Web Service are completely hidden and the service consumer requires no further information about the Web Service except knowledge of the interface (WSDL) document. This model allows a very loose coupling between service consumers and providers thereby improving interoperability.

Web Services bring the benefits of service-oriented application to the Web and they are the dominant implementation technology for SOAs.

Web Services are described using WSDL and published in a UDDI registry so that they can be discovered and used by Web Service clients. The elaborations of these operations are as follows:

1. **Publish:** the Web service providers register/advertise their Web service information in the service registry for Web service requesters to locate the advertised Web services.
2. **Find/Discovery**: The Web service requester searches the service registry to find Web services that matches their Web service requirements. The discovery process presents a list of Web services that matches the service consumer's Web service requirements.
3. **Bind**: After the discovery process, the discovery system will provide a list of service providers that offers the functionality of the Web service requested by the service consumer. The service consumer will select one service provider from the list. Then the service consumer will bind to selected service provider using the SOAP (Rajendran and Balasubramanie, 2009).

The Web Service Description Language (WSDL) it is a standard language used to describe more information about the Web service. This information includes the operation (methods) that the Web service performs it also defines the data type being transmitted and the location of the Web service (Walsh, 2002).

2.3. Web Service Discovery

A Web services discovery process is defined as an act of finding a machine readable interface of a Web service. This interface contains the functionality (what a Web service does) of the

Web service that meets the service requester's functional requirements (Ziqiang*et al*., 2007). The aim of the web service discovery is to locate an appropriate Web service that matches a service requester's requirements.

A provider agent or consumer agent is used to assist the discovery mechanism by finding the Web service that matches the service requester's Web service requirement.The three leading approaches on how a discovery system should be implemented include: service registry, peer-to-peer (P2P) system and an Index. Another service discovery system is called an index which represents a collection of Web services that are published by service providers. The information in the index service discovery system is neither centrally controlled nor authoritative. A company or an individual can form their own index, which assembles information of Web services advertised on the World Wide Web (WWW) frequently using web spiders. The disadvantage of the index is that the information may have expired, but the good thing about it is that the expired can be verified before anyone uses it.

Peer-to-Peer (P2P) is defined as a fully decentralized and also interoperable discovery service that has a full semantic-level-matching. TheP2P discovery service is known as a more reliable registry approach because it does not use a centralized registry.The disadvantage of a P2P discovery service includes high performance costs (Ziqiang*et al*., 2007). The characteristics of a registry are that it is centrally controlled and authoritative repository of Web services information. A Web service is published by a service provider to the service registry before it is discovered and be used by service consumers. The authority to publish a Web service and update its information is controlled by the owner of that particular service registry. No other company is allowed to publish Web services or update the information of Web services that are provided by a different company. UDDI is a very good example of the service registry approach. Among these approaches of Web service discovery, our work uses the registry approach, because it is widely used approach (Chenliang *et al*., 2004).

2.3.1.UDDI Service Registry

The Universal Description, Discovery, and Integration (UDDI) registries are intended to become the world-wide lookup mechanism for web-services.

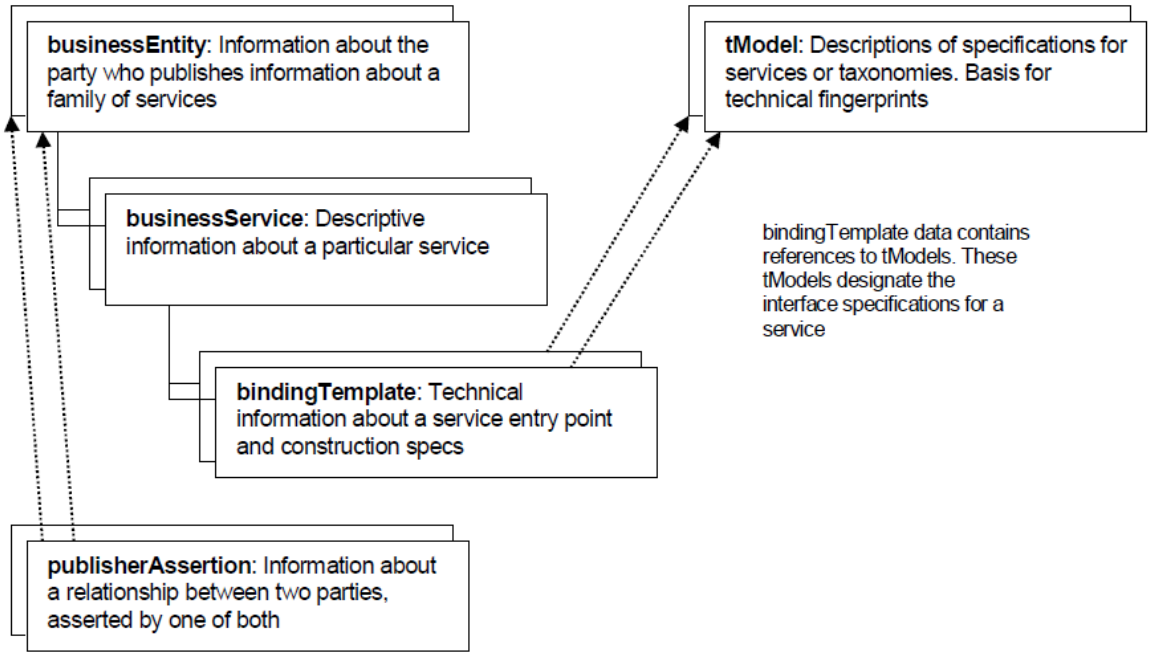

Figure 2.2: UDDI core data structure UDDI(2002)

As such, the registry has to provide high throughput, low response times, high availability, and access to accurate data. The UDDI registryis among the other Web services discovery mechanisms discussed above (Chenliang *et al*., 2004). The UDDI registry isthe leading mechanism used as repository to store the Web services' information.Service providers register their service with the UDDI registry. This registry is used by service requestors to find services that match certain criteria. These service requesters are normally business or individuals who find the Web services by searching either public or private service registries. A typical example of a service provider, can be an airlines company that publishes their fare Web services to a service registry (UDDI). Then an example of a service consumer can be a travel agency company that uses the UDDI registry to find the Web services that are provided by different airline companies, and bind to one Web service that best matches their service

requirements. The information that is stored in the UDDI about the Web service includes the business description of the Web service, the name of the service provider, the functionality that the Web service offers and technical information on how to access and administer the Web service. Figure 2.2 illustrates the four data structures that exist in a UDDI registry namely: *businessEntity*, *businessService*, *bindingTemplate* and *tModel*.

- *BusinessEntity* illustrate all the information of the service provider that has published the Web service.
- *BusinessService* describes the functionality that the Web service performs.
- *BindingTemplate* provides the technical details of a single Web service.
- *tModel*describe categorizations and specifications of a Web service.

2.4. Quality of Service (QoS) and Web Service Discovery

Web service discovery and selection is a very important part of Service-Oriented Architecture. The Service Oriented Architecture follows the find-bind-execute conceptwhereby the Web service providers can register their Web services in Web service registries.Then the service consumers locate Web services from the Web service registry as shown in Figure 2.1. The Web Services technologyis now widely used because of its benefits such as loose coupling, heterogeneous services, interaction and dynamic sharing. However, most of today's Web Service implementations do not guarantee the level of service quality delivered to their consumers.The Quality of Service (QoS) proves to play a very important part in selection of Web service that can meet the service consumer's level of service quality delivery in Service Oriented Architecture (SOA) (Rajendran *et al*., 2010).Unfortunately, the current Web service registry of the SOA cannot store QoS information of Web service (Rajendran *et al*., 2010). Web Service Selection denotes the action of selecting one Web service among a number of Web services which provides the required functionalities.When

there are many Web services published in a particular service registry it is possible to find a number of Web services providing the same functionality but with different QoS parameters. In this case differentiatingone Web service from each other becomes very easy and a service consumer can be able to request a Web service that meets hisQoS requirement(s) (i.e. select the service with minimum cost or with better response time). Quality of Service (QoS) is a collective of effects of service performance which determine the degree of satisfaction of a user of the service. (In simple form), the quality of service of a system describes the system's ability to provide the desired quality of a given service at the given time and given price that the user is willing to pay (Dobson, 2009).

Thereare a number of QoS parameters which one can consider; but Performance Dependability, Price and Reputation are the QoS parameters which are usually relevant for Web Services (Mareno and Mirandola, 2010).

- **Performance:**This factor includes*response time, execution time, latency, transaction time and throughput. Response time* is the amount of time it takes to complete a single Web service request. *Execution time* is the amount of time that it takes to complete a sequence of processes. *Latency* is the time between sending a Web service request and the time when the request arrives at its destination (the time that a message spends travelling from one node to another). *Transaction time* is the time it takes to complete a single transaction in a request. *Throughput*is the number Web service requests completed in a given time interval.
- **Dependability.** The dependability of Web Services is a property that integrates several attributes that include: *Availability* which represents the amount of time in which a particular Web service is ready to be used. And*Reliability*signifies the capability of a Web Service to deliver the required functions in a particular time interval.

- **Price.** Representsthe amount of money that each Web Service costs. The price of a Web servicesometimes is fixed for each invocation or it can change based on the demand of that particular Web service. In most cases the Web service providers which provide faster and more reliable Web services, their Web services may cost more.

- **Reputation.** Web service reputationreveals a common perception that other service customer have towards that particular Web service. Basically Web service reputation is the ratings given by service consumers about a particular Web service.

2.4.1. Storing QoS information in the Service Registry (UDDI)

The UDDI registry has a tModel as one of its features. The function of the tModel is to describe the technical information of each Web service. A tModel (short for technical model) is a data structure representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system.The tModel keys in a service description are in a technical fingerprint that you can use to trace the compatibility origins of a given service. They provide a common point of reference so that you can identify compatible services. *tModels* are used to establish the existence of a variety of concepts and to point to their technical definitions. *tModels*that represent value sets such as category, identifier, and relationship systems are used to provide additional data to the UDDI core entities to facilitate discovery along a number of dimensions. This additional data is captured in *keyedReferences* that reside in *categoryBags, identifierBags,* or *publisherAssertions*. The *tModel* also register categorization. This categorization can be used as a mechanism for storing additional information about the Web service to a UDDI registry (Blum and Fred, 2004). Blum and Fred (2004) suggestedthat the QoS information of a Web service can be stored inside UDDI registries by using the categorization tModel.

Blum and Fred (2004) defined a tModel called *QoSInformation* that references an external quality of service web service. A resource at the *tModel's overviewURL* (such as a generated XML file) would contain all of the performance and reliability data (QoS properties). Each UDDI *bindingTemplate* contain a tModel reference to the *QoSInformation* tModel, added to its tModelInstanceDetails collection.

To illustrate how the QoS information is stored in the service registry (UDDI),Blum and Fred (2004) presented the *bindingTemplate* (XML file) which references the *tModel*. The referenced *tModel* has the QoS parameters categories. The *bindingTemplate* example that references the *tModel* is illustrated in Figure 2.3. A Stock Quote Web service that has some QoS information is used in the example. The*bindingTemplate*contains the *tModelkey*of the referenced *tModel,* each tModel is identified by its unique *tModelkey*.

Blum and Fred (2004), then presented the *tModel* which is referenced by the *bindingTemplate.* Figure 2.4 illustrate that tModel. The tModel has the tModelKey which corresponds to the tModelKey that is in the *bindingTemplate* that references the *tModel*. The QoS information of the Stoke Quote Web service is specified in the *categoryBag* that is inside the *tModel*. Average Throughput, Average Response Time and Average Reliability are the QoS parameters used in this example.

```
<businessService
 serviceKey="uddi:mycompany.com:StockQuoteService"
 businessKey="uddi:mycompany.com:business>
  <name>Stock Quote Service</name>
  <bindingTemplates>
    <bindingTemplate
              bindingKey="uddi:mycompany.com:StockQuoteService:primaryBinding"
              serviceKey="uddi:mycompany.com:StockQuoteService">
      <accessPoint URLType="http">
            http://location/sample
      </accessPoint>
      <tModelInstanceDetails>
            <tModelInstanceInfo
                  tModelKey="uddi:mycompany.com:StockQuoteService:Primary
                  Binding:QoSInformation">
                  <description xml:lang="en">
                        This is the reference to the tModel that will have all of the
                        QOS related categories attached.
                  </description>
            </tModelInstanceInfo>
            <tModelInstanceInfo
                  tModelKey="uddi:mycompany.com:StockQuoteService:Primary
                  Binding:QoSDetail">
                  <description xml:lang="en">
                        This points to the tModel that has the reference to the
                        web service endpoint that allows detailed retrieval of
                        information
                  </description>
            </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
</businessService
```

Figure 2.3: QoS information on BindingTemplate (Blum and Fred, 2004)

```
<tModel tModelKey="mycompany.com:StockQuoteService:
PrimaryBinding:QoSInformation"" >
  <name>QoS Information for Stock Quote Service</name>
  <overviewDoc>
    <overviewURL>
      http://<URL describing schema of QoS attributes>
    <overviewURL>
  <overviewDoc>
  <categoryBag>
    <keyedReference
            tModelKey="uddi:uddi.org:QoS:ResponseTime"
            keyName="Average ResponseTime"
            keyValue="fast" />
    <keyedReference
            tModelKey="uddi:uddi.org:QoS:Throughput"
            keyName="Average Throughput"
            keyValue=">10Mbps" />
    <keyedReference
            tModelKey="uddi:uddi.org:QoS:Reliability"
            keyName="Average Reliability"
            keyValue="99.9%" />
  </categoryBag>
</tModel>
```

Figure 2.4: QoS information represented in a tModel Blum and Fred (2004)

There are many researches including (Menascee*t al*., 2007; Badidi*et al*., 2006) and many more) who adopted this technique and used the technique as it is without changing it. Marc (2009), stated that the Blum and Fred (2004), proposed technique for storing QoS information in a UDDI registry is a straightforward and widely used technique.

2.5 Broker

Due to the increasing number of Web services offering similar functionalities; non-functional properties (QoS) have become essential criteria to enhance the selection process of services. Because of the increasingof number of Web services in the Web, there are many Web services which offer similar functionality. Then selecting one Web service from a set of Web services which offer similar functionality is a challenge. Gang *et al*., (2009) stated that the QoS information of each Web service can be used to differentiate these Web services, which make the Web service selection to be easy. The current SOA UDDI registry (Figure 2.1) does not have the capability to store QoS information. Blum and Fred (2004), proposed a widely used technique that is used to store QoS information in a UDDI registry as discussed in Section 2.4.1 above.Another limitation of the current SOA environment is that its discovery mechanism is based on the functionality (what the Web service does). The current SOA environment does not support QoS. Other challenges of SOA presented by Egambaram(2008) are:

(1) It makes the selection of a desired Web service more complex for the consumer.

(2) If new selection algorithms should be introduced or old selection algorithms should be updated, all consumers have to be updated.

(3) Consumer gains inside knowledge of the Web Service and the Web Service's network structure.

(4) Every consumer makes its own decision; therefore, this architecture cannot be used for Web Service load balancing.

(5) No centralized disqualification of unreliable Web Service providers.

(6) The current registry does not support Quality of Service.

(7) Its discovery mechanism is only based on what the web service does without considering Quality of service (QoS) guarantee.

To address the above issues many researchers such as Gang *et al*., (2009); Gwyduk *et al* (2009); Serhani *et al*., (2005) and many more, propose a middle tier component that would mediate between the client, service provider and the UDDI registry. These researchers extend the normal SOA environment presented in Figure 2.1 above. The middle tier component must be QoS aware. This middle tier component is also referred to as broker/dispatcher and is commonly called Broker. The Broker is delegated to make intelligent service selection decision for business process requests.

2.4.1. Typical Broker functionalities

Tao and Kwei(2005) defined some of the main functions of a Broker. These functions include Service tracking, Dynamic service composition, Dynamic service selection and Dynamic service adaptation:

- Service tracking: This functionality means that a broker is able to store most of the frequently and recently requested Web services.
- Dynamic service composition: This functionality means that a broker is able to build, update and maintain a number of predefined business processes.
- Dynamic service selection: This functionality is the main functionality of a Broker. A Broker able to discover and selects a Web service that meets both the functional requirement and QoS requirements of a Web service specified by a service consumer.
- Dynamic service adaptation: With this functionality a broker can reconstruct a process when it has failed.

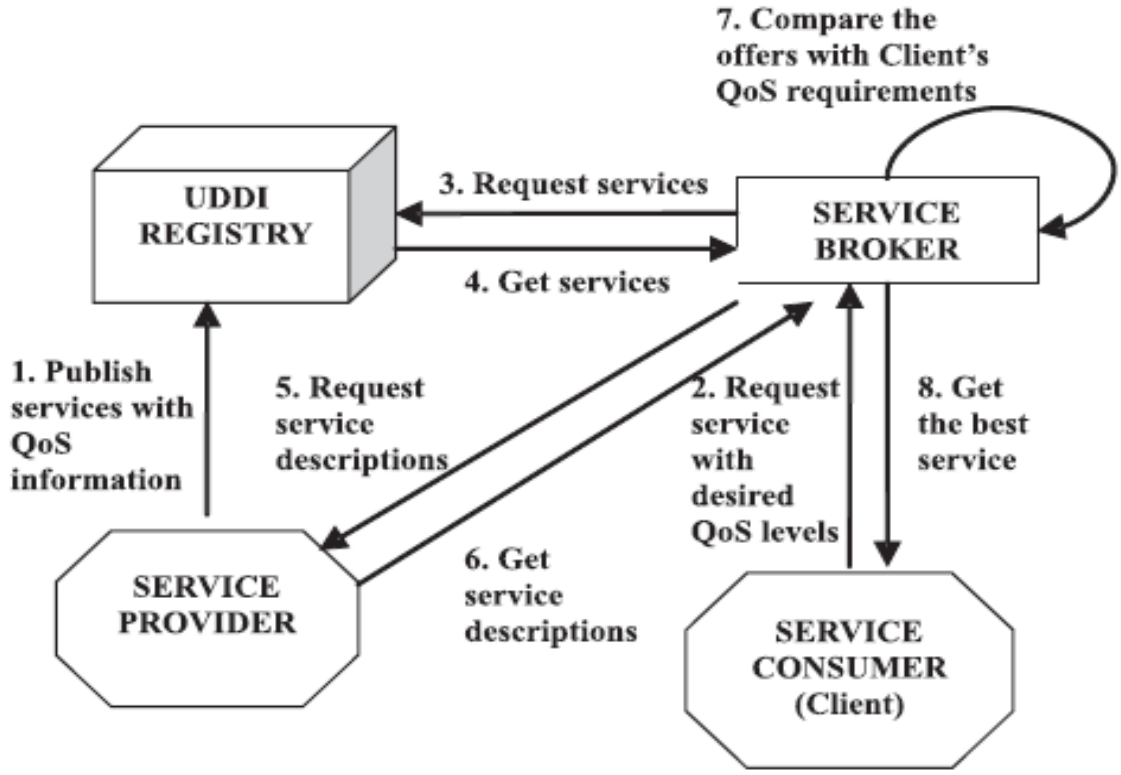

Figure 2.5: QoS-aware Broker Yu et al., (2008).

The focus of our research is on QoS-aware Broker for dynamic Web service selection. Yu *et al*., (2005) present a typical QoS-aware Web service selection broker, shown in Figure 2.5 below. A typical QoS-aware broker's functionalities can be; receiving in coming Web service request form service consumers, the request contains QoS requirement of that particular service consumer.Select a Web service that meets the service consumer's QoS requirements from the UDDI registry.After selecting that Web service from the UDDI registry, the broker can then contact the provider of that selected Web service, to get more information about the Web service including information about how that Web service can be accessed.After getting the required information the broker passes this information to the service consumer so that the service consumer can bind to that Web service.

In a QoS-aware Web service provisioning infrastructure Dobson (2009), suggests that there are other issues that need to be considered, these include: QoS measurement, QoS monitoring And QoS Negotiation and Service Level Agreement (SLA).When a Web service consumer requires a Web service that can meet his QoS requirements needs, to simply consider the QoS guarantees that are advertised by the service provider is not advisable. This is because some Web service providers have a tendency of advertising false QoS guarantees. Therefore, to avoid the problem of having some service provider publishing false QoS guarantees, the actual QoS guarantees being provided by any Web Service provider should be verified every time. Therefore QoS measurement is another function that a Broker performs. There is also the need for QoS negotiation; the aim of which is to allow service consumers to negotiate with the service providers for some QoS parameters. During the QoS negotiation process the information from the negotiating parties is put to the negotiation process as input. QoS monitoring is another function that a Broker supports.QoS monitoring of Web service is very essential as a tool to detect any kind of violation of the agreed QoS information. Among the existing QoS-aware brokers, none of them does combine these capabilities together. In our GUISET infrastructure, this is also a Web service provisioning infrastructure. We want to propose a QoS-aware broker that can combine these capabilities. The goal of GUISET infrastructure is to support Web service provisioning infrastructure in which all the above mentioned capabilities are combined.

2.6. GUISET, Approach to QoS Awareness

GUISET is the acronym for Grid-based Utility Infrastructure for SMME-enabling Technologies. GUISET is an on-going research conducted by the Department of Computer Science at University of Zululand(Adigun *et al*., 2006). The conceptualization of GUISET is

based on the idea that technology that is affordable is there for SMME; what we need to do is, find an appropriate strategy to make affordable technologies available using the utility approach to service delivery (Adigun *et al*., (2006). In other words, we envisage a future in which service providers will competitively provide computing services at variable pricesbased on their quality of service requirements, instead of the current fixed price options..The opportunities provided by this project include: (1) making technology diffusion have more direct impact on the community that is being assisted; (2) uplifting the community to the status of becoming part of the Mobile business value network; and (3) leveraging economic benefits of mobile technology such as access to market information, new business opportunities and staying within the reach of customers and business contacts (WITFOR, 2007).

GUISET is an enabling platform for the SMMEs to access ICT services without owning the infrastructure on which the services are deployed. The GUISET services are distributed and GUISET middleware allows the virtualization and access to the services. In this context, users (providers, SMMEs and customers) with different capabilities are able to use services without owning the infrastructure and knowing the service provider. Service providers deploy services. SMMEs subscribe for services that help them in their business. End users do not interact with the GUISET infrastructure only SMMES do.

2.6.1. GUISET Challenges

There are a number of challenges that still need to be addressed in this GUISET infrastructure. These challenges include security issues, service composition and context-ware operating environment.

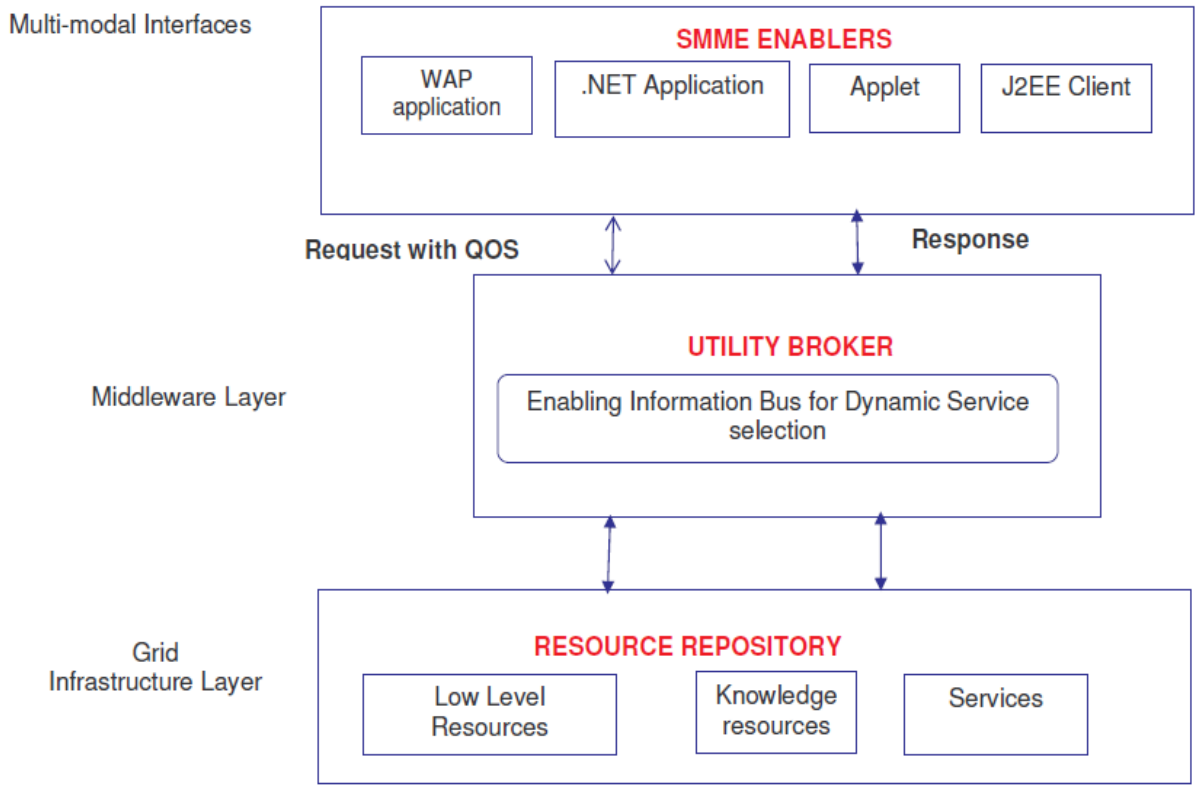

Figure 2.6: Grid-Based Utility Infrastructure for SMME-Enabling Technologies Adigun et al.,(2006).

Questions to be answered under security are which groups of components are interacting and what sort of vulnerability will arise as a result of interface techniques, communication styles, and Component implementation model? Questions that need to be addressed under service composition are: What interfaces would services expose to users? Can composite services be composed from other high level services? How would sharing of composite service be supported? What sort of billing protocol would be well-suited to the need of the envisaged infrastructure? The brokerage and context-aware operating environment also has some questions to be answered, how QoS-aware service components would be coordinated? What

sort of protocol will be used? How would the service components interface? How would quality of service be guaranteed (Adigun *et al*., 2006)?

The research focuses on how Quality of Service (QoS) would be guaranteed. Figure 2.6 shows the proposed GUISET infrastructure. The GUISET infrastructure consists of three layers, the Multi-model interface layer which deals with rendering information to the user, the Middleware layer which deals with managing the sharing of resources and service selection, the last layer is the Grid infrastructure layer which hosts the resources and web services. Therefore, this research focuses on enhancing the middleware layer by proposing a mechanism that would enable Web service consumers to specify their QoS requirement when requesting a Web service. The Utility Broker in Figure 2.6 is envisioned to support QoS aware service selection, QoS measurement due to the fact that some web service providers may be tempted to publish false QoS information. QoS negotiation and Service Level Agreement (SLA) is another function that our envisioned Utility Broker must support.Our Utility Broker will also support QoS monitoring, to monitor any QoS violation by both parties during the Web service provisioning. The additional functionalities which our proposed Utility Broker must perform raise performance issues, therefore, it would be necessary to study the performance of our Broker and compare it with a Broker that does not perform QoS negotiation and SLA, QoS measurement and QoS monitoring.

2.7. Summary

In this chapter we have introduced and discussed basic concepts ofSOA.Quality of service with Web service discovery, Storing QoS information in UDDI registry, Brokerage in SOA and GUISET were also discussed. The idea of extending the normal Service Oriented Architecture by introducing a fourth component called Broker, which mediates between the service consumer, service registry (UDDI) and service provider, to enable a QoS-aware Web

service selection is widely adopted in the field of QoS-aware Web service selection. In the next chapter, we critically analyse the existing QoS-aware brokers. The focus will be on existing QoS-aware brokers and associated mechanisms, and how they select Web services to match QoS requirements specified by service consumers. Moreover, the chapter will also examine what other capabilities a particular broker may possess as suggested by Dobson (2006).

CHAPTER THREE

REVIEW OF RELATED WORKS

3.1.Introduction

As more and more Web services appear on the public Internet, Quality of Service (QoS) becomes one of the most important factors in Web service provisioning. This is because of the fact that with the increase in the number of Web services providing similar functionality, QoS is becoming an important criterion for differentiating those Web services from each other. Unfortunately, the current Web service environment based on the Service Oriented Architecture (SOA) does not offer comprehensive QoS support (Tian *et al*., 2004). To address the issue of QoS-aware Web service selection, many researchers extend the normal SOA environment by introducing a Broker as an intermediate component between the service consumer, service provider and the Service registry. In Chapter 2, we discussed an efficient technique used to store QoS information in a Web service registry, particularly the UDDI registry. A broker is said to be QoS-aware if it allows service consumers not only to specify the functionality of a Web service but also the QoS requirements when requesting a Web service.Dobson (2006), discovered from his survey that there are a number of mechanisms needed to be considered in a QoS-aware Web service provisioning infrastructure: includes; a QoS-aware Web service discovering and selection mechanism; a mechanism for QoS Negotiation; a mechanism for QoS measurement; a mechanism for QoS monitoring and lastly a mechanism for mechanism for Service Level Agreement creation . InSection 3.2 we review a number of proposed QoS-aware Web service selection approaches and we discuss the inclusion of the above mechanisms.

This chapter reviews the literature that inspired and supports our research. The first section of this chapter, Section 3.2, serves as a heart of our literature review chapter. In this section, we review the existing Brokers that enable QoS-aware Web service selection. Then in the other sections we review each of the mechanisms that are required in a QoS-aware Web service provisioning infrastructure as recommended in the literature, since these mechanisms must be supported by a QoS-aware broker. In Section 3.3 we review exiting QoS Negotiation and Service Level Agreement (SLA) mechanisms.Section 3.4 reviews existing QoS measurements. In Section 3.5, we present QoS monitoring techniques then lastly we give a conclusion drawn from the surveyed literatures.

3.2. QoS-aware Web Service selection frameworks

QoS is used to differentiate Web services which offer similar functionality. Thus, makes it easy to select a Web service from a set of Web services which offer similar functionality(Gang *et al*., 2009; Rafael and Carlos, 2006; Serhani *et al*., 2005 and many more). But the standard discovery mechanism of the today's SOA environment does not support QoS. There are two common approaches being used to improve the current SOA environment to support QoS during the Web service discovery and selection process. The first approach, extends current SOA discovery and selection mechanism to consider the service consumer's QoS requirement. The second approach extends the current SOA architecture by adding a fourth component known as a broker(Gang at al.,2009; and Serhani *et al*., 2005; D'Mello *et al*., 2008; Rajendram *et al*., 2010; Badidi *et al*., 2006).

3.2.1. The Extended SOA Discovery and Selection Mechanism

The extended SOA discovery and selection mechanism is an enhancement to the current SOA framework to support QoS during the Web service discovery and selection process. The discovery and selection of the current SOA frameworkis only based on functional (what the Web service does) requirements only. In a SOA environment it is likelythat two or more Web services can exhibit the same functional requirements,but with different QoS parameters(Ziqiang *et al*., 2007).In light of this view, incorporating QoS into the service discovery process becomes very important. The current service registry of the SOA framework was not built to store QoS information(Rajendran *et al*., 2010). The first step in this approach is to extend the current service registry so that it can store QoS information of every Web service published.After extending the service registry, then the Web service provider publishesits Web service to the service registry the way it is done in the current SOA framework. The service registry is where all the Web services advertised by different service provider are registered Web services with lookup facilities; the Web service consumer needs the Web service offered by the service provider. The Web service consumer sends a Web service request to the service registry, as it is in the current SOA environment. The only difference is that the request does not only contain the functional description of the Web services requested, but it also contains the QoS requirements specified by the Web service consumer. Service registry is able to discover all the Web services that meet the functional description of the Web service requested by the service consumer and it uses the QoS information specified by the service consumer to select one Web service that meets the service consumer's QoS requirements. After selecting a Web service that meets both the functional and QoS requirements description the service registry sends back all the necessary information to the service consumer to bind to the selected Web service (Ran, 2003 and Chen at al., 2003).

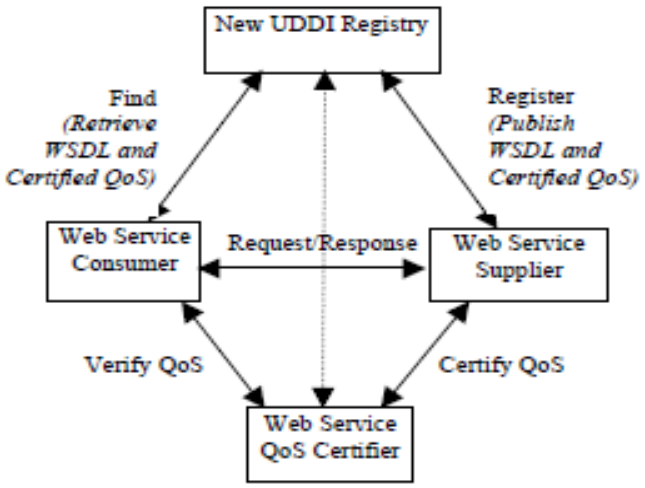

Figure 3.1:` A Web Services Registration and Discovery Model (Ran, 2003)

There are several researchers (Ran, 2003; Singhera, 2004; Chen *et al*., 2003 and (Yutu *et al*., 2004), who use this approach in their Web service discovery and selection. Ran (2003) extended SOA discovery model by adding a newcomponent called theWeb service QoS certifier, to the normal SOA model, as shown in Figure 3.1. The certifier verifies the QoS claims for a Web service before its registration. And the Web service consumer uses the Web service QoS certifier to verify the QoS certification of the selected Web service before binding to it. Apart from the new component added by Ran (2003), the model performs the Web service discovery and selection the same way the extended SOA discovery and selection does.Deng and Xing (2004) also use the extended SOA discovery and selection mechanism, whereby the service consumer sends a Web service request to the service registry and the service registry returns with a Web service that satisfies both the desired functional and QoS requirement description specified by the service consumer.

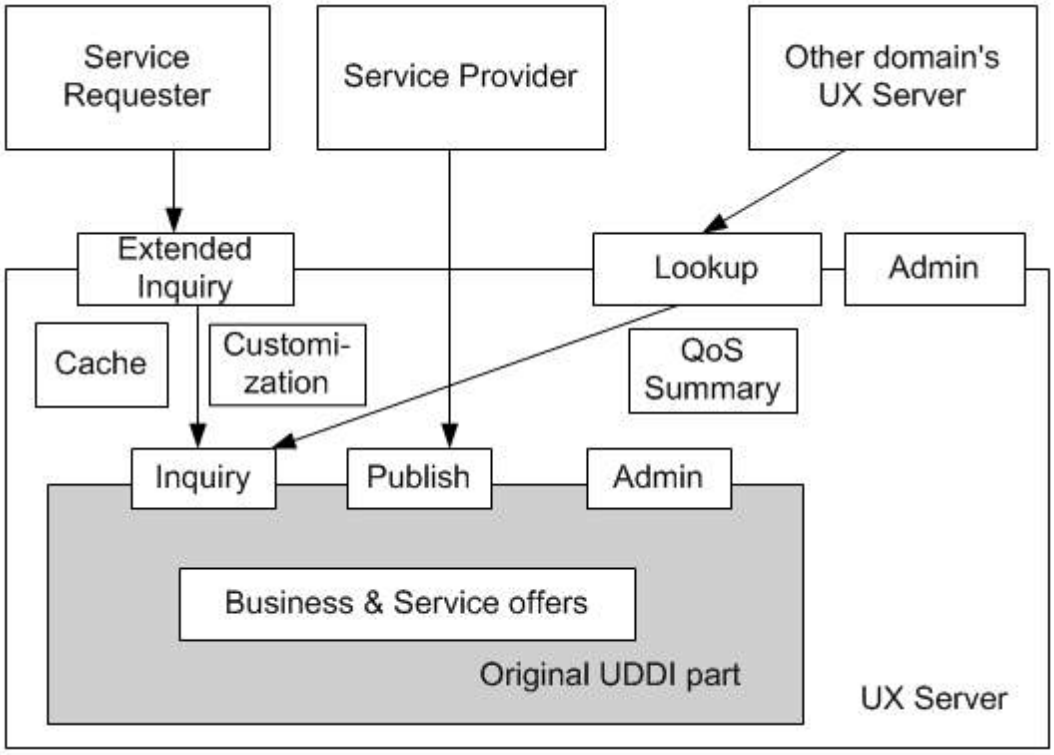

Figure 3.2: UX- An Architecture Providing QoS-aware Web service SelectionChen et al(2003)

Deng and Xing (2004) proposed the grouping of Web services inside the service registry based on functionality that they offer. Deng and Xing (2004), argue that this grouping makes discovery and selection to be faster, but theydid not conduct any experiments to prove theseclams. Chen *et al.*, (2003) also extended the Web service inquiry to include QoS information desired by the service consumer. All Web services that meet the functional requirements of service consumers are discovered.The desired QoS parameters specified by the service consumer areused to enforce a finer search. Chen *et al*., (2003) proposed model is unique in a manner that it can use other service registries if a Web service requested by the service consumer is not found, as shown in Figure 3.2.Zafar (2004) and Yutu *et al*., (2004) also used an extended SOA model. They added a fourth component called the Web service monitor to the normal SOA model. This component is used to store compliments or

complains made by the service consumers about QoS of some Web services. The service registry uses these compliments and complains to aid its decision of selecting a Web service that meets the consumer's QoS requirements.

The disadvantage of extending the SOA discovery and selection mechanism is that it can result in a very complex model. The complexity would then be a problem if one wants to add more mechanisms (like QoS measurement, QoS negotiation, SLA and QoS monitoring) to the extended QoS-aware Web service selection framework. Using the model proposed by Ran (2003), shown in Figure 3.1 above as an example, we see that Ran (2003), added a fourth component to the normal SOA model, the added component is used to verify the QoS information published. Every service consumer needs to find this component to verify the QoS guarantees of the selected Web service. In Ran (2003),adding QoS negotiation, SLA or QoS monitoring would result in a very complex model, since three extra components need to be added to the existing ones. Complexity is what we want to reduce in our GUISET QoS-aware Web service selection infrastructure.

3.2.2. The Broker Approach

Several frameworkswere proposed in an effort to address the issue of QoS-aware Web service selection.Eyhab and Qusay (2007); Kambiz *et al*., (2006); Rajendran and Balasubramanie (2010), extended the current *register-find-bind* modelby introducing a middle tier component that mediates amongthe service consumer, theservice provider, and the service registry.

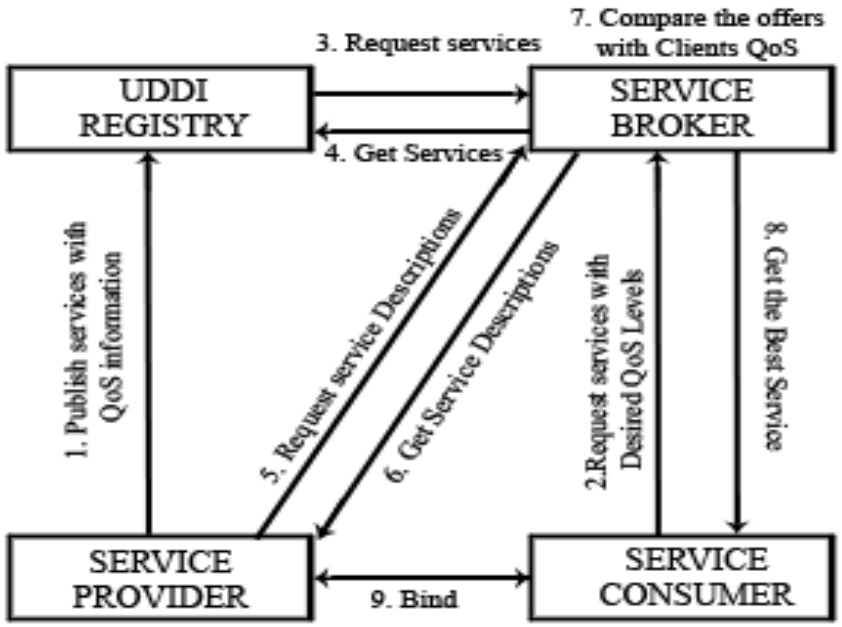

Figure 3.3: QoS-aware Web service operations (Rejendran and Balasubramanie, 2009)

This middle tier component is normally called a QoS-aware broker becauseits discovery mechanism is not only based on the functionality of a Web service but it also considers the QoS requirements specified by the service consumer. The QoS-aware broker selects from the service registry a Web service that meets the service consumer's functional and QoS requirements. A typical QoS aware broker is a service which is capable of

(1) Accepting incoming requests from consumer.

(2) Finding, from the service registry the Web Services that deliver the QoS required by the client.

(3) Selecting the most appropriate Web Services able to satisfy the consumers required QoS.

(4) Notifying consumers of the approval or rejection of their request (Rejendran and Balasubramanie, 2009).

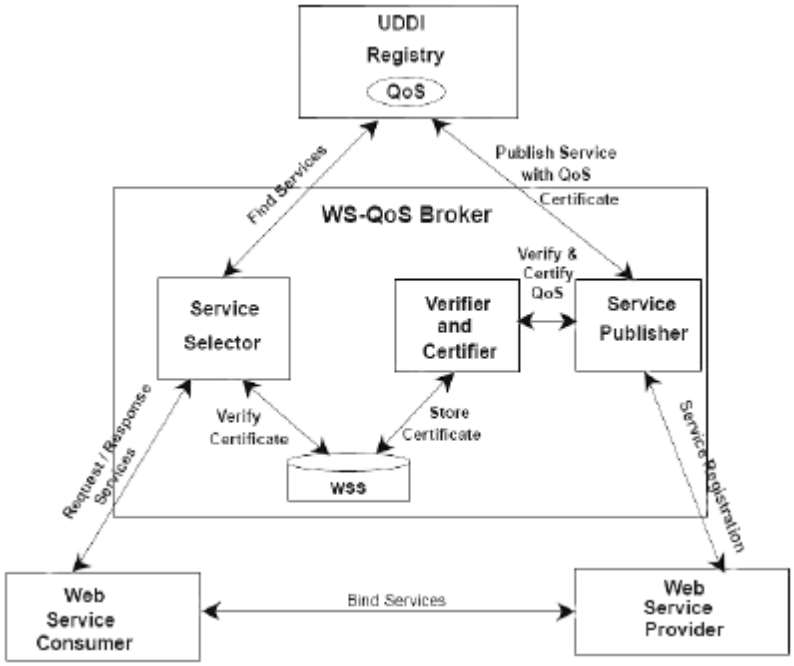

Figure 3.4: Architecture for WS-QoS Broker (Rajendran et al, 2010)

One of the works that uses the broker approach is the work of Rejendran and Balasubramanie (2009) shown in Figure 3.3. Rejendran and Balasubramanie (2009)argued that the broker approach reduces the complexity that is experienced in the extended SOA discovery and selection approach discussed above. The complexities which Rejendran and Balasubramanie (2009) stated include the discovery and selection of QoS-aware Web services which match the consumer's QoS requirements. Tavares and Westphall (2006) also used the QoS-aware broker approach in their Web service discovery and selection infrastructure. Apart from the fact that their broker can select Web services that meet the service consumer's QoS requirement, Tavares and Westphall (2006) added QoS negotiation and QoS monitoring components in their broker. But they did not give details on how the two added components in their broker work.

Rajendran *et al*., (2010) used the broker approach; they call it an efficient Web service QoS broker based architecture for Web services Selection, shown in Figure 3.4. The architecture

consists of the basic web service model components like the web service provider, web service consumer and the service registry. In extending the current SOA environment Rajendran *et al*.,(2010) added a Web Service QoS-aware mediator component called a Web Service QoS-aware Broker. The Web Service QoS-aware Broker component helpsin selecting a Web service that meets the service consumer's QoS requirement on behalf of the service consumer. Rajendran *et al*., (2010) added a component that performs QoS measurement in their broker. The added component performs measurement of the QoS information published by a service provider. This measurement ensures that service providers do not publish false QoS information; it measures the QoS information of a Web service prior its registration with the service registry. If the QoS information is correct then that service provider is given a certificatewhich assures that all the QoS information that the service provider published are correct.

Tao and Kwei (2005) designed a QoS-aware broker that selects Web services that meet the service consumer's QoS requirements and also enables QoS negotiation with any service provider if the requested QoS was not found. Apart from the fact that Tao and Kwei (2005) proposed broker selects QoS-aware Web services on behalf of service consumers, the broker can also perform QoS negotiations and it creates SLA at the end of the negotiation process. Gang *et al*., (2009) proposed a QoS-aware model for Web service discovery, this model add a fourth component in the already existing SOA components (Service provider, Service consumer and Service registry) just like in the work of Tao and Kwei (2005); Serhani *et al*., (2005;,D'Mello, *et al*., (2008). The broker proposed by Gang *et al*., (2009) has a component that keeps a record of every recently accessed Web service and the QoS information experienced during the Web service provisioning. Their broker updates this information

every time when a particular Web service is accessed. It uses the QoS information to recommend a Web service which can provide better QoS to the service consumer.

3.2.3. The State of the art in QoS-ware Web services discovery frameworks

The concept of a broker based approach to enabling QoS-aware Web service discovery is widely used in the literature. Therefore, for our GUISET infrastructure we have adopted the idea of a QoS-ware Web service selection Broker that selects a Web service on behalf of the service consumer, and the selection criteria is based on the service consumer's QoS requirements and the functionality of the Web service. The QoS-aware broker may implement various capabilities depending on each organisation's requirements. In any Web service provisioning infrastructure there are other issues that need to be considered as Dobson (2006) suggested. The first issue is QoS measurement; this is because the advertised QoS information of a Web service is not always trustworthy, therefore the QoS information of any advertised Web service needs to be measured before the service consumer binds to that service. The second issue is to provide means of enable the service consumer to negotiate for some QoS guarantees with a service provider and sign a Service Level Agreement (SLA) about the negotiated QoS guarantees. Finally, we need to consider QoS monitoring and detection of SLA violations. The QoS monitor detects the real behavior of the service. Then a monitored profile is constructed to be compared to the agreements included in the SLA. From the literature that we have reviewed above we discovered that all the brokers reviewed enable QoS-aware Web service selection, but there is no broker that combines all these capabilities (QoS negotiation in the context of SLA, QoS measurement and QoS monitoring) together. In our proposed broker for GUISET infrastructure we added QoS negotiation, SLA, QoS measurement and QoS monitoring capabilities.

3.3. QoS negotiation and SLA techniques

In a QoS-aware Web service provisioning infrastructure a broker is suggested by many researches to be a mediator between the Web service provider and service consumer. A brokerissupposed to select Web service on behalf of service consumers, by considering both the QoS requirements andthe functionality of the Web services (what the Web service does) specified by the service consumer. The literature suggests that the QoS-aware broker should have QoS negotiation as one of its functionalities. In a QoS-aware Web Service provisioning, it is important to enable service consumers to negotiate with service providers for some QoS constraints when a need to do so arises. Therefore, it is important that in our proposed QoS-aware Broker for GUISET infrastructure, we alsoinclude QoS negotiation.

The negotiation mechanism is the software component that accepts requirements and offers from parties as input, negotiates with its peer and delivers either deal or a conflict. QoS negotiation is when two or more parties try to reach an agreement on the QoS that they are willing to deliver to one another. QoS negotiation must be executed dynamically without the intervention of humans.Involved parties dynamically communicate their QoS requirements and their QoS provisions. A system can adapt easily in any changing conditions, if QoS negotiation is performed without any human intervention.It also allows the system to operate with different QoS levels, depending on the resources and the QoS available (Jarikoistinen and Aparna Seetheramal, 1998). Jarikoistinen and Aparna Seetheramal (1998) suggested that QoS negotiations must be performed dynamically as the system executes and it must involve Web services or Agents, not humans. During the QoS negotiations process, the negotiating entities can either agree or disagree on a deal. When a deal is agreed between the negotiating entities, the QoS constraints which were negotiated must be delivered in the context of the negotiated deal. Some deals are event limited and some are time limited. The agreed deal is said to be executed when the communication between the parties which were negotiating is

performed using the deal. Every executed deal ends and the deal is terminated once it ends. When the deal has been terminated at the end of execution, each involved party must check if the deal was executed successfully or not. A deal is only successful if both the parties which were involved in a deal honoured what was negotiated.

The "buy-and-offer"relationship that exists between the service consumer and provider is usuallymanaged by Service Level Agreement (SLA). SLA is a contract obligation between the service provider and consumer. The SLA specifies the functionality of aWeb service and the QoS constraints (such as response time, reliability, security, performance. Benyun *et al*(2009). Therefore, at the end of QoS negotiation, when both parties have agreed on the QoS parameters which were negotiated an SLA is created.

The service consumers and providers may have different goals, intentions andrequirements on the QoS; a Web service based negotiation mechanism is used in this case to manage the conflicts between service consumers and providers. Marco and Barbara (2006), stated that a complete QoS negotiation mechanism must define a negotiation protocol, negotiation objectives and a decision model. A negotiation protocol as the rules that have to be followed for offers posting, the negotiation objectives are a set of attributes on which it is possible to negotiate and the decision model is the rule used by the negotiating parties to decide to reject or accept an offer. Negotiation approaches can be classified as either many-to-one negotiations or one-to-one negotiations. A one-to-one negotiation approach is when there is exactly one party negotiating with exactly one party. While a many-to-one is when there is one party negotiating with many parties.

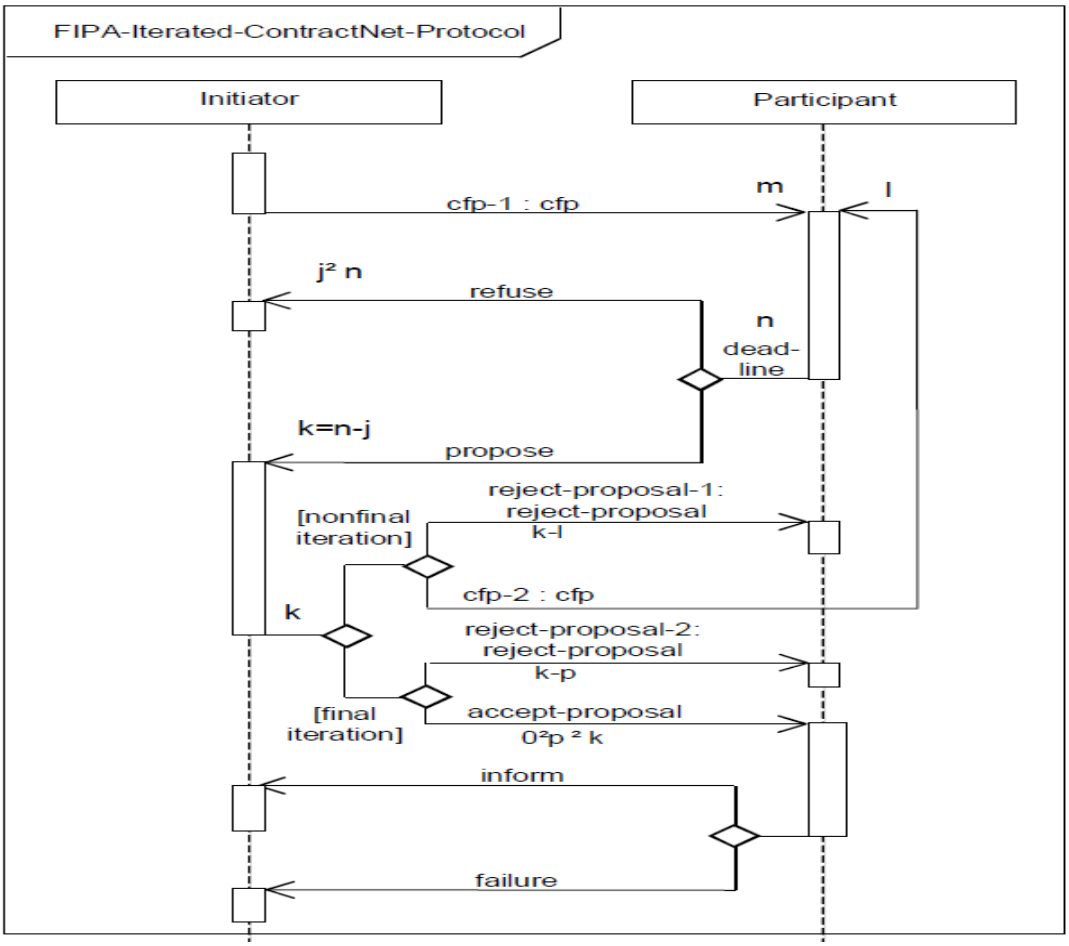

Figure 3.5: FIPA Iterated Contract Net Interaction Protocol (FIPA, 2002).

3.3.1. Many- to-one Negotiation Technique

A many-to-one negotiation approach is where many parties negotiate with just one party. Thus is the standard setting of auctions which have been popular on the *Internet* for some time now Alessio *et al*.,(2000). In this approach a request is issued first by the service requester. This request contains the functionality of the Web service that the service requester needs and also the QoS requirements. The different service providers can make offers, stating the QoS guarantees that they can possibly offer given the state of their resources capabilities on that time. The service requester can select one Web service that meets his QoS requirements. If none of the offers does meet the service requester's QoS requirements, the service requester can modify its QoS requirements and start the negotiation again.

The Foundation for Intelligent Physical Agents (FIPA) is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA have produced several standards specifications for heterogeneous and interacting agents, and for agent based systems. One of the standards produced by FIPA is the Interacted Contract Net Interaction Protocol (FIPA, 2002). The FIPA Interacted Contract Net Interaction Protocol for negotiation employs the many-to-one negotiation approach as shown in Figure 3.5.

Yan *et al*., (2007) stated that this FIPA Interacted Contract Net Interaction Protocol is the mostly widely used protocol for negotiation. Yan *et al*., (2007) also stated that the FIPA Interacted Contract Net Interaction Protocol is designed for one-to-many negotiation process, whereby one party negotiates with many parties, which are providing a similar service. In this negotiation approach, one party calls for proposals for some QoS constraints that are needed, then many parties send their proposals to that requesting party. Then the requesting party can choose one party that meets the requesting party's QoS requirements. If none of the responded parties does meet the requesting party's QoS requirements, then the requesting party would modify his QoS requirements and send another call for proposals. For our negotiation approach in GUISET's QoS-aware broker we will not adopted the many-to-one approach, because in our GUISET's QoS-aware brokera many-to-one negotiation approach scenario is not envisioned.

3.3.2. The One- to- One Negotiation Technique

An automated negotiation technique is said to be a one-to-one negotiation techniques if there is exactly one party negotiating with exactly one party. In most classical QoS negotiation processes, a one-to-one negotiation techniquerequest is issued first by the service requester. This request contains the functionality of the Web service that the service requester needs and also the QoS requirements. Then a service provider can make an offer of QoS guarantees that it can possible offer at that time. The service requester can accept the offer if it meets its QoS requirements, or reject the offer it does not its QoS requirements. This approach is used in the current SOA environment, where Web services are not published together with their QoS guarantees. This would mean that has a problem of selecting one Web service if there are many Web services which provide asimilar functionality.

Another form of one-to-one QoS negotiationis where the QoS information of a Web service is known, since it has been published in the service registry. In this approach, the service consumer is given an option to formulate his QoS requirements by combining different QoS classes from different available classes of QoS guarantees of the service provider. On the other hand, the provider could accept the new service consumer expectations or reject them for its reasons. This approach is widely used in QoS-aware Web service selection environments. And it fits perfectly to our GUISET QoS-aware Web service selection infrastructure, since in this infrastructure we propose that service providers should provide QoS information of every Web service published. And service consumers should specify their QoS requirements when sending a Web services request. The QoS requirements of the service consumers are used to select one Web service that meets the service consumer's QoS requirements. And service consumers should be able to negotiate for some QoS guarantees with the service providers.

As the literature suggests that a negotiation mechanism must have (1) the negotiation protocol, (2) a decision model, (3) QoS parameters that are being negotiated (4) Service Level Agreement (SLA) should be created at the end of the negotiations. The SLA serves as a contract agreement between the service consumer and the service provider.

A lot of research efforts have gone in to improving the modern approach of one-to-one negotiationsSerhani *et al*., (2005). Serhani *et al*., (2005) defined a protocol for negotiation which is similar to the modern one-to-one negotiation approach. In Serhani *et al*., (2005) the service consumer considers the QoS information published, then if there is no QoS guarantees that meet his requirement, then the service consumer negotiates for QoS requirements. Serhani *et al*., (2005) suggested that after every successful QoS negotiation the parties must sign an SLA. But in their negotiation approach, they did not discuss the decision model used by participating parties. Badidi*et al*., (2006) also followed the second approach (modern QoS negotiation approach). In Badidi*et al*., (2006) proposed a component called QoS negotiator which is responsible for conducting the QoS negotiations between the service consumer and service provider. The service consumer is starts the negotiating process by notifying the QoS negotiator about the QoS information it requires. Then the QoS negotiator component selects one service provider that meets the QoS requirements of the service consumer. Then, the QoS negotiator approaches the selected service provider to determine if the selected service provider can deliver the required QoS information or not. If the selected service provider confirms that it would be able to deliver the QoS requirements requested by the service consumer, the QoS negotiator component gives the service consumer all the necessary information about that particular service provider, to be used in the binding process. But if the service provider does not confirm that it would be able to deliver that requested QoS, and then the QoS negotiator can ask the service consumer to lower its QoS

requirements or wait until the conditions of the selected service provider become available to satisfy the service consumer's QoS requirement.

We acknowledge the inputof Badidi*et al*., (2006), because they considered SLA in their negotiation approach, although the approach talks about approaching the selected service provider to determine whether it would really deliver the agreed QoS. Badidi *et al*., (2006) didnot mention that how this process is being conducted. And they did not discuss the decision model used by participating parties.

However, there is a one-to-onenegotiation approach that is not fully employing the modern QoS negotiation approach discussed above. In this the QoS description is also known, since every service provider publishes its Web service together with its QoS information. Hongan *et al*., (2003) proposed a QoS Web service System Architecture that has a QoS negotiation manager component which employs the modern approach. In Hongan *et al*., (2003) the QoS description is known. The QoS negotiation manager selects all the Web services which offer similar functionality having the QoS description that is similar to the ones specified by the service consumer. The negotiator component then negotiates with the QoS server to make sure that the guaranteed quality of services can be provided to the service consumer. This process is conducted to all the selected Web service providers until one service provider confirms that the QoS guarantees would be provided to the service consumer. The negotiation approach presented in Hongan *et al*., (2003) is not really a negotiation approach, but rather a confirmation of QoS constraints, although the authors claim that it is a negotiation approach. However, this approach can waste a lot of timewhen it is confirming each QoS advertised by the service providers. Therefore, in our QoS negotiation approach, we will not need this kind of negotiation.

3.3.3. Summary QoS negotiation and SLA techniques

In the context of our work, during the web service discovery, the QoS descriptions are known. They are published in advance by the Web service provider. Then a Web service consumer has as a first alternative to choose from what have been described, the QoS combination that meets his requirements and then binds to that Web service provider who is provides the exact QoS guarantees he or she has requested. The second option is that if the Web service consumer does not find a Web service that has the QoS guarantees that matches the consumer's QoS requirement therefore there is a need for a QoS negotiation mechanism that would enable the Web service consumer to formulate QoS expectations and send them to a Web service provider. The provider can accept the new QoS expectations or reject them at will. From the foregoing discussion, it is apparent that this approach is a one-to-one negotiation approach which is discussed in Serhani *et al* (2005) Badidi *et al*., (2006) and Hongan *et al*., (2003).We will adoptthe one-on-one negotiation approach in our work, to enable QoS negotiation between the provider and the consumer, we need to develop a QoS oriented negotiation algorithm that defines the main steps of bi-directional negotiation between the client and the provider, and the decision model used by both parties to agree or reject a proposal. On the basis of the decision model the negotiating parties would then be forced to consider the SLA after a successful QoS negotiation.

3.4. QoS measurement

In a QoS-aware Web service provisioning environment like in our GUISET infrastructure, service consumers specify their QoS requirements when requesting a particular Web service. Then a QoS-aware broker is used to select a Web service on behalf of the service consumer.The broker uses the required QoS properties and the functionality of the Web service specified by the service consumer. Jan and Hasso (2007), stated that in an operational

ecosystem, services which are offering only weak QoS properties would most probably be traded at a lower price, allowing for small competitors to enter the market and to introduce strong QoS guarantees at a later point in time, when statistics prove stability for a certain QoS level. Therefore, many service consumers will select those Web services with strong QoS guarantees. This act might prompt some of those service providers which are offering weak QoS properties to advertise false QoS guarantees, to attract service consumers to select their Web service. And then at the end of the Web service provisioning process you find out that they did not fulfil the QoS properties they guaranteed. In the GUISET infrastructure, to avoid service consumers from choosing a service provider which has advertised false QoS guarantees, we need to measure those QoS property values at the service provider's side before the service consumer binds to that Web service. This act is called QoS measurement, in our proposed GUISET infrastructure we want to measure every QoS property value after the successful QoS negotiation mechanism to avoid service provider from misleading service consumers. This QoS measurement mechanism is one capability which our QoS-aware broker for our GUISET infrastructure must perform during the QoS-aware Web service provisioning state. There are techniques which are used for QoS measurement, namely QoS verification and certification and QoS measurement at run time. The next Sections present two techniques together with their pros and cons.

3.4.1. QoS verification and certification technique

The QoS verification and certification techniques force service providers to go through the verification of their QoS claims. Then issue a certificate is issued if the QoS guarantees are true, before allowing the service provider to publish a Web service together with its QoS guarantees to the registry. In this technique the service provider must find the QoS certifier componentfirst, and then the service provider submits the QoS claim to the QoS certifier. The

QoS certifier component can now verify the claims. During the verification process, the QoS certifier component checks the correctness of the QoS claims. If the service provider's QoS claims prove to be correct then a certificate is issued. But if the service provider's QoS claims are not correct then the QoS certifier will not issue a certificate. After the verification process, the QoS certifier component informs the service provider about the outcome of the verification process. The service providers with the correct QoS claims are allowed to register their Web service to the service registry, while those with incorrect QoS claims are not allowed to register their Web service.

But if the QoS claims are not proved to be what the service provider claimed then the certificate will not be given to that service provider.

Other research works e.g. Serhani *et al*.,(2005) use the Broker to keep the verified certificates of each Web service. The broker communicates with the certifier component to check the existence of the certification. After successful checking, the registry then registers the service in its repository. So when a service consumer sends a Web service request which has QoS requirements, the QoS-aware broker would then select a Web service that meets the service consumer's QoS requirements then validates the certification of those QoS claims. This technique has been adopted by a number of research works in this field of QoS-aware Web service selection for QoS measurement e.g. Badidi*et al*.,(2006), Serhani *et al*.,(2005). In Badidi*et al*.,(2006) the process of howis the verification conducted is not specified. But Serhani *et al*., (2005) discussed how the verification process is conducted;and provided the formulas that are used for measuring the QoS property value during the verification process.

However, this QoS verification and certification has been widely criticised because the verification of QoS properties at the time of registration donot provide any guarantees of having up-to-date QoS information (i.e. in cases of Web service updates or changes), since *Internet* applications and traffic are dynamic and impermissible in nature (Liu *et al*.,2004; Eyhab and Qusay 2007 and Core,2009). This means for example, if the QoS information of a Web service was verified maybe 5 minutes ago it is possible that after that 5 minutes the QoS parameter value might have changed due to the dynamic nature of the *Internet* applications and traffic. So using the verification and certification technique can still lead to the dissatisfaction of guaranteed QoS. Another disadvantage of the QoS verification and certification approach is that it requires service providers to go through additional steps during the publishing process which forces them to think of ways to measure the QoS of their Web services. Owing to these disadvantages of QoS verification and certification technique, we will not appropriate adopt this technique in our work. In our approach we need a technique that would measure the QoS parameter values at run time before a service consumer binds to the Web service.

3.4.2. QoS measurement at runtime

In Section 3.4.1 we mentioned that the QoS verification and certification approach does not provide up-to-date QoS information due to the ever changing nature of the *Internet*. Other research works consider measuring the QoS information of the selected service provider at run time. In QoS measurement at runtime, a ping or probe program measures the QoS properties and their respective values. A *Uniform Resource Locator* (url) of the selected service provider used in the ping program. With this ping program one can be able to measure a number of QoS properties like response time, availability, throughput, reliability,

etc. Heiko *et al*., (2003) used the ping concept to measure availability and response time of a particular service provider; they also defined formulas which they use during the ping, since a ping program does not have formulas for calculating these QoS properties. Thomas *et al*., (2009) used the ping program to the measure the response time of a particular Web service.

In our own work, we will needa technique that willperformQoS measurement at runtime. The QoS measurement at runtime gives real-time QoS property values compared tothe QoS verification and certification technique. And QoS measurement at runtime does not require service providers to go through additional steps during the publishing process, which enforces them to think of ways on how to measure the QoS of their Web services. Service providers can just publish their Web services together with QoS information. During the Web service selection process before a service consumer binds to that Web service then the QoS measurement is performed. The QoS measurement process requires formulas for calculating QoS properties (response time, throughput, and availability) defined by Eunju and Chair*et al*., (2005).In their OASIS Web Services Quality Model TC document. OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society(Eunju and Chairet, 2005). After the QoS measurement has been performed and the QoS guarantees have proved to be true, then the service consumer is allowed to bind to that Web service. In our proposed QoS-aware broker we will also requirea QoS negotiation mechanism, QoS monitoringtechnique and SLA. The QoS monitoring capability is initiated after the Web service provisioning has started, to monitor any violation of the QoS guarantees.

3.5. QoS monitoring

The QoS monitoring is an essential foundation for SLA, where SLA defines the expected QoS between service providers and consumers. QoS monitoring is used to check for any violation of the agreed QoS guarantees. If two or more organisations are dependent on each other on the type of Web services which they use in their individual business purposes, they would need a contractually written and signed guarantee of the quality of services of the Web services. The service providers would need a guarantee that the service consumers would not misuse their Web services. On the other hand, service consumers would also need a guarantee that the service providers would be able to deliver the QoS guarantees which were agreed upon. These guarantees must be defined in a SLA, which serve as a contract agreement between the service provider and service consumerFranco *et al*.,(2008). Monitoring plays an important role in determining whether a SLA has been violated. SLA violation monitoring begins once an SLA has been defined. The SLA is defined during the QoS negotiation and SLA stage, where Web service consumer and service provider agree on some QoS constraints. The SLA is a signed agreement is between these two parties. SLA can contain the following information:

- Web Service functionality: Describe the functionality which the selected Web service performs.
- Validity period: Describes the amountof time in which the SLA would be able to cover (starting time and ending time).
- Parties: Contains the details of the parties which are involved in the SLA.

- Service level objectives: Defines the QoS parameters and their values (e.g. Response time, Availability throughput, etc.) which both the service consumer and service provider agreed upon.

- Penalties: Describe the type of action that would be taken if one party violates the QoS guarantees.

- Price: define the cost of the web service

There are two forms of monitoring namely: online monitoring and offline monitoring.

Online monitoring: Inthis form of monitoring, alerts about any form of violation are generated immediately since every data areanalysed while the Web service is being delivered.This involves periodically testing if the parties are keeping the agreement terms.. The monitoring interval can vary, depending on the agreement's SLOs, but in general it has to be quite small (of the order of seconds) (Dalia and Martijn, 2009). There are a number of research works that use the online monitoring technique e.g. Bahareh *et al*.,(2010) and Dalia and Julian(2010). The Windows Performance Counters (WPC) used for monitoring is. WPC is a windows component that presents an object framework for supporting application monitoring. This WPC is implementation dependent. The disadvantages of any online monitoring technique are that online monitoring is not easy to implement in aproficient manner and it causes a very high performance overhead. Another disadvantage of an online monitoring system is that it can form a bottleneck (Dalia and Julian, 2010). Therefore, this technique cannot be adopted for our own QoS monitoring because of the above disadvantages.

Offline monitoring: In this form of monitoring, all the interactions during the Web service provisioning are recorded and stored in a securely place. This functionality is performed by a third party and if any party that was involved in the creation the SLA believes that there isa

violation of the agreement, the log file is retrieved and examined to find out that whether there is a violation or not (Dalia and Martijn, 2009). Clark *et al*., (2010) adopted the offline monitoring technique where the Trusted Third Party (TTP), an independent module that can monitor (and log) all communication between consumers and service providers, is used. The main disadvantage of this form of monitoring is the need for storing the log files.

Literature(Bahareh *et al*., 2010; Dalian and Julian, 2010 and Clark *et al*., 2010) shows a total distinction between the online monitoring and offline monitoring by specifying the advantages and disadvantages of each. By weighing the advantages and disadvantages of each approach, we decided to use the offline monitoring because its only dominating disadvantage is that of requiring storage of the agreed SLAs during the Web service provisioning ,and the disadvantages of online monitoring are many. The first one goes with our experience and the fact that the literature started that it is difficult to implement Dalian and Julian (2010).Secondly, it causes overheads. Our proposed QoS-aware broker would be the Trusted Third Party (TTP) that Clark *et al*., (2010) used to perform QoS monitoring. The QoS-aware broker stores the agreed SLA and then logs the communication between consumers and service providers during the Web service provisioning. At the end of the delivery of the Web service, the broker compares the QoS parameter values which were agreed on the SLA with the QoS parameters values which the broker logged during the Web service provisioning.

3.6. Summary

In a QoS-aware Web service provisioning infrastructure like our GUISET infrastructurediscussed in Adigun *et al*., (2006), a middleware/broker component is

introduced to enable a QoS-aware Web service selection, (meaning that broker must select Web service for service consumers not only based on the functionality of the Web service but also considering the QoS requirements specified by the service consumer during the Web service request). Apart from the fact the broker must enable QoS-aware Web service selection. There are other capabilities that need to be considered in a QoS-aware Web service provisioning infrastructure (Dobson, 2006). These capabilities are incorporated in the QoS-aware broker, the capabilities includes QoS negotiation (Yan *et al*.,2007; Serhani *et al*., 2005; Badidi*et al*., 2006), QoS measurement (Heike *et al*., 2003), QoS monitoring and Service Level Agreement Franco *et al*., (2008). From the literature there is none of the existing QoS-aware brokers that perform all these capabilities. Therefore, in our own proposed QoS-aware broker we want to combine all of these capabilities, meaning that our QoS-aware broker would support QoS negotiation and SLA, QoS measurement and QoS monitoring.

CHAPTER FOUR

GUISET-INSPIRED QOS-AWARE WEB SERVICE SELECTION MODEL

4.1. Introduction

Web Services are becoming popular these days and businesses are planning to build their future solutions on the Web service technology. A GUISET environment will typically be a coalition of geographically dispersed mobile individuals, groups, organizational units or organizations that pool resources, capabilities and information to achieve common objectives (Adigun al at., 2006). This would result in having a large number of Web services available in our GUISET infrastructure and there would be a large number of Web services which provide similar functionality and which are probably hosted by different individuals, groups or organization. As Yang *et al* (2008) stated many grid services with same functionalities and different quality of service (QoS) are presented in the grid. As a result service consumers are facing challenges in selecting the oneneeded Web service among a set of Web services offering similar functionality. Gwyduk (2006) suggested that with different non-functional properties, QoS parameters serve as metrics to differentiate the services and their providers. But the current Service Oriented Architecture (SOA) environment called 'register—find—bind' model is neither accountable nor responsible for the QoS descriptions. This is because its discovery mechanism is only based on what the Web service does without considering Quality of service (QoS) guarantee. In chapter one, we stated that Dobson (2006) suggests that during QoS-aware Web service provisioning, there is a need for the following capabilities:

- QoS enabled Service registry,
- QoS measurement scheme,
- QoS negotiation scheme,
- Service Level Agreement (SLA), and
- QoS monitoring during Web service provisioning;

We have observed from the literature that a number of researchers(Rajendran *et al*., 2010; Gang *et al*., 2009;Serhani *et al*.,2005) have already proposed a QoS-aware web service selection Brokers and some of the proposed QoS-ware Brokers have already addressed QoS measurement(D'Mello, *et al*.,2008;Rajendran *et al*.,2010;Refael and Carlos, 2006), QoS monitoring (Gang *et al*., 2009;Badidi*et al*.,2006;Daneil and Vinod, 2007), QoS negotiation (Serhani *et al*.,2005;Refael and Carlos, 2008;Hongan *et al*., 2003) and SLA (Gang *et al*., 2009, Tao and Kwei2004;Badidi*et al*., 2006). But we also discovered from the literature that none of the already proposed QoS-aware Broker combines these capabilities. Therefore in this research, we propose a client framework that demonstrates the feasibility of combining all the four capabilities in a single broker.

Section 4.2 discusses the design criteria for the GUISET QoS-aware Web service Selection Framework. Section 4.3 gives the proposed framework based on the design criteria. A description of the framework's components and how the components interact is presented in Section 4.4. Section 4.5 gives the concluding remark.

4.2. Design Criteria for a QoS-aware Web service Provisioning Framework

The GUISET environment is envisioned to be community of a large number of heterogeneous services deployed in different administrative domains. The service providers and consumers may join and leave GUISET dynamically. In GUISET, service providers would be allowed to advertise their Web services. As the number of service providers advertising their Web services increases, the number of Web services that offers similar functionality would also increase. As mentioned before that the literature suggests that quality of service can be used to differentiate Web services even though their functionality is similar. Adigun at al., (2006) proposed the GUISET architecture. The middleware layer of the GUISET architecture proposes a utility broker, which deals with managing the sharing of resources and service selection. The utility broker as envisioned, accepts Web service request from the service consumer, and then selects a Web service that meets the service consumer's requirements (the functionality of the Web service and the guaranteed QoS by the service provider). The mechanism of using utility broker in QoS-ware Web service provisioning is being rapidly adopted in the literature (Menasce*et al*.,(2007), Damian *et al*.,(2008),Badidi*et al*.,(2006)). Our research is aimed at modeling the utility broker for GUISET proposed in (Adigun, *et al*., 2006). Based on the research questions outlined in chapter one, Section 3, there is a need to create a consumer-centric framework that selects services based on the interest of the consumer. From our review of literature, we have identified the design criteria to consider when designing a QoS-ware Web service provisioning environment. The design criteria for crafting the QoS-aware Web service selection include: QoS enabled Service registry QoS measurement scheme, QoS negotiation scheme, Service Level Agreement (SLA), and QoS monitoring during Web service provisioning.

1. **QoS enabled Service registry:** A service registry in SOA defines a set of data structures to describe Web services for the purpose of advertisement and discovery.

Service providers can publish the information of businesses, services and protocols in the service registry, and service requesters can look into the services and discover the required services. One shortcoming of the current SOA's service registry is that it was not built to store QoS information of a Web Service. Quality of Service becomes a key to differentiate those Web services with similar functionality. Therefore, there is a need for a service registry that will enable Web service providers to publish Web services together with their QoS information. So far there is no standard method for publishing Web services with their QoS information (Shuping, 2003).

2. **QoS negotiation:** During QoS-aware Web service provisioning, it might be possible that a service consumer is not getting the exact QoS guarantees that matches the consumer's request. If that is the case, there must be a mechanism that will enable the service consumer to negotiate for some QoS guarantees with a service provider. The negotiation phase is a process conducted at the end to reach an agreement between concerned parties (Web service consumer and service provider) for a required level of QoS. The QoS negotiation mechanism is executed dynamically without human's intervention. During the QoS negotiation the negotiating parties dynamically communicate their QoS requirements and their QoS provisions. During the negotiation process the participating parties can agree or disagree on a deal(Jari and Aperna, 1998). If a deal has been established, it is desirable that both parties sign the Service Level Agreement (SLA). The SLA contains the details of the agreed deal. These details includes the names of the parties which were involved in the negotiation, details of the payment methods to be used after the execution of the deal the functionality that the requested Web services performs, agreed QoS parameters and their values, and a penalty to be applied if one party violates the agreed QoS guarantees (Dobson, 2004).

3. **QoS measurement:** In a QoS-aware Web services selection environment, some service providers may publish false QoS guarantees just to attract more service consumers. Therefore,it is not advisable to simply consider the QoS guarantees that are advertised by the service providers (Janm, 2007). There is a need for a mechanism to measure QoS guarantees before allowing the service consumer to bind to that Web service. The QoS measurement mechanism can provide better QoS-aware Web service provisioning by providing up to date quality of service information. And preventing service consumers from bind to services with false QoS guarantees(Dobson, 2004). The Measurement Service maintains information on the current system configuration, as well as run-time information on the metrics that are part of the Service Level Agreement (SLA) which is a signed agreement between the service consumer and service provider.

4. **QoS monitoring:** Once an agreement between the service provider and consumer has been established and the Service Level Agreement has been created, the service consumer binds to that particular service provider. During this stage it is desirable to have a monitoring mechanism which plays a critical role in determining whether SLAs are achieved or violated at the end of the Web service provisioning. SLA monitoring may require the involvement of third parties: They come into play when either a function needs to be carried out that neither service provider nor customer wants to do, or if one signatory party does not trust the other to perform a function correctly

It is believed, in our work, that considering a QoS enabled registry, a QoS measurement technique, QoS negotiation, SLA creation and QoS monitoring during drafting Web service

provisioning will ensure an appropriate QoS-aware Web service provisioning environment in our GUISET infrastructure.

4.3. QoS-aware Web service Selection Model

As more and more Web services are created by many different service providers, there is often a case where a number of Web services can have similar functionality. In this case the service consumers are always presented with a group of Web services offering the similar functionality. How to discover and select the appropriate Web service among the large number of Web services is the problem in hand. A discovery mechanism in view of functional and QoS properties is essential for Web service consumers. Based on the design criteria outlined in Section 4.2, and the analysis of existing models, a QoS-aware Web service selection model for our GUISET research focus is proposed. The model proposed in this work extends the standard Service Oriented Architecture (SOA) architecture, which is composed of service consumer, service provider and Service registry by adding a broker. GUISET now has a service selection model defined for all clients. The model derives from the idea that SOA architecture has been successfully extended in the past using a broker. Therefore, we think our GUISET utility broker should be of a QoS-aware framework consisting of:

- QoS Negotiation Protocol
- Decision Model
- QoS Measurement Mechanism
- Service Level Agreement
- QoS Monitor
- A modified t-Model for GUISET service repository.

There are many existing architectures like this. The broker acts as a mediator between the service consumer, service provider and the Service registry. This QoS-aware broker is responsible for accessing the Service registry for the discovery of QoS-aware Web services. Thus, the service consumer does not need to make the Web service discovery in the Service registry. The service consumer application submits a Web service request to the broker. The service consumer's request contains the functionality of the Web service (that is what the Web Service does) and the QoS requirement. The broker receives the request. After receiving the service consumer's request, the broker contacts the Service registry. The broker discovers from the Service registry all the Web services which offer similar functionality that the service consumer specified on the request, as shown in Figure 2.5. Among these discovered Web services which offers similar functionality but different with the QoS guarantees which they provide, the broker selects one Web service that meets the service consumer's QoS requirements using the QoS information specified on the request. If there is no Web service that meets the exact QoS requirements, the broker can select one Web service provider to negotiate with for some QoS commitments. Some service providers may publish inaccurate QoS information to attract more service consumers (Dobson,2006). To avoid that situation in our GUISET QoS-aware Web service selection infrastructure we designed a QoS measurement mechanism that measures the compliance of the advertised QoS before allowing the service consumer to bind to the Web service. A QoS negotiation mechanism for allowing the service consumer and service provider to negotiate for some QoS constraint is also designed. After every successful negotiation process the broker creates the SLA. The SLA is a contractual obligation between the service provider and service consumer which specifies mutually agreed understanding and expectation of the provision of a service. The SLA describes a set of QoS constraints and their values which both parties agreed upon. During the Web service provisioning process a QoS monitoring technique is used to check

for any violation of the agreed QoS guarantees. Sections 4.3.2, 4.3.3, 4.3.4 and 4.3.5 respectively give a detailed discussion on these mechanisms outlined above, namely: QoS measurement, QoS negotiation, SLA and QoS monitoring.

4.3.1. Web Service registry

A service registry is an important part of any service-oriented architecture. A service registry is designed to allow service provider to register their Web services and for service consumers to locate useful Web services. The basic functionality of a Service registry is to store information about Web services in an SOA. Several other types of service registry exist, such as the JISC's IESR (Information Environment Service Registry) Pete(2004), electronic business XML (ebXML) OASIS(2003) and the UDDI (Universal Description, Discovery and Integration) OASIS (2003). Luc (2005) stated that the UDDI registry offers the greatest flexibility in implementing SOA application. David (2005) stated that the UDDI registry is the widely accepted Service registry and is currently being voted by the member of OASIS as a solid and matured Service registry as compared to the ebXML. The IESR Service registry is only being used by the United Kingdom Higher and Further Education and it focuses on improving resource discovery mechanisms for electronic resources. As David (2005) stated, that the UDDI registry is the widely used and accepted services registry, a number of research works adopted the UDDI registry (Rafeal and Carlos, 2006;Tao and Kwei 2004; Gang *et al*.,2009). We have also chosen the UDDI registry as our service registry in this work.

To have a QoS-aware Web service provisioning infrastructure, firstly you require a platform that will enable service providers to publish Web services together with their QoS information. UDDI (Universal Description, Discovery and Integration) defines a set of data structures to describe Web services for the purpose of advertisement and discovery. Service

providers can publish the information of businesses, services and protocols in the UDDI and service requesters can look into the services and select the required services to compose desired services Clement *et al*.,(2004). One of the important criteria for Web service selection is QoS. But the current UDDI registry was not built to store QoS information of a Web service. A tModel in UDDI is used to store some technical information of a Web service and it can also register categorization. The categorization provides mechanism for putting more property information to a UDDI service registry (Ziqiang *et al*.,2007). Blum (2004) proposed that the QoS information of a Web service can be stored in the UDDI service registry using the tModel categorisation. In his approach each *bindingTemplates* have a tModel reference in its *tModelInstanceDetails* collection. The *tModel* have multiple categories on it, each reflecting a different QoS attributes with particular values or ranges of values. This approach is more efficient Chi-Chun *et al*.,(2008) and has been adopted by many researchers (like in Ziqiang*et al*., (2007); Rajendran and Balasubramanie (2010); Diego *et al*., (2006)).

For enabling service providers to publish Web services together with their QoS information we apply the same technique proposed by (Blum and Fred, 2004). In this approach, whenever a service provider publishes a Web service, a *tModel* within a UDDI service registry is created and be registered. The created and registered *tModel* of the Web service published contains the QoS information of that Web service. The *bindingTemplate* inside the UDDI structure contains a unique *tModelKey* that points to the location of the *tModel*. The *tModel* has a *catagoryBag* inside it.

```
<tModel
tModelKey="uuid:268A3C80-BCF4-11DF-B0B4-A7D1A7835784">
<name xml:lang="en-ZA">EmailValidation</name>
<categoryBag>
<keyedReference
keyName="EmailValidation"
keyValue="EmailValidation"
tModelKey="uuid:2EC65201-9109-3919-9BEC-C9DBEFCACCF6" />
<keyedReference
keyName="uz-org:ubr_browser:qos:availability"
keyValue="(50.0,60.0)"
tModelKey="uuid:A035A07C-F362-44DD-8F95-E2B134BF43B4" />
<keyedReference
keyName="uz-org:ubr_browser:qos:responsetime"
keyValue="(1439,2658)"
tModelKey="uuid:A035A07C-F362-44DD-8F95-E2B134BF43B4" />
<keyedReference
keyName="uz-org:ubr_browser:qos:negotiable"
keyValue="true"
tModelKey="uuid:A035A07C-F362-44DD-8F95-E2B134BF43B4" />
</categoryBag
</tModelInstanceDetails>
```

Listing 4.1. The *tModel* with the QoS Information

Inside the *catagoryBag* there is *keyedReference* that contains the *keyName*, *KeyValue* and *tModelKey* of the QoS information of the Web service. The *keyName*presents the name of the QoS attribute. The *keyValue* presents the value that is assigned to the QoS attribute. The *tModelKey* inside each *keyedReference* represents a unique namespace.

Listing 4.1 illustrates a *tModel* with the QoS information of an *EmailValidation* Web service. The QoS attributes that are presented in *tModel* includes Response Time and Availability.

4.3.2. QoS measurement mechanism

The discovery and selection of a Web service which meets the service consumer's QoS requirement is conducted based on the advertised QoS information by the service providers. As we have mentioned before that, in this work we do not trust the QoS information

advertised by the service providers. Netra and Arpita (2010) stated that service providers have a vested interest in overstating their QoS capabilities or the published QoS information may be out of date. We do not let the service consumer bind to the selected service provider based on the advertised QoS information; rather we measure the QoS information of the selected service provider to verify the correctness of the QoS parameters' values. In doing so, we avoid situations where the service consumer binds to the service provider that would not satisfy the service consumer's QoS requirement at the end of the Web service provisioning process. A QoS measurement mechanism is designed; the main functionality of this mechanism is to measure each QoS parameter value that is advertised by the service provider. There are standard formulas that are used to measure the QoS parameter values; these standard formulas are defined by the Organization for the Advancement of Structured Information Standards*(*OASIS 2002). There are many QoS metrics and standard techniques for measuring them, but in our case for simplicity purposes, we consider *response time, availability, and throughput*. However, other QoS metrics could be easily added to our framework without changing the methodology. The standard formulas for measuring these QoS metrics are as follows:

- **Response time:** This is the time taken to send a request and to receive the response. The Response Time is measured at an actual Web service call and it can be calculated by applying the following formula.

Response time = Response CompletionTime – User Request Time

The *Response Completion Time* is the time that all the data for response arrives at a user, while the *User Request Time* is the time when the user sends a request.

- **Availability:** It is the probability that the service is accessible (available for use), or is the percentage of time that the service is operating.

  Availability = $\frac{number\ of\ successful\ invocations}{total\ invocations} \times 100\%$

  Number of successful invocations is the total number of Web service invocations which were successfully invoked. *Total invocations* is the total number of the Web service invocations, including successful and - failed invocations.

- **Throughput:** This is the maximum number of services that a platform providing the Web Service can process in a given period of time.

Throughput = number of completed request per period

4.3.3. QoS negotiation mechanism

During the QoS-aware Web service discovery and selection, a Web service that meets the exact service consumer's QoS requirement is selected. If none of the discovered QoS-aware Web services has the exact QoS requirement as specified by the service consumer', then one service provider that has QoS guarantees which are closer to what the service consumer requested, is then selected to enable negotiation with the service consumer for some QoS commitments. QoS measurement technique is applied after the discovery and selection process to get the true QoS parameters' values of the selected Web service. Rafeal and Carlos (2006) stated that the *Internet*'s application and traffic are known for being dynamic; therefore it is highly unlikely that the measured QoS values can be exact as the service consumer's QoS requirements. A QoS negotiation mechanism is applied to enable

negotiation between the service provider and service consumer about some QoS commitment. The QoS values measured during the QoS measurement process are used in this QoS negotiation mechanism. Marco and Barbara (2006) stated that a complete QoS negotiation mechanism must define a negotiation protocol, negotiation objectives and a decision model. Section 4.3.3.1 discusses the negotiation protocol that we used, which explains the flow of messages between the negotiating parties. The negotiation objectives are the set of QoS parameters specified by the service consumer. We also explain the decision model that service consumers and service providers use to agree or disagree on a deal during QoS negotiation process in Section 4.3.3.2.

4.3.3.1.Explanation of the Negotiation Protocol flow

A negotiation protocol depicts the flow of messages that occurs during the negotiation process. In this work we assume that every service provider can set his QoS parameters as non-negotiable or negotiable. And every negotiating party has a decision model that it uses to accept or reject a deal during the negotiation process. The decision model used by the negotiating parties is discussed below in Section 4.3.3.2. The GUISET QoS-aware Broker proposed in this work is capable of negotiating on behalf of the service consumer. The service consumer sends its decision model to the broker. The GUISET QoS-aware Broker sends a request for negotiating to the service provider. The request contains the QoS parameter being negotiated, which is the QoS value measured by the QoS measurement component. Since the GUISET QoS-aware Broker knows the decision model of the service consumer, it knows that the negotiated value fits well in the decision model of the service consumer. The service provider can choose to refuse to negotiate or can accept. If the service provider accepts to negotiate then the GUISET QoS-aware Broker sends a proposal. The service provider can reject or accept the proposed QoS constraints. If the service provider

accepts the proposed value then the Broker would inform the service consumer that an agreement has been reached. Both the service provider and service consumer they can enter the Service Level Agreement (SLA). If the service provider rejects the proposal, it sends the rejection message and its decision model to the GUISET QoS-aware Broker. The GUISET QoS-aware Broker then sends an improved proposal using the decision model of both service consumer and service provider. The GUISET QoS-aware Broker would be aware if the service provider would accept or reject the proposal since it knows the decision model for the service consumer and provider now. If it is aware that the service provider would not accept the service consumer's proposal it does not send the proposal. This process prevents infinite iteration. Figure 4.1 shows the negotiation protocol discussed here.

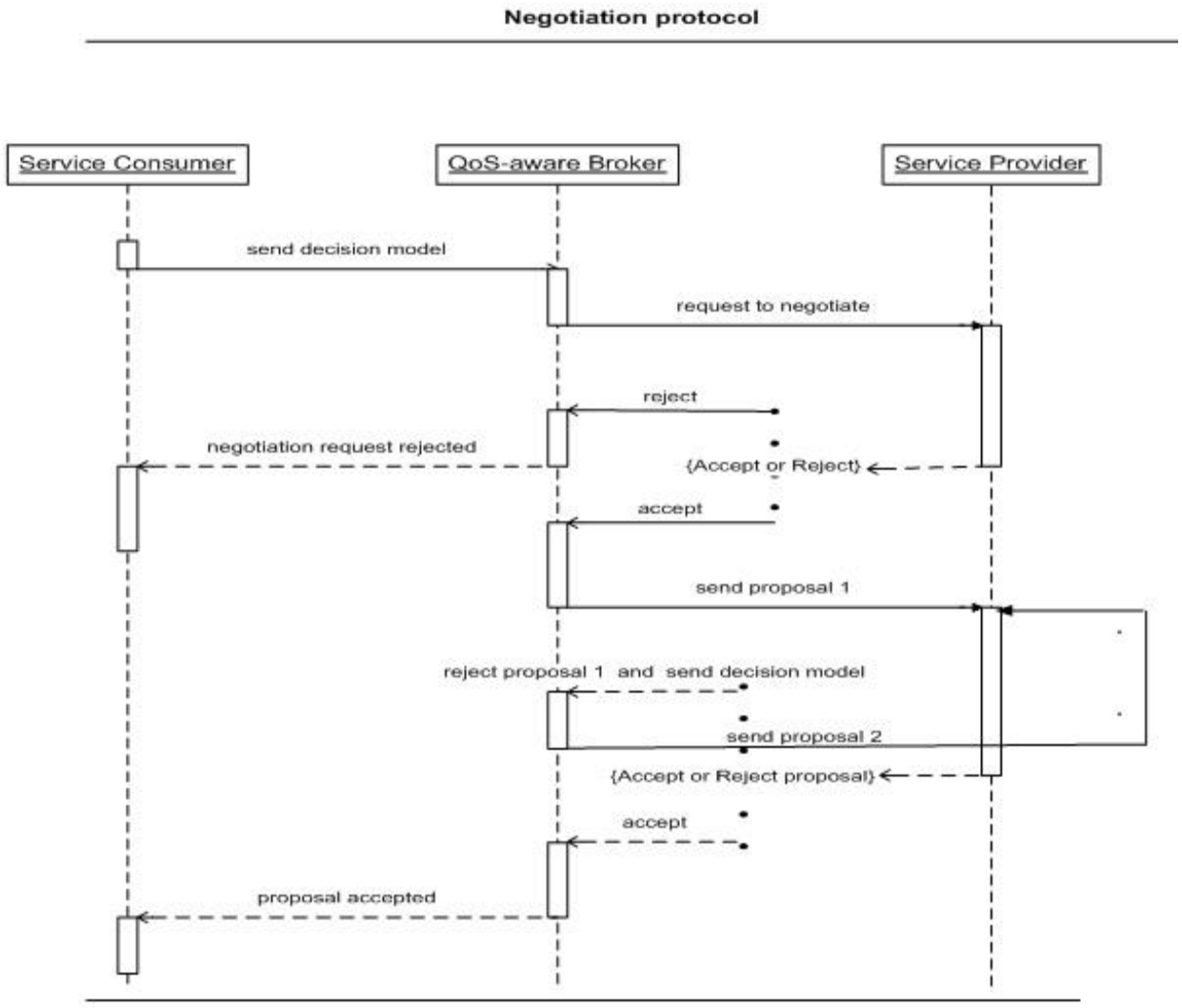

Figure 4.1: Negotiation Protocol

Our negotiation protocol is inspired by FIPA (2002). Some researchers e.g. Jun *et al*., (2007) suggested that although the FIPA Iterative Contract Net Interaction Protocol is a commonly used negotiation protocol for one-to-many negotiation approaches, the protocol can be modified to suit a one-to-one negotiation. Therefore, it is this reason that inspired the formulation of our own negotiation protocol that suits our negotiation approach.

4.3.3.2.Decision Model

Negotiation can be viewed as a bargaining process by which a joint decision is made by two parties. The parties first verbalize contradictory demands and then move towards agreement. Therefore, for them (the service consumer and a provider) to take a decision to agree or disagree on a deal each party must have a decision model. A Decision model is a private process at the service consumer or provider side. Each negotiation party uses its own negotiation strategies, in conjunction with an agreement template, to arrive at an offer or counter-offer in the negotiation process. It is employed by all parties in negotiation to act autonomously in accordance with the negotiation protocol. Our research adopts the decision model used in (Marco and Barbara, 2005 and Jari and Aparna, 1998). This decision model allows the service consumer and service provider to set up their maximum and minimum values for each QoS attribute. So during the negotiation process each party makes a decision by looking at its maximum and minimum value. Each party accepts an offer if negotiated QoS parameter value falls between the maximum and minimum value and the offer is not accepted if it falls outside of the maximum and minimum value.

4.3.4. Service Level Agreement (SLA)

Once the service consumer and provider reach an agreement on the negotiated QoS constraints an SLA needs to be created. The SLA defines mutual understandings and expectations between the service provider and consumer about the negotiated QoS constraints and their values. SLA contains the following:

- Web Service functionality: Describe the functionality which the selected Web service performs.

- Validity period: Describes the amountof time in which the SLA would be able to cover (starting time and ending time).

- Parties: Contains the details of the parties which are involved in the SLA.

- Service level objectives: Defines the QoS parameters and their values (e.g. Response time, Availability throughput, and etc.), which both the service consumer and service provider agreed upon.

- Penalties: Describe the type of action that will be taken if one of the parties violates the QoS guarantees.

- Price: define the cost of the web service.

4.3.5. QoS monitoring

After the service consumer and service provider have negotiated and signed the SLA then binding process starts. During this service provisioning process there is a need for monitoring any form violation of the QoS guarantees which were negotiated and agreed upon by both the service consumer and service provider. The service consumer wants the service provider to deliver the QoS guarantees which were negotiated while the service provider does not want the service consumer to misuse the Web service being delivered.Therefore, we have developed a QoS monitor component that is part of our GUISET QoS-aware Broker to monitor any violation of agreed QoS parameter. We chose to use Offline monitoring technique rather than the online. The decision to use Offline monitoring technique was based on the fact that it is easy to implement and it does not cause system overhead. And a decision to not use Online monitoring isbecause, Online monitoring is difficult to implement and it also causes system overhead Dalia and Julian(2010); Clark *et al*., (2010);Dalia and Martijn(2009).

4.4. The GUISET QoS-aware Web service Selection Model Architecture

As we have stated in our design criteria that in the GUISET infrastructure we need a platform that would allow service providers to specify QoS information of each Web service during the publishing stage. The QoS information of each Web service published to the Web service registry is essential in differentiating all the Web services that offers similar functionality and this make the Web service discovery and selection to be more consumer-centric. However, the discovery mechanism of the current SOA environment does not support QoS. To address this issue we extend the current SOA environment by introducing a mediator component, also known as a broker. This broker supports QoS-aware Web service selection. With this QoS-aware broker, service consumers specify their QoS requirements during the Web service request and the QoS-aware broker is able to select a Web service that meets the service consumer's QoS requirement. This proposed QoS-aware broker also performs QoS measurement, QoS negotiation, SLA and QoS monitoring mechanisms. Section 4.4.1 discusses the architecture proposed.

4.4.1. GUISET QoS-aware Web service Selection Architecture

This section presents the GUISET QoS-aware Web service Selection architecture which provides a solution to the issue of QoS-aware in service discovery and selection. Figure 4.2 presents the architecture. The architecture has four components and these are: Service Registry, Service Consumer, Service Provider and the GUISET QoS-aware Broker. The role of each component in the architecture is discussed below.

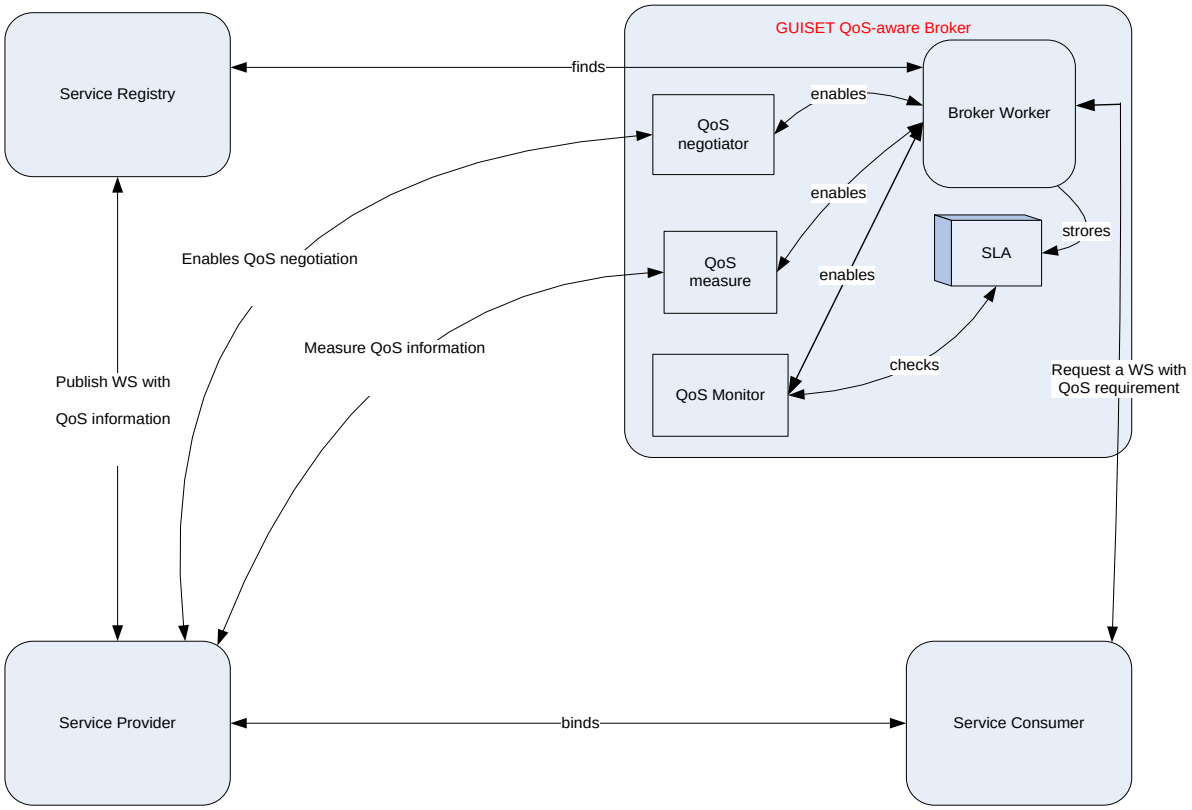

Figure 4.2: GUIEST QoS-aware Web service Selection Architecture

4.4.1.1.Service Registry component

The Service registry component allows service providers to publish their Web services. To have a QoS-aware Web service provisioning infrastructure, our service registry has the capability to allow the service providers to specify QoS information of each Web service published. The service registry is also used by the GUISET QoS-aware Broker component to discover and select a Web Service that meets the service consumer's QoS requirements. We used UDDI as our service registry.

4.4.1.2.Service Provider component

The provider is the entity that develops the Web service and specifies QoS information of each Web service. The service provider registers its Web service in the Service registry.

Along with the other information about that service, the provider should also include the QoS metrics. The service provider publishes a minimum and maximum QoS values for each QoS parameter.

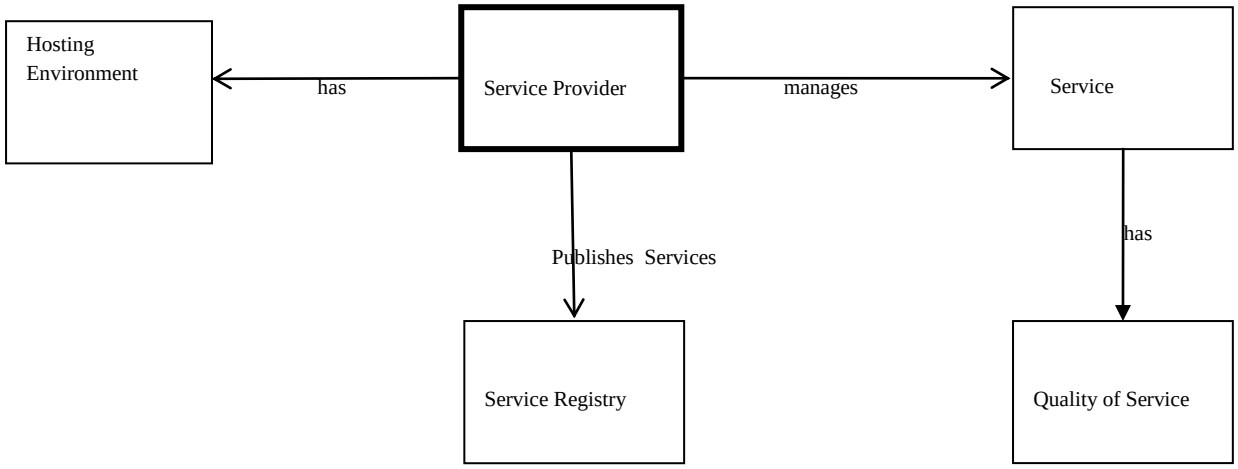

Figure 4.3: Service Providers concepts

Figure 4.3 depicts the whole concept of a Service provider, in the context of our work. Every service provider on our GUISET infrastructure has its own hosting environment. A hosting environment of the service provider may contain several computational and hardware resources like servers. The Service provider is responsible for managing his own Web services and the QoS information. These Web services are published in the Service registry for advertisement.

4.4.1.3.Service Consumer

The client application operates as a service consumer of the advertised Web services. The service consumer sends a Web service request to the GUISET QoS-aware Broker.The request contains the functionality of the Web service being requested (what the Web service does) and also the QoS requirement of the service consumer. The service consumer specifies the expected QoS parameter value and also specifies the worse value it can accept for that QoS

parameter. Then the GUISET QoS-aware Broker searches the service registry to find a set of Web services that can provide the functionality required by the customer, and selects the most appropriate Web Services able to satisfy the customer required QoS.

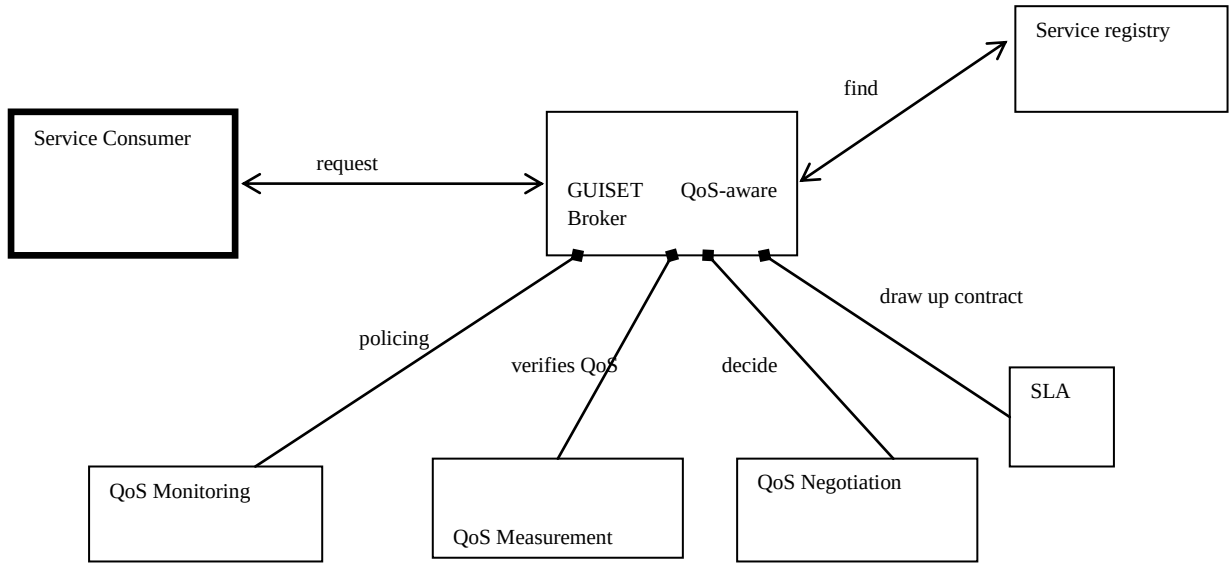

Figure 4.4 Service Consumer concepts

Figure 4.4 shows the whole concept of our Service consumer. Here the Service consumer relies on the QoS-aware Broker for QoS management. This management includes; QoS measurement for making sure that each advertised QoS attribute is measured before the service consumer binds to that service, and QoS negotiation that enables the service consumer to negotiate with the service provider for some QoS constraints. Then the GUISET QoS-aware Broker creates the SLA when the agreement between the two parties has been reached. Lastly, QoS monitoring, which deals with monitoring any violation of QoS attributes which the service provider and consumer agreed upon.

4.4.1.4.GUISET QoS-aware Broker (G-Broker)

The functionalities that the proposed G-Broker t performs are:

1. Accepts incoming requests from service customers.
2. Discovers, through the Service registry, functionally similar Web Services.
3. Selects the most appropriate Web Services able to satisfy the customer required QoS.
4. Measures QoS information the Web service provider guaranteed to deliver.
5. Negotiating QoS between the web service consumer and the web service provider.
6. Creates an SLA, which serves as a contract between the service consumer and service provider about the QoS constraints agreed upon.
7. Monitors any violation of the guaranteed QoS information.

To perform all of these capabilities, the GUISET QoS-aware Broker has the following sub components; Broker Worker, QoS negotiator, SLA, QoS measure and QoS monitor embedded in it. Each component is explained below.

A. Broker Worker

The Broker Worker component selects a Web service that satisfies the service consumer's QoS requirement and calls the other component that includes QoS measure, QoS negotiator, SLA and QoS monitor component. The Broker Worker receives a Web service request from the Web service consumer;the request contains the functionality which the requested Web service performs and also the service consumer's QoS requirements. Then the Broker Worker component discovers from the Service registry all the Web services which exhibit similar functionally. Then it selects one Web service that meets the service consumer's QoS requirement specified during the time when a Web service request was sent. After selecting

one Web service that meets the service consumer's QoS requirements, the Broker Worker component enables the QoS measure component to measure the true values of the QoS parameters from the service provider's side. The Broker Worker can create SLA and also enable QoS monitoring. The following Sections (B, C, D and E) discuss in details all the components that the Broker Worker interacts with.

B. QoS measure component

After the Broker Worker has selected the Web service that meets the service consumer's QoS requirements,since the selection was conducted based on the advertised QoS information by the service providers, a mechanism to validate the QoS information is needed. Some service providers may publish inaccurate QoS information in order to attract more service consumers, this QoS measure component is used to measure the QoS information of the selected service provider to verify the correctness of the QoS parameters' values. The QoS measure component is initiated by the Broker Worker before the service consumer bind to the Web service.

C. QoS negotiator component

The QoS negotiator component uses the measured QoS values that are from the QoS measure component to establish QoS negotiation. Rafeal and Carlos (2006) stated that the *Internet*'s application and traffic are known for being dynamic; therefore it is highly unlikely that the measured QoS values can be exact as the service consumer's QoS requirements. The Broker Worker calls this QoS negotiator component to start a negotiation process that is based on the measured QoS values.

D. SLA component

At the end of any QoS negotiation process, when an agreement between the service provider and service consumer about the QoS constrains which were negotiated has been reached, an SLA is created. The Broker Worker calls the SLA component to create the SLA which serve as a contract obligation between the service provider and consumer, which specifies both the functionality of the service to be offered and the constraint (such as response time, reliability, security, performance, etc.) of the quality of service.

E. QoS monitor component

After the service consumer and service provider have negotiated and signed the SLA then the binding process starts. During this service provisioning process it is essential for the Web service consumer to monitor for any infringement of the agreed QoS commitments that is saved in the SLA. We have developed a QoS monitor component that is part of our GUISET QoS-aware Broker to monitor any kind violation of agreed QoS parameter.

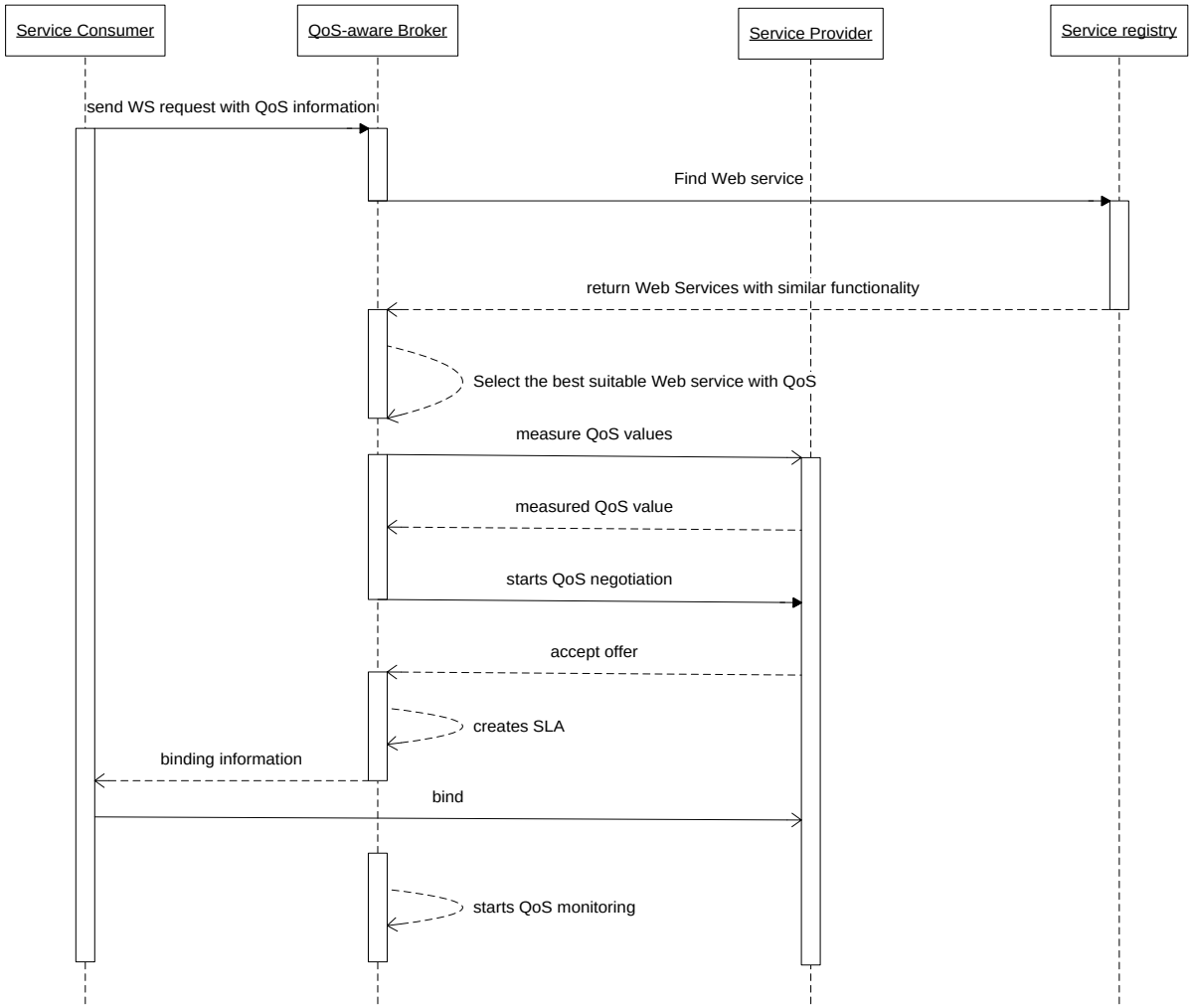

Figure 4.5: Interaction DiagramoftheGUISETcomponents

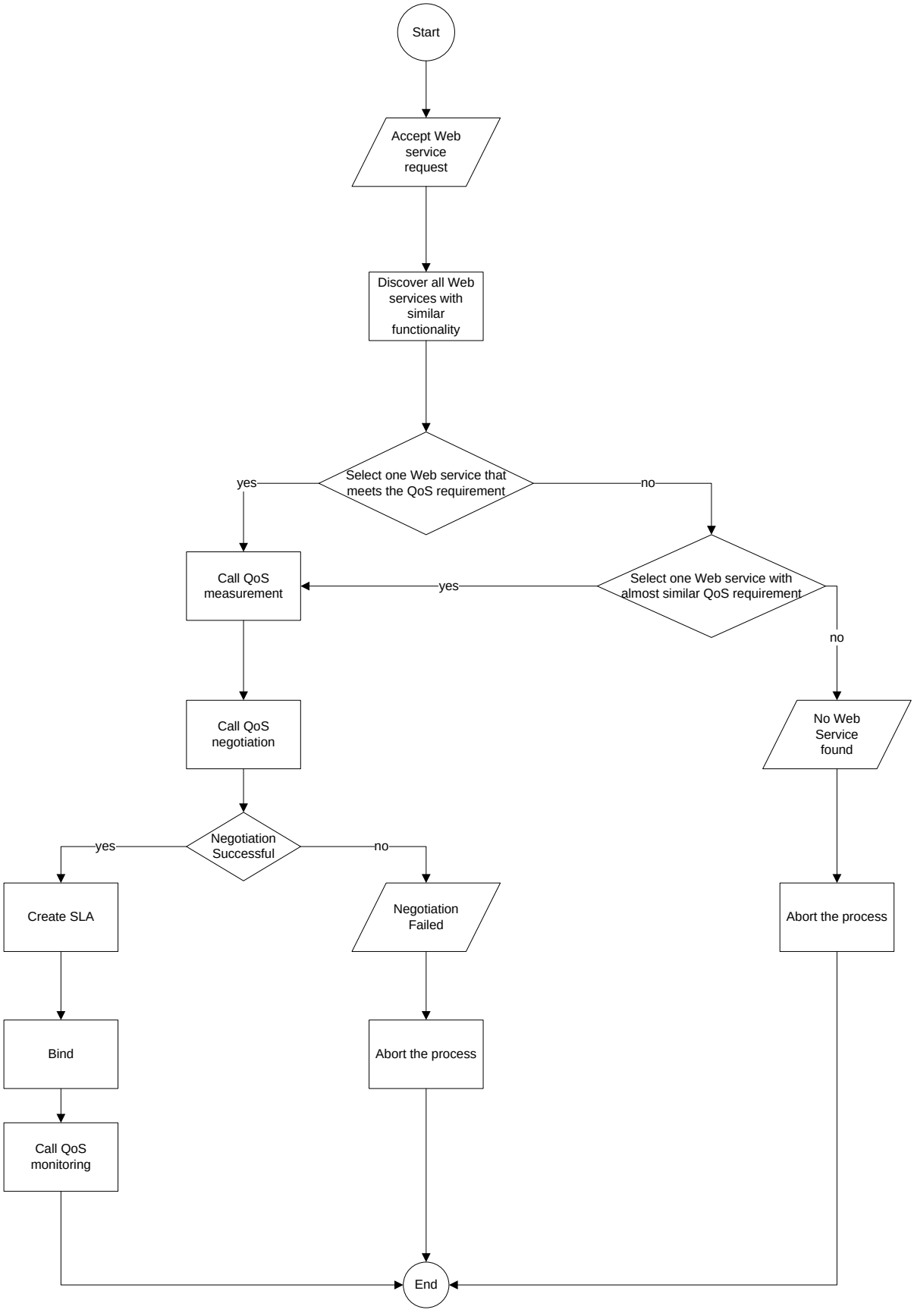

Figure 4.6: Algorithm for QoS-aware Web service provisioning Architecture

Figure 4.5 depicts the interaction diagram of the components of the GUISET QoS-aware Web service Selection Architecture. For reducing complexities in the diagram, all the components discussed in subsections A, B, C, D and E, are represented by QoS-aware Broker in the diagram, as we have already mentioned that the GUISET QoS-aware Broker has these components embedded in it. Figure 4.6 depicts the algorithm for the GUISET QoS-aware Broker.

4.4.2. The G-Broker's QoS-aware Web service Selection Approach

In Section 4.3.1 we discussed how the *tMotel* is used to store QoS information of each Web service in a service registry (UDDI registry). Service providers publish Web services into a service registry and also state the QoS guarantees of each Web service. As we have mentioned that the G-Broker acts as a mediator between the service provider, service consumer and the service registry. The service consumer sends a Web service request to the G-Broker. This Web service request contains the functionality of the Web service being requested and the QoS requirements of the service consumer. Then the G-Broker discovers all the Web services that match the functionality of the Web service request specified by the service consumer. The G-Broker then uses the QoS requirements specified by the service consumer to select one Web services amongst the group of Web services that offer similar functionality. During the QoS-aware Web service selection process, the G-Broker selects all the Web services that have the QoS guarantees that match with the QoS requirements specified by the service consumer. For example, if the service consumer specified response time as QoS requirement, the G-Broker would group all the Web services with response time as a QoS guarantee. Then the G-Broker would search for a Web service with a QoS parameter value that matches exactly with the QoS parameter value specified by the service consumer. If an exact match is not found the G-Broker selects one Web service with QoS

parameter value that is almost equal or closer to the QoS parameter value specified by the service consumer. In a case where the service consumer has specified more than one QoS parameter, the G-Broker would require the service consumer to assign priorities on each and every QoS parameter. The QoS parameter with the highest priority will be used in the search.

After selecting the Web service that meets the service consumer's QoS requirement the G-Broker performs QoS measurement. The QoS measurement mechanism is used to verify the correctness of the QoS information published by the service provider. This is because some service providers may publish false QoS information to attract more service consumers. In QoS measurement a ping or probe program is used to measure the QoS properties and their respective values. A *Uniform Resource Locator* (url) of the selected service provider is used in the ping program. With this ping program one can be able to measure a number of QoS properties like response time, availability, throughput, etc Heiko*et al*., (2003). Formulas that are used to calculate response time, availability, and throughput were discussed in Section 4.3.2 above. After the QoS measurement the G-Broker enables QoS negotiation and SLA creation mechanisms which were presented in Section 4.3.3, Section 4.3.4 respectively. Then G-Broker allows the service consumer to bind to the service provider, then G-Broker calls the QoS monitoring that checks for any violation of the agreed QoS guaranteed. The QoS monitoring mechanism was presented in Section 4.3.5.

4.5. Summary

In this chapter, we have described the design of QoS-ware Web Selection framework for our Grid based infrastructure (GUISET). We have also suggested that the QoS-aware Web service provisioning infrastructure must combine QoS negotiation and SLA, QoS measurement and QoS monitoring. These capabilities must be combined in a Broker. Our proposed GUISET QoS-aware Web service Selection framework can work or can be used on

any Grid based infrastructure. As stated in section 1.5, the goal of this project is to design a GUISET middleware layer capability as a generic framework service for QoS-driven Clients. This has been partly achieved in this chapter with the detailed description of the model design and algorithm.

CHAPTER FIVE

MODEL IMPLEMENTATION AND

EXPERIMENTATION

5.1. Introduction

The current SOA environment lacks the capability to support QoS, since its discovery mechanism is only based on the functionality of the Web service (what the Web service does). In Chapter 4 we presented a solution approach for the problem above. Then this Chapter describes the simulation of our proposed QoS-aware Broker for our GUISET infrastructure. As we explained in the previous chapters, the goal of this work is to design a GUISET middleware layer capability as a generic framework service for QoS-driven clients. However, the proposed model can be used in any QoS-aware Web service selection environments. The objectives of this study were revisited in chapter four in order to ascertain how these were to be achieved in the model design. This chapter presents the simulation setup, simulation results of the proposed GUISET middleware layer called GUISET QoS-aware Broker (G-Broker) and also analyse the results obtained, in order to prove the concepts discussed in this dissertation.

In demonstrating the performance and the behaviour of our model, we present the assumptions considered in Section 5.2, the simulation environment and the system interface in Section 5.3. In Section 5.4, we present performance evaluation of the proposed model. Section 5.5 presents the experiment results analysis. Then in Section 5.6 we discuss the results and provide summary in Section 5.7.

5.2. Basic assumptions of the simulation model

In developing our simulation, following are the assumptions we considered due to the duration of this project and considerations of the environment where it will function.

1. We assume that all the deployed Web service in the GUISET infrastructure have some QoS guarantees attached to them.
2. We also assume that all the deployed Web Services have similar functionality.
3. We assume that every service provider will describe a minimum and a maximum QoS values for each QoS parameter.

5.3. Simulation Setup and Simulation Environment

Our GUISET-based service provisioning infrastructure is envisioned to provide a platform for Web service providers to advertise their Web services. In this infrastructure the service providers would publish any number of Web services. The fact that there are many service providers and any service provider can publish any number of Web services, the chances of having many Web services offering similar functionality is very high. If there are many Web services that offer similar functionality, the act of selecting one Web service among that group is a challenge. Quality of service (QoS) proves to be an essential mechanism to differentiate those Web services that offer similar functionality. We developed a QoS-aware infrastructure, where a service provider specifies their QoS guarantees for each and every Web service they publish to the infrastructure. This enables easy differentiation of Web services that offer similar functionality and enables the service consumers to specify their QoS requirements when requesting a Web service as it is envisioned in the GUISET infrastructure.

Our simulation environment was designed to form a perfect competition. It is based on the idea that the service providers need to formulate to QoS-aware offers in order to derive the highest possible profit from their business and on the other hand Service consumers expects to experience an excellent value such as higher availability, low response time and higher throughputwhen they are using a particular Web service. In this simulation there are many different service providers. Each service provider can publish any number of Web services and supply QoS guarantee for each. The simulation environment consists of a platform that is used by the service providers to publish their Web services to the UDDI as well as QoS guarantees to any of their Web services. In addition, we simulated the GUISET QoS-aware broker (G-Broker) and the Classical QoS-aware broker (C-Broker). The G-Broker and C-Broker are discussed in the following subsections.

5.3.1. The GUISET QoS-aware broker (G-Broker)

This broker was defined in Chapter 4. A simulation of the proposed architecture was developed, in order to observe if G-Broker offers any performance benefit over the C-Broker during dynamic selection of appropriate QoS-aware Web services. The G-Broker selects a Web service from the UDDI registry and measure the QoS information of the service provider to verify the correctness of the QoS information published by the service provider, and then it negotiates with the service provider based on the measured QoS information. It also has the Service Level Agreement (SLA) component that stores the agreed contract between the service provider and service consumer. The SLA is used by a QoS monitor component to check for any violated QoS agreement at the end of the service provisioning.

5.3.2. The Classical QoS-aware broker (C-Broker)

The service providers solely create the Web service(s) and assign QoS information to each and every Web service published in the UDDI registry. The C-Broker deals with accepting QoS-aware Web service requests from service consumers and selecting QoS-aware Web services that matches the service consumer's QoS requirement, without measuring the values of the advertised QoS parameter since some published QoS information may be incorrect. This broker has no negotiation mechanism in case the required QoS is not found and it has no SLA and it has no QoS monitoring techniques to find out whether the service provider did deliver the QoS guarantees or not.

5.3.3. Simulation Environment

To mimic a distributed system in our own simulation, we used three different computers. First computer was a laptop machine running Windows Vista Business. The machine was an Intel (R) Pentium (R) processor with a Dual CPU T3400 @ 2.16GHz 2.17 GHz processor and 2.00 GB of RAM. This machine was used to represent our server; we used GlassFish v 3.1 Domain and Derby as our database. We deployed jUDDI 2.1 as our service registry using Netbeans 6.9. The second computer servedasa service provider. It was a Desktop machine running Windows XP Inter (R) Pentium (R) 4 CPU 3.20GHz processor and 1.24 GB of RAM). We used Netbeans 6.9 to build a platform that is used by the service provider to publish Web services to our service registry.

Our third computer was also a Desktop machine running Windows XP Inter (R) Pentium (R) 4 CPU 3.20GHz processor and 1.24 GB of RAM). In this third computer we used Netbeans 6.9 to build our proposed QoS-aware Broker (G-Broker) that has QoS Measurement, QoS monitoring QoS Negotiation and SLA. We also build a platform that a service consumer can use to send a Web services request that contains the QoS requirements of that service

consumer. This client application (platform) forwards the request being sent by the service consumer to our QoS-Broker. We also build the Classical QoS-aware Broker (C-Broker) that is different from our proposed QoS-aware Broker as discussed in Section 5.5.5. Both brokers were running on our third computer. All these three computers were connected via a wireless network. The application consumed less than 70.4 MB of hard-disk storage.

5.4. Performance Evaluation

This section describes the performance comparison between our proposed G-Broker for Web service selection and the C-Broker that is presented in Tain *et al* (2005); and Rajendran and Balasurbramanie (2009).As we mentioned before that the main aim of this study was to develop a QoS-aware Web service selection Broker that would be used in our GUISET infrastructure. The G-Broker selects Web services from the registry on behalf of service consumers. The selected Web service must satisfy the service consumer's QoS requirement specified by the service consumer during the request. We compared the performance of the G-Broker with that of the C-Broker. The following performance metrics were used for the comparison:

(i) Consumer Satisfaction

(ii) Scalability

5.4.1. Consumer Satisfaction

Consumer satisfaction is achieved when the selected service did deliver the QoS guarantees that meet the service consumer's minimum QoS requirements. A service consumer creates a *Satisfaction threshold utility* (D_u)

$D_u = (QoS_{max})$

Where QoS_{max} is the maximum quality of service value the service consumer can afford or we can say it is the worst QoS value the services consumer can accept.

Once the service consumer binds to the selected service provider, we observed the QoS value, which the service provider delivered X_i. In other words X_i is the quality of experience (QoE).

The service consumer is said to be satisfied if X_i is less or equal to D_u. If the condition is not met, then the service consumer is said to be dissatisfied.

Pass *n* number of requests to the simulator

Observe X_i

If $X_i \leq$ *Satisfaction Threshold utility* D_u

then the service consumer is satisfied

else the service consumer is dissatisfied

Observe all the X_i that satisfied the service consumer

Figure 5.1*Comparative Analysis of Consumer Satisfaction for G-Broker and C-Broker*

5.4.2. Scalability

Scalability is the ability of a system, network, or process, to handle growing amounts of work in a graceful manner or its ability to be enlarged to accommodate that growth. We use Transaction time and Response time to observe that how each broker would scale when the number of Web services published in a service registry increases. Transaction time is the

amount of time it takes for each Web service request to be processed, it is also called execution time. Response time is the time taken to send a request and to receive the response.

Publish n number of Web services

Pass *n* number of requests to the simulator

Record the transaction time at each interval

Figure 5.2Capturing of Transaction time of Requests for G-Broker and C-Broker

Publish n number of Web services

Pass *n* number of requests to the simulator

Record the response time at each interval

Figure 5.3 Capturing of Response time of Requests for G-Broker and C-Broker

5.5. Experimental Results Analysis

This section describes in detail all the experiments and analysis and discusses the results that were obtained.

5.5.1. Experiment I: Consumer Satisfaction Level for G-Broker and C-Broker

An experiment was conducted to determine the consumer satisfaction derived from each approach. The experiment was set up as follows: 100 consumer requests were randomly generated with specified QoS parameters. These requests were made to a database with 40

published Web services. The selection of the appropriate services was then done by the two schemes. Both the requests and the satisfaction results generated by each scheme were observed and analyzed. To compare the performance of the two approaches, a *z-test* of difference between two proportions was conducted upon the observed results. The hypothesis was that the G-Broker selects appropriate Web services that better satisfies the consumer's QoS requirements than C-Broker.

i. *The Hypothesis*

Null hypothesis: H_0 : There is no difference in consumer satisfaction of the G-Broker and C-Broker selection process. Alternative Hypothesis: H_1 : G-Broker select Web services that better satisfies the consumer than the C-Broker.

i. *The Decision Criteria*

The critical value of $z = Z_{tab}(0.05)$=1.6449. If the calculated test statistic, Z_{calc}, is greater than the tabulated statistic, Z_{tab} ($Z_{calc} > Z_{tab}$), we reject the null hypothesis.

ii. *Data Gathered*

	G-Broker	C-Broker	$\hat{P}$
Observations (p)	88	39	0.635
Sample Size (n)	100	100	

Table 5.2: Results from hypothesis testing

iii. Hypothesis Evaluation

The Test statistic is given by:

$$Z_{calc} = \frac{P_{gi} - P_{ci}}{\sqrt{\hat{P}(1-\hat{P})(\frac{1}{n_{g1}} + \frac{1}{n_{c1}})}}$$

Where: $\hat{P} = \frac{P_{g1} + P_{c1}}{n_{g1} + n_{c1}}$

$$P_{gi} = \frac{P_{g1}}{n_{g1}}$$

$$P_{ci} = \frac{P_{c1}}{n_{c1}}$$

Since Z_{calc} =7.3657 >$Z_{tab}(0.05)$ =1.6449, we reject, the null hypothesis,H_0 ,and conclude that the GUISET QoS-aware broker Web service selection process performs better than the Classical QoS-aware broker.

This is due to the fact that the G-Broker has the capability to measure the QoS values and enable negotiation based on the measured QoS values before allowing the service consumer to bind to the selected service. This act enable the G-Broker to select an appropriate service provider that will deliver the QoS requirements of the service consumer, and eliminate the cases where the service consumer binds to the service provider that won't deliver the QoS guarantees that he promised to deliver. The G-Broker also keeps a Service Level Agreement (SLA) that acts as a contract agreement made by both the service consumer and service provider about the QoS values agreed upon during the negotiation. This SLA is used at the end by the QoS monitoring capability that checks if the agreed QoS values on the SLA were

honoured or not. It is based on these capabilities that the G-Broker selects and enables the service consumers to bind to service providers who would honour their QoS requirements.

5.5.2. Experiment II: Scalability for G-Broker and C-Broker

Having proved that the G-Broker selects an appropriate Web service that satisfies the QoS requirements of the service consumer as compared to the C-Broker and the following experiment was carried out in order to investigate the scalability of our G-Broker. The reason for investigating scalability was motivated by the fact that the G-Broker proved to select a appropriate Web services that satisfy the service consumer's QoS requirements than the C-Broker. This was because the G-Broker has more capabilities than the C-Broker. Therefore, it is necessary to investigate whether these capabilities render the G-Broker not scalable. Brent *et al* (2000) defined scalability as the ability of a computer application or product to continue to function well when it is changed in size or volume in order to meet the user need.

The design was to test the response time and transaction time of a Web service request for a given number of Web services published in the UDDI registry. Every Web service published has QoS guarantees attached to it. We started by publishing 20 Web services. We sent 10 requests to our proposed G-Broker; each Web service request has QoS requirements specified by the service consumer. The average response time and the average transaction time were recorded. The same procedure was followed on the C-Broker.

The average response time is given by the sum of n service request response time divided by the n requests sent as presented Equation 5.1

$$AV\ response\ time\ = \frac{1}{n}\sum_{i}^{n} RT \tag{5.1}$$

And the average transaction time is given by the sum of n service request transaction time divided by the n requests as presented by Equation 5.2.

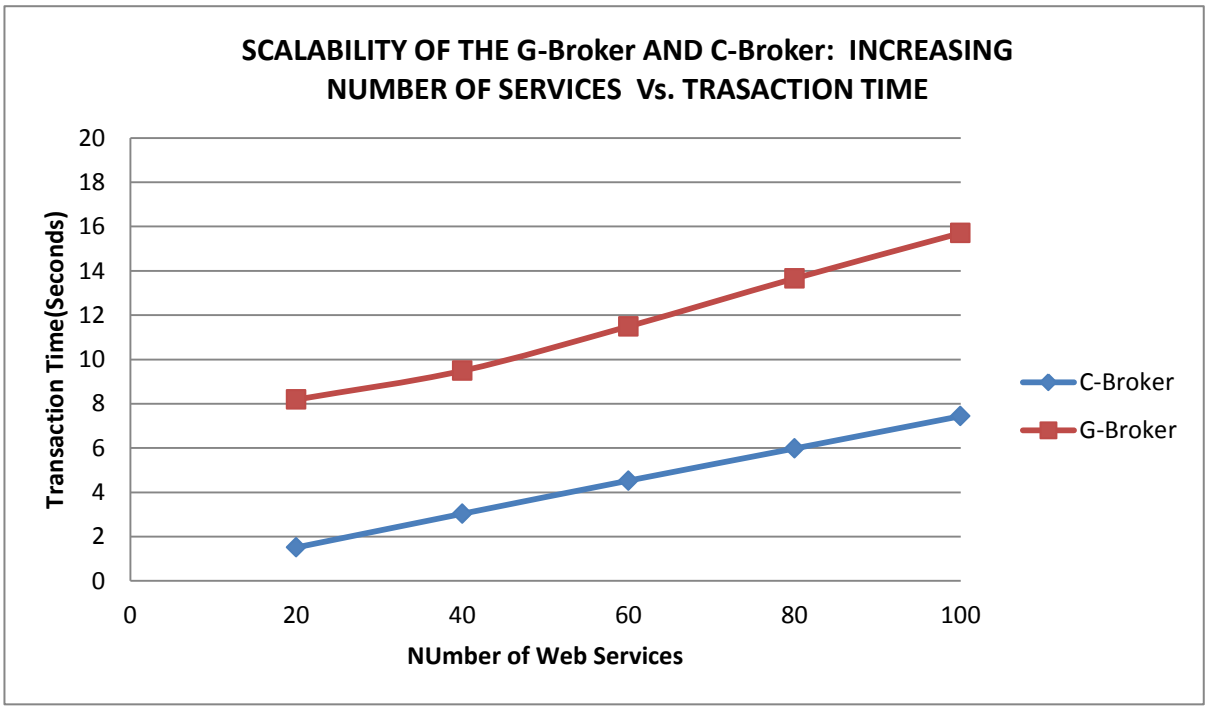

Figure 5.4 Transaction Time vs Number of Web Services for Scalability of G-Broker and C-Broker

$$AV\ transantion\ time = \frac{1}{n}\sum_{i}^{n} TT \tag{5.2}$$

We increased the number of Web services by 20, starting from 20, 40, 60, and 80 until 100 Web services. We recorded the average response time and transaction time using the above equations on each time we increased the number of Web services.

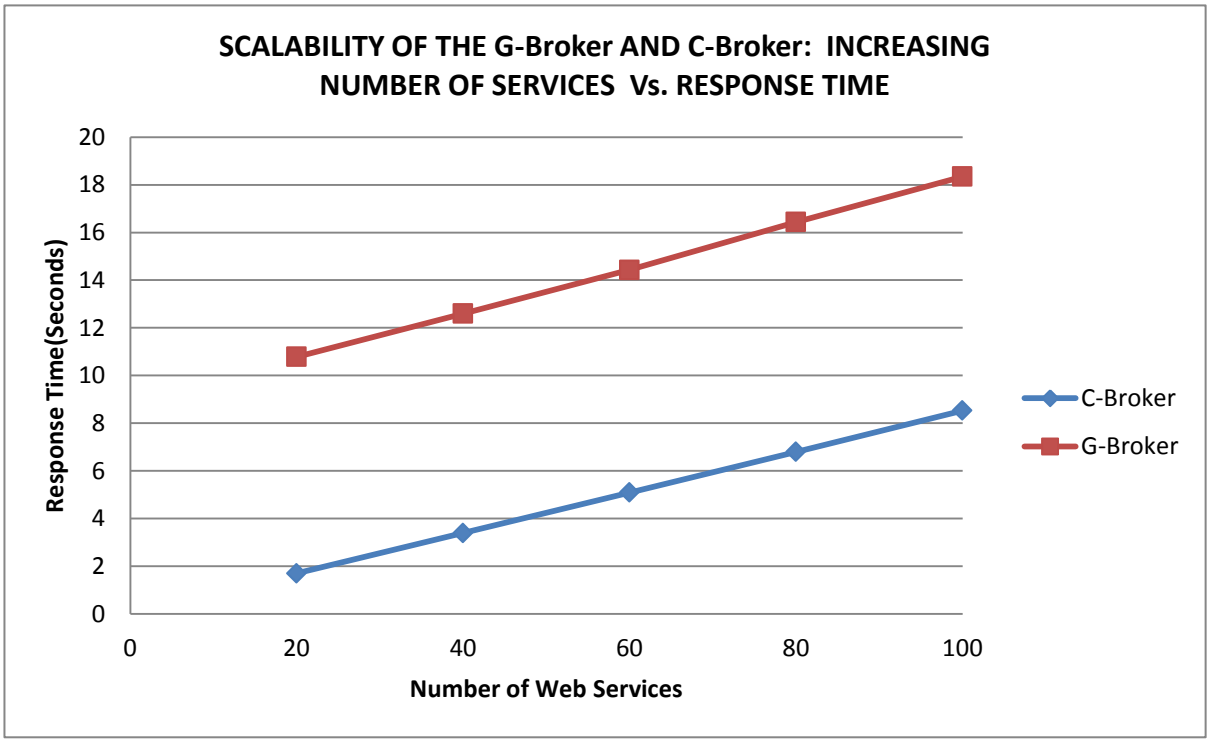

Figure 5.5 Response Time vs Number of Web Services for Scalability of G-Broker and C-Broker

The results of the experiments discussed in Section 5.5.2 are represented by the graphs in Figure 5.4 and Figure 5.5. Figure 5.4 presents the results of the transaction time versus the number of Web services. The results in Figure 5.4 show that the transaction time of both G-Broker and C-Broker increases every time when we increase the number of Web services published. Figure 5.5 depict the findings of our experiment for response time versus the number of Web services published. Again the graph in Figure 5.5 shows that the response time increases as the number of Web services also increases on both the G-Broker and C-Broker. In both graphs (Figure 5.4 and Figure 5.5) we observed that the number of Web services published is directly proportional to the transaction time and response time. Therefore; these experiments show that the G-Broker is scalable as the C-Broker is. The only difference between the two brokers is that the G-Broker has a large response and transaction time compared to the C-Broker as was expected, given the fact that G-Broker has four more

capabilities (QoS measurement, QoS negotiation, QoS monitoring and SLA), that the C-Broker does not support.

5.5.3. Experiment III: Performance Evaluation of a G-Broker with one Capability

In Section 5.5.2 we investigated the scalability of the GUISET QoS-aware Broker (G-Broker) and the results in Figure 5.4 and Figure 5.5 show that the G-Broker is scalable as the C-Broker. However, the G-Broker proved to have a response time and transaction time that are five times higher than the Classical QoS-aware broker (C-Broker). This was expected since the G-Broker has these capabilities QoS measurement, QoS negotiation, QoS monitoring and SLA while the C-Broker deals with selecting Web service based on the QoS requirements of a service consumer only. Therefore, the following experiment was carried out in order to investigate that which capability has the greatest effect on the response time and transaction time of the G-Broker, and also to observe which capability has the smallest effect on the response time and transaction time of the G-Broker. This would make us to be aware of how much does each capability contribute to the transaction time and response time of the G-Broker.

For experimentation purposes; we built our prototype in such a way that we can enable the QoS-aware broker to have one capability working, while all the three capabilities are disabled. Having this kind of flexibility allowed us to run experiments for observing the response time and transaction time when our broker has one capability amongst the four capabilities. In each experiment we increased the number of published, while where sent 10 requests to the QoS-aware broker. Then we recorded the response time and transaction time

on every

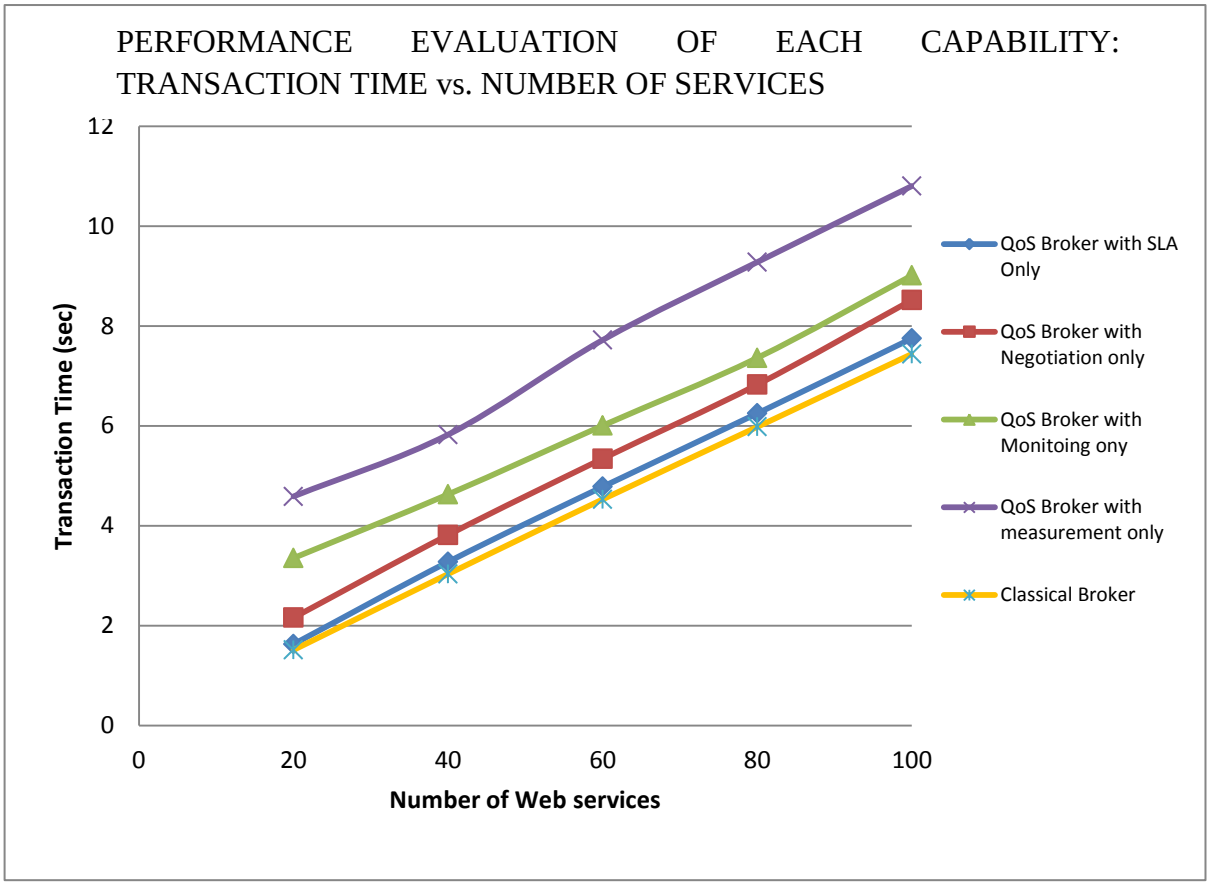

Figure 5.6 Transaction Time vs Number of Web Services for Performance Evaluation of a G-Broker with one Capability

response. Then we calculated the average response time and average transaction time using the equation 5.1 and 5.2 respectively. We started with publishing 20 Web services and we increased by 20 Web services until 100 Web services.

Figure 5.6 shows the results of transaction time when our QoS-aware broker has one capability added to it. These capabilities include QoS negotiation, SLA, QoS measurement and QoS monitoring as discussed above. The results show that QoS measurement is the one that has the highest transaction time as compared to SLA, QoS negotiation and QoS

monitoring. The transaction time that is observed when our QoS-aware broker performs QoS

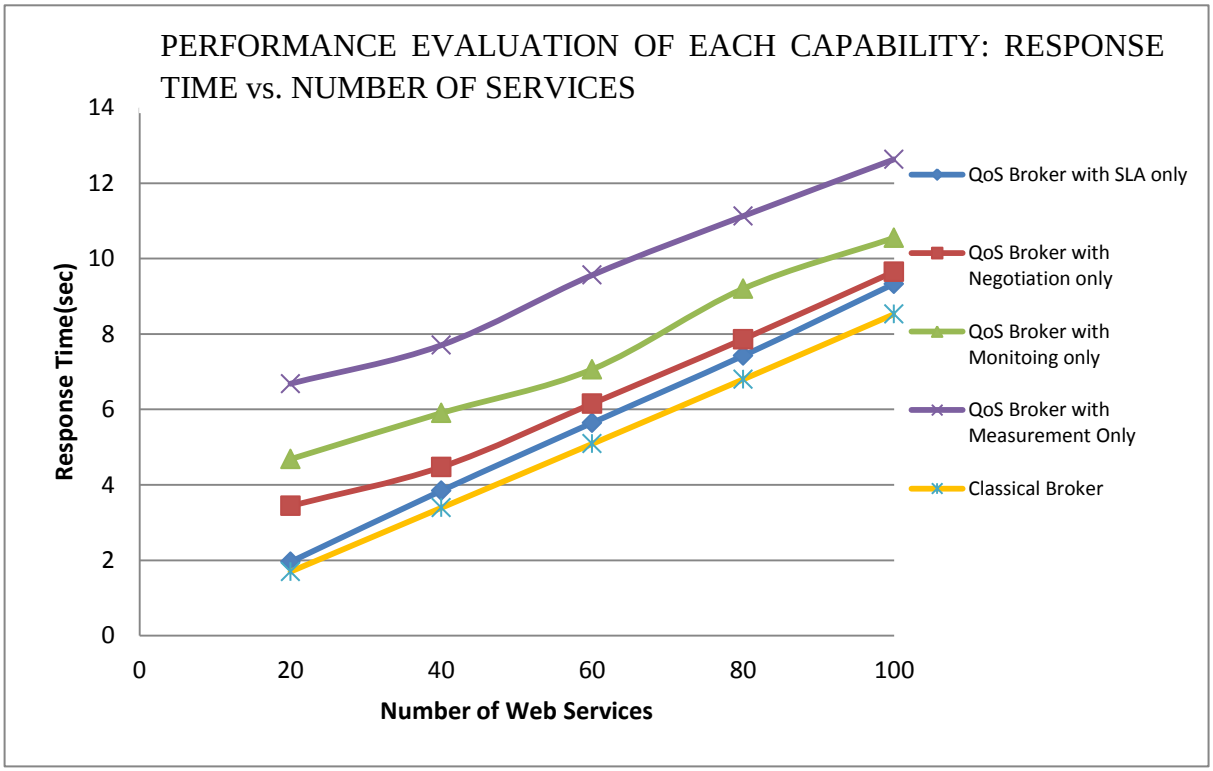

Figure 5.7 Response Rime vs Number of Web Services for Performance Evaluation of a G-Broker with one Capability

Measurement only, starts at 4.6 seconds when there were 20 Web services published and 10.802 seconds when there were 100 Web services published. The QoS Broker with SLA only prove to have smallest transaction time as compared to the other three broker, its transaction time starts at 1.6 seconds when there were 20 Web services published and it became 7.75 seconds at 100 Web services.

The results in Figure 5.7 depict the response time of the four different brokers who are having one capability. These capabilities include QoS negotiation, SLA, QoS measurement and QoS monitoring as discussed above. The QoS Broker with measurement capability only, proved to

have the greatest response time as compared to the other three brokers and including the Classical Broker. The response time of the Broker with QoS measurement capability only was 6.68 seconds at 20 Web services and it was 12.63 seconds at 100 Web services. The QoS Broker that has SLA only showed the smallest response time as compared to the other three brokers. Its response time was 1.958 seconds when there were 20 Web services and it became 9.33 seconds at 100 Web services. Even though the QoS Broker with SLA capability showedthe smallest response time as compared to the other three brokers, but its response time is higher than that of the Classical Broker.

5.5.4. Experiment IV:Performance Evaluation of a G-Broker with two Capabilities

In the previous Chapters we discussed the importance and the role that each capability plays in a QoS-aware Web service provisioning infrastructure. And we mentioned that none of the existing QoS-aware brokers implements all the four capabilities combined together, a QoS-aware broker model may implement various criteria depending on each organization's requirement. In Section 5.5.3, we conducted experiments to find out which capability amongst the four (QoS measurement, QoS monitoring, QoS negotiation and SLA) has the biggest effect on transaction time and response time of the G-Broker. In that experiment the broker was performing one capability. The results showed that the QoS measurement seems to have the biggest effect and the SLA seem to have the smallest effect. In this experiment, we wanted to determine the greatest transaction time and response time that the two combined capabilities causes (QoS measurement, QoS monitoring, QoS negotiation and SLA). We also determine that which combination has the transaction time and response time that is closer to the transaction time and response time of the C-Broker (C-Broker being the broker with the better transaction time and response time). The results of the experiment will help us to be able to advise any organization about the transaction time and response time,

which they can expect when they want to implement a broker with two capabilities. The same idea also applies in the experiment conducted in Section 5.5.5 where we used a combination of three capabilities, to also determine the effect on performance when a QoS-aware broker implements three capabilities. This will also help us to be aware of the performance issues of a broker with three capabilities. There are six combinations that can be drawn from the four capabilities.

1. QoS measurement and QoS monitoring
2. QoS measurement and QoS negotiation
3. QoS measurement and SLA
4. QoS monitoring and QoS negotiation
5. QoS monitoring and SLA
6. QoS negotiation and SLA

As we mentioned above, we built our QoS-aware broker in way that one can enable or disable any capability(s) amongst the four capabilities, avoiding a situation where we have to program six QoS-aware brokers with different capabilities for example. Therefore, we conducted experiment on six different brokers equipped with two different capabilities. The experiments were conducted one combination at a time, recording the transaction time and response time a set of published Web service. We started from 20 Web services and increased the number of published Web service by 20 until 100 Web services.

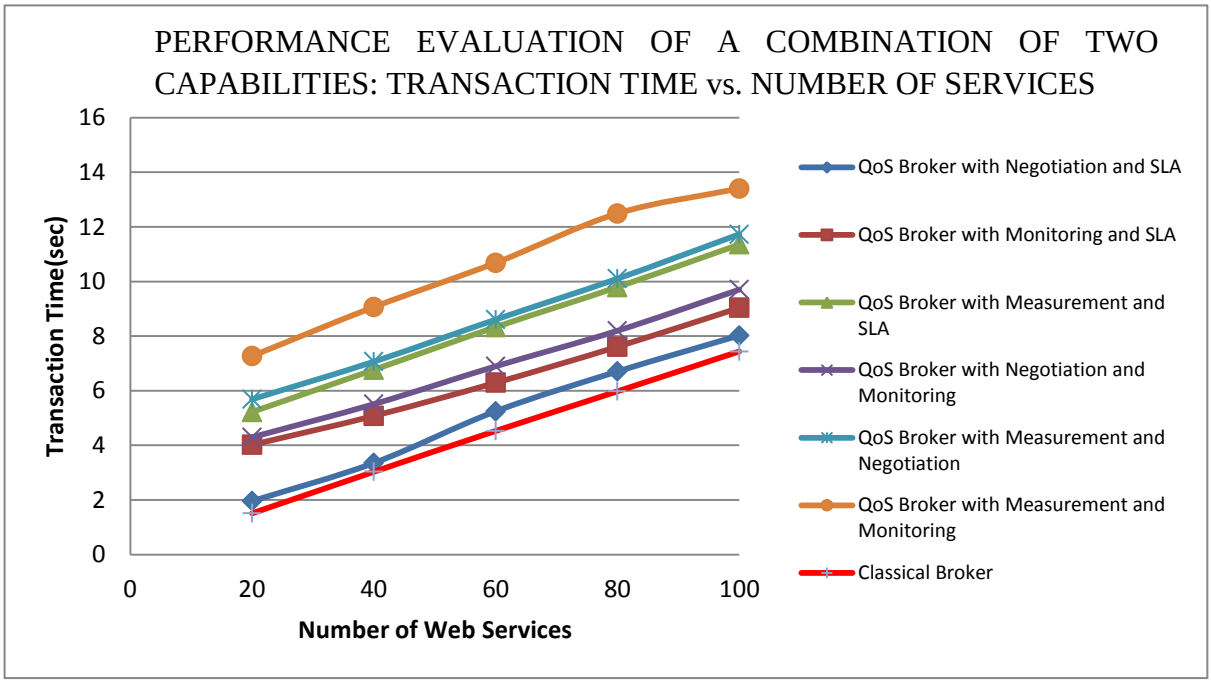

Figure 5.8 Transaction Time vs Number of Web Services for Performance Evaluation of a G-Broker with two Capabilities

Figure 5.8 shows the transaction time of six brokers that performs two capabilities, and each combination that each broker performs is different from each other. As we mentioned that the aim of this experiment is to determine which combination has the highest response time and transaction time, and also to determine which combination has the smallest response time and transaction time. We observed that the combination of QoS measurement and QoS monitoring has a very high transaction time as compared to all the other five combinations. The transaction time of QoS measurement and QoS monitoring starts at 7.2 seconds when there were 20 Web services published and it became 13.4 seconds at 100 Web services. The combination with the lowest transaction time is the QoS negotiation and SLA, its transaction time starts at 1.9 seconds when there were 20 Web services and it became 8.02 second at 100 Web services. The transaction time of the broker with combination of QoS negotiation and SLA is the closer to the transaction time of the Classical Broker.

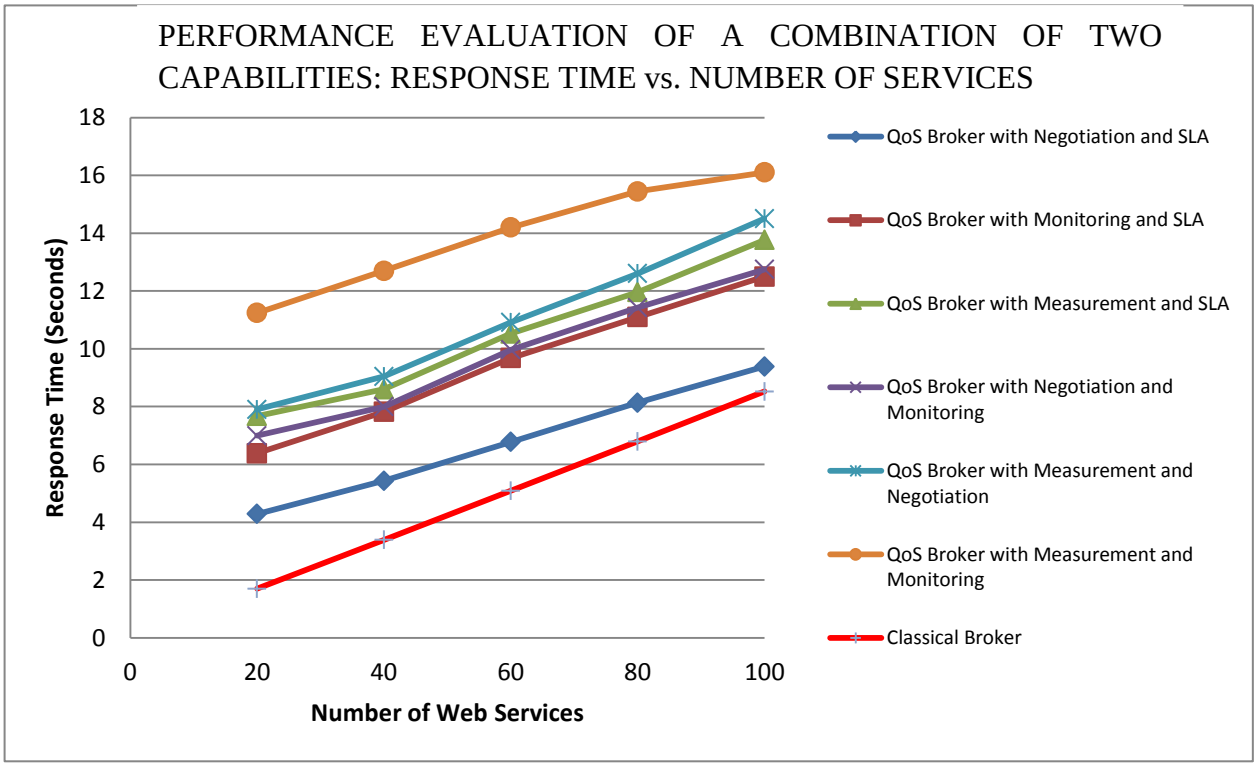

Figure 5.9 Response Time vs Number of Web Services for Performance Evaluation of a G-Broker with two Capabilities

Figure 5.9 presents the response time of each combination of two capabilities amongst QoS negotiation, QoS measurement, QoS monitoring and SLA. There are six combinations created from the four capabilities. Amongst the six combinations, the combination of QoS measurement and QoS monitoring proved to have high response time as compared to other combinations. The results show that the response time of QoS measurement and QoS monitoring starts at 11.2 seconds when there we 20 Web services and it became 16.1 seconds at 100 Web services published. The combination with the smallest response time, is the QoS negotiation and SLA with a response time of 4.2 at 20 Web services and end with 9.3 seconds where there were 100 Web services published.

5.5.5. Experiment V:Performance Evaluation of a G-Broker with three Capabilities

The purpose of this experiment was to investigate the transaction time and response time of a QoS-aware broker with three capabilities and determine that which combination of three capabilities has the highest transaction time and response time. And also determine that which combination has the transaction time and response time that is closer to the transaction time and response time of the C-Broker. The results of this experiment would make us to be aware of the transaction time and response time that one can expect when a QoS-aware broker implements three capabilities, this would also help us to advise organisations about the combination which they can choose according to what they can afford. Amongst these four capabilities (QoS measurement, QoS negotiation, QoS monitoring and SLA), there are four combinations of three capabilities which can be drawn from them. Therefore, we had four different QoS-aware brokers with three different combinations, namely:

- A broker with QoS measurement, QoS monitoring and QoS negotiation capability
- A broker with QoS measurement, QoS monitoring and SLA capability
- A broker with QoS measurement, QoS negotiation and SLA capability
- A broker with QoS monitoring, QoS negotiation and SLA capability

We conducted experiments and recorded the performance of each broker. These experiments also helped us to determine, which combination amongst these four has higher performance as compared to the others. The experiments were conducted the same way as experiment I, II, III were conducted, starting with 20 Web services and increase the number of Web services 20 until 100 Web service and record the performance at each stage.

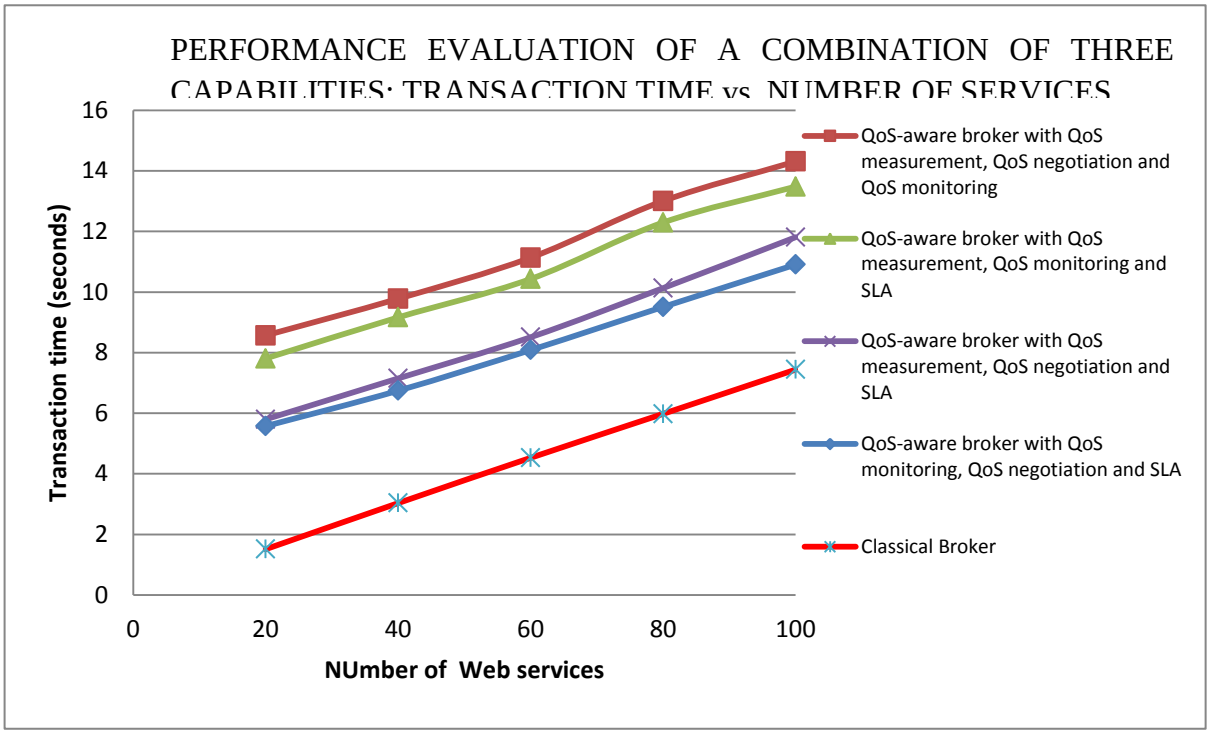

Figure 5.10 Transaction time versus number of Web services for Performance Evaluation of a G-Broker with three Capabilities

Figure 5.10 represents the transaction time of each QoS-aware broker with a combination of three capabilities. The broker with QoS measurement, QoS monitoring and QoS negotiation proved to have higher transaction time as compared to the others brokers. Its transaction time is 8.5 seconds when the number of Web services was 20 and it became 14.3 seconds at 100 Web services. The broker with QoS monitoring, QoS negotiation and SLA proved to have the lowest transaction time as compared to the other three brokers. Its transaction time starts at 5.5 seconds when there were 20 Web services and it became 10.9 at 100 Web services. And the transaction time of this broker is not too high as compared to the transaction time of the C-Broker, as it is three times higher as the C-broker one.

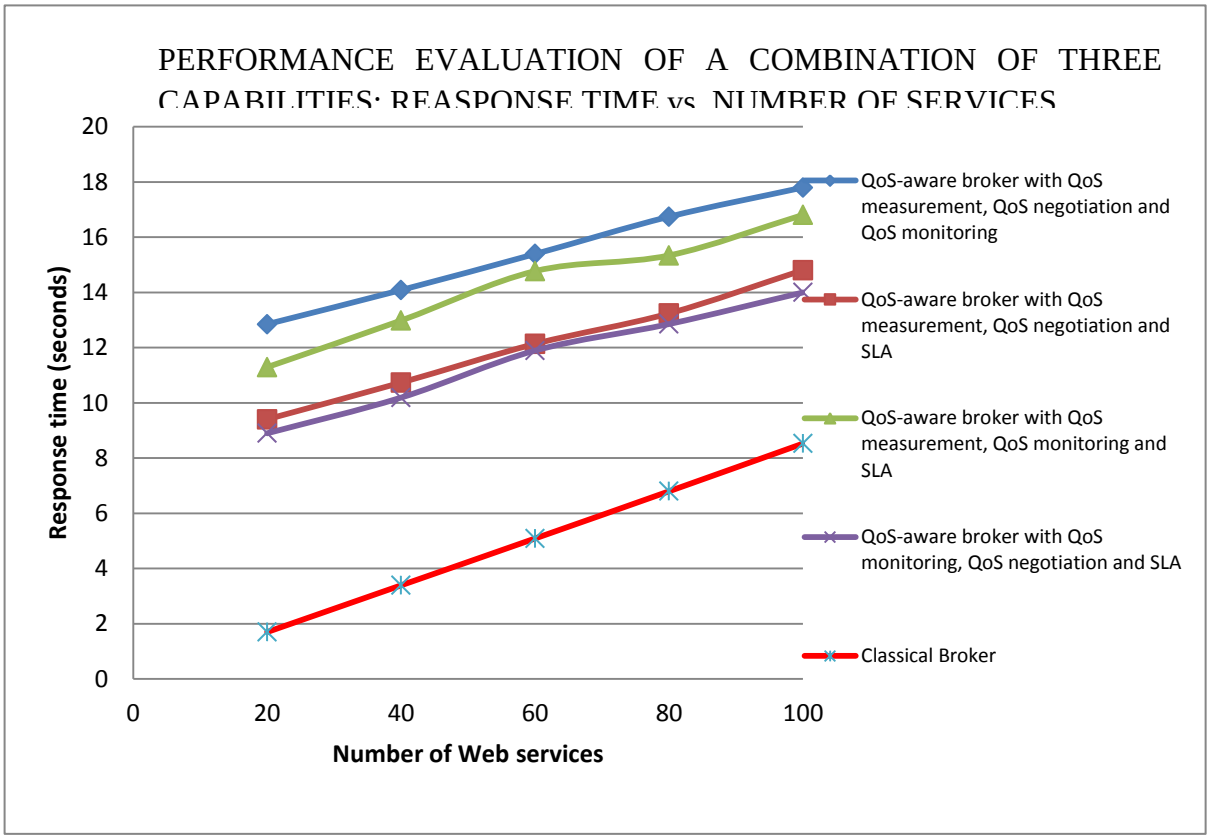

Figure 5.11 Response time versus number of Web services for Performance Evaluation of a G-Broker with three Capabilities

Figure 5.11 shows the response time of each broker. The broker with QoS measurement, QoS monitoring and QoS negotiation capabilities proved to have higher response time as compared to the others brokers. The response time of this broker is 11.3 seconds at 20 Web services and is 16.8 seconds at 100 Web services. The broker with QoS monitoring, QoS negotiation and SLA proved to have the lowest response time as compared to the other three brokers. Its response time is 8.8 seconds at 20 Web services and is 13.9 at 100 Web services

5.6. Discussion of Results

In Section 5.5.4 we presented the experimentation results of consumer satisfaction level for G-Broker and C-Broker. The results showed that the G-Broker selects appropriate Web services that deliver the QoS that satisfies the service consumer as compared to the C-Broker. The reason for the better performance of the G-Broker on consumer satisfaction is because it incorporates the QoS measurement, QoS negotiation QoS monitoring and SLA capabilities. The experimental results on scalability in Section 5.5.5 also proved that the G-Broker is scalable, as Iqan (2010) stated that the Service Oriented Architecture (SOA), is intended to provide scalability. Therefore, the experiments showed that the G-Broker does not bring scalability problems to the SOA as we thought. However, even though the G-Broker performs better than the C-Broker on consumer satisfaction and is also scalable but the response time and transaction time of the G-Broker proved to be five times higher than that of the C-Broker. Since the G-Broker performs more functions than the C-Broker, the high response time and transaction time was expected. Serhani *et al* (2005) and Rajendran *et al* (2010) stated that the performance of the broker can be a key to the success of any proposed architecture. Some users may not want to wait longer time before their request being served.

Owing to this higher response time and transaction time that the G-Broker has, we conducted experiments to investigate that which capability amongst QoS measurement, QoS negotiation, QoS monitoring and SLA has too much effect on the response time and transaction time of the G-Broker, and also observe which capability has least effect. The experimentation results presented shows that QoS measurement is the one that contributes to the response time and transaction time of the G-Broker. This is because, the QoS measurement capability measures the QoS information published by the service provider at run time before the service consumer binds to that Web service, this is because service providers may give incorrect guarantees concerning the QoS of their Web services, especially

when having in mind that weaker QoS guarantees leads to degradation in ranking Jan(2007). In this work, we could have adopted the QoS certification and verification techniques which are conducted by a third party whenever a service provider publishes a Web service, but we learnt from the literature that considering QoS information that was measurement and certified long time ago is not advisable since the internet's application and traffic are known for being of dynamic nature Rafael and Carlos(2006).

The SLA capability proved to have the smallest effect on the transaction time and response time of the G-Broker. As much as we recommend the use of the G-Broker on any QoS-aware Web service selection environment because of its benefits discussed above, but we also acknowledge the disadvantages of high response time and transaction time that the G-Broker brings. Therefore, we conducted experiments where we reduce the number of capabilities which the G-Broker has. We started with a broker that performs two capabilities; we did a combination of two capabilities selected among the four capabilities (QoS measurement, QoS negotiation QoS monitoring and SLA), so that we can observe which combination of two has the response time and transaction time that is closer to the transaction time and response time of the C-Broker, and also observe that which one has a higher response time and transaction time.

The experimented results shows that the cost of combining QoS measurement and QoS monitoring prove to have a very high transaction time and response time compared to the response time and transaction time of the C-Broker. Moreover, the broker with the QoS negotiation and SLA combination proved to have the smallest transaction time and response times as compared to the other combinations. This broker has the capability to enable negotiation between the service consumer and provider about some QoS constraints and it creates the SLA at the end of the negotiation. And its transaction time and response time is

not very high, so one can adopt it. Although it would not monitor for any violation of the agreed QoS guarantees since there is no QoS monitoring capability. The experimented results from the combinations of three capabilities showed that the combination of QoS measurement, QoS monitoring and QoS negotiation proved to have the highest transaction time and response time as compared to the other combinations of three capabilities. While the combination of QoS monitoring, QoS negotiation and SLA proved to have the smallest response time and transaction as compared to the other combinations of three capabilities. And the transaction time and response time is also closer to the transaction time and response time of the C-Broker in the category of three combinations. This broker has the capability to enable negotiation between the service consumer and provider about some QoS constraints, it also creates the SLA at the end of the negotiation and monitors for any QoS violation since it has the QoS monitoring capability. This broker can be the alternative to the G-Broker since its transaction time and response time is not too high as compared to the G-Broker in relation to the C-Broker.

5.7. Summary

In this chapter we presented the simulation results of our QoS-aware Web service selection broker (G-Broker). We compared our QoS-aware Web service selection broker with the classical QoS-aware Web service selection broker. Experiments on consumer satisfaction and scalability were conducted. The goal was to achieve increased consumer satisfaction on the selection of QoS-aware Web services. This goal was achieved adding QoS measurement QoS negotiation techniques, QoS monitoring and SLA. This act causes service consumers to selects and binds to Web services that will satisfy their QoS requirement. And also checks that whether the service consumers was satisfied or not by the QoS delivered. With respect to consumer satisfaction our G-Broker performs better than the C-Broker. With respect to

scalability the G-Broker is scalable and also the C-Broker is scalable.In this work we argue that the advantageof delivering service consumer satisfactory Web services that the G-Broker brings outweigh the disadvantages of high response time and transaction time. The C-Broker has good response time and transaction time compared to the G-Broker but the C-Broker does not deliver service consumer satisfactory Web services.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1. Introduction

In this dissertation, we have tried to address the challenges that exist in the current Service Oriented Architecture (SOA) environment. The SOA environment lacks the capacity to select a Web service from a group of Web services who offers similar functionality. We simulated a QoS-aware broker (G-Broker) that extends the current SOA environment to support QoS-aware Web service selection. Since the current SOA environment lacks the capability to select Web services from the group of Web services that provide similar functionality Ran(2004). A QoS is essential as a useful mechanism to differentiate those Web services which offer similar functions Vuckovic *et al*(2006) but the current SOA environment does not support QoS is because its discovery mechanism is only based on what the Web service does (functionality) without considering QoS guarantee. Weset up a framework(G-Broker)which consists of a mechanism to differentiate QoS-driven clients from non QoS-driven client. The G-Broker consists of four brokerage properties which are not commonly found in a standard middleware for SOA architectures like GUISET. These properties include QoS measurement, QoS negotiation, QoS monitoring and SLA capabilities, these properties enabled our G-Broker to provide more satisfaction to service consumers by selecting more appropriate Web services.

From the existing literature, we learnt the importance of QoS measurement, QoS monitoring, QoS negotiation and Service Level Agreement (SLA) in a QoS-aware Web Service selection environment. The *QoS measurement* avoids false QoS information claims from service

providers; *QoS negotiation* enables service consumer to negotiate with a particular service provider about the QoS constraints; *QoS monitoring* for checking any violation of agreed QoS guarantees between the service consumer and service provider; and *Service Level Agreement* (*SLA)* that serves as a contract between the service consumer and service providers about the agreed QoS guarantees. The overall goal of this research was to develop a QoS-aware Web service selection client (framework). To this end, we simulated a QoS-aware broker which enables clients to select Web services from the service registry based on QoS requirements specified by the service consumers. The QoS-aware broker is what we referred to as our GUISET QoS-aware broker (G-Broker).To ensure that our proposed QoS-aware broker is able to select appropriate Web services that would satisfy the consumer's QoS requirements, our QoS-aware broker is incorporated with all the four properties mentioned above. A simulation of two brokers was then carried out to compare the performance of a QoS-aware broker that consists of the four properties and with a broker that does not have these properties.

The remainder of this chapter is organized as follows: In Section 6.2 we highlight the achievements of this research. In section 6.3, we discuss the limitations of this work and give some suggestions on how the model can be extended in future.

6.2. Conclusion

The quality of service is a key to differentiate Web services with similar functionality. But the discovery mechanism of the current SOA architecture does not support QoS, and the service registry of the SOA architecture was not built to store QoS information of a Web Service. To have a service registry that would be able store QoS information of each Web service, Blum (2004) proposed that the categorization *tModels* in UDDI registries can be used

to provide QoS information on *bindingTemplates*.Chi-Chun *et al*(2008) stated that the approach of using *tModels* in UDDI registries store QoS information is more efficient.This approach has been adopted by many researchers e.g. Ziqiang *et al* (2007), Rajendran and Balasubramanie (2010), Diego *et al* (2006). In our research we applied the same technique proposed by Blum (2004), for storing QoS information in a service registry. This technique enables service provider to publish Web services together with their QoS information. For enabling a QoS-driven Web service selection environment we followed the most common approach of extending the current *register-find-bind* model SOA environment by introducing a middle tier component that mediates between the service consumer, the service provider, and the service registry (UDDI). This middle tier component is normally called a QoS-aware broker because its discovery mechanism is not only based on the functionality of a Web service, but it also considers the QoS requirements specified by the service consumer. The QoS-aware broker developed in this work is equipped with QoS measurement, QoS negotiation, SLA and QoS monitoring. These properties are proved to be important in a QoS-driven Web service selection environment. Adding these properties to our QoS-aware broker makes it to be unique from the other QoS-aware brokers, because from the related work that we analysed none of the existing QoS-aware brokerscombines all the four properties.

We simulated our proposed QoS-aware Web service selection broker (G-Broker) as well as the Classical QoS-aware Web service selection broker (C-Broker). Our proposed QoS-aware Web service selection broker was compared to the classical QoS-aware Web service selection broker. We used this simulation to conduct performance evaluation experiments between the two approaches. In the simulation, the following evaluation metrics were used:

1. Consumer satisfaction
2. Scalability

Analysis of the results obtained from the simulation revealed that our G-Broker brings more satisfaction to a consumer as compared to the C-Broker approach. The satisfaction in this context means that the G-Broker is able to select Web services that meet the service consumer's expectations. The results also showed that the G-Broker is scalable given the inclusion of the four component capabilities (QoS measurement, QoS negotiation, QoS monitoring and SLA). As Iqan, (2010) stated that the Service Oriented Architecture (SOA), is intended to provide scalability. Therefore, the experiments showed that the G-Broker does not bring scalability problems to the SOA as we had thought.

In essence, the evaluations concluded that G-Broker as proposed in this research is able to selects appropriate QoS-aware Web services that satisfy the service consumer's QoS requirements.

6.3. Limitation and Future Work

Although our proposed QoS-aware Web service selection broker has been proved to be an applicable approach and more effective to dynamic selection of Web service based of service consumer's QoS requirements, it has some limitations which could be recommended for future researching in this area. The G-Broker is intended to be deployed in an actual Grid based infrastructure where there are a number Web service providers, Web services and service consumers.However, owing to the infeasibility of having actual Grid infrastructure with a lot of service providers and service consumer, we simulated the environment using the three computers as described in Section 5.3.Thus makes our simulation set up to be an oversimplification of the reality that is inherently complex.

Our proposed G-Broker incorporate the four QoS capabilities (QoS measurement, QoS negotiation, QoS monitoring and SLA) and we argue that the fact that the G-Broker incorporates these capabilities makes the G-Broker to be unique, but combining these capabilities in one broker resulted in performance degradation for our G-Broker, thus might be the reason why other researchers do not incorporate all the four QoS capabilities in one broker. However; in future we would like to find appropriate approaches that wouldbring the G-Broker's performance down to a more satisfactory level.

Overall, considering that the results of this work are based on a simulation experiment which is only an approximation of the reality, therefore, another primary goal in the future work is to observe the behaviour of the proposed approach in real life environment.

BIBLIOGRAPHY

Adigun, M.O, bEmuoyibofarhe, O.J.,and cMigiro, S.O. (2006). Challenges to Access and Opportunity to use SMME enabling Technologies in Africa, a Presentation at 1st All Africa Technology Diffusion Conference, June 12-14, 2006 Johannesburg- South Africa.

Anbazhagan, M., and Arun, N. (2002). Understanding Quality of Service for Web Services", Available at: http://www-106.ibm.com/developerworks/library. January 2002. Last accessed 10 November 2010.

Austin,D., Barbir A., Newcomer E., Champion M., Ferris C., andOrchard D. (2004). Web Service Architecture Requirements, Proceeding of the W3C Working Group Notes, 2004.

Badidi,E., Esmahi,L.,and Serhani, M.A. (2005). A Queuing Model for Service Selection of Multi-classes QoS aware Web Services, Proceeding of the Third European Conference of Web Services, 2005.

Badidi, E., Esmahi,L., Serhani,M.A.,and Elkoutbi,M. (2006). WS-QoSM: A Broker-based Architecture for Web services QoS Management, 4th International Conference on Innovation in Information technology, 2006.

Benyun, S., and Kwang, M.S. (2009). Coordination and Concurrent Negotiation for Multiple Web Services Procurement, Proceeding of the International Multi Conference of Engineering and Computer Scientists, Hong Kong Vol 1 IMECS 2009, March 18-20, 2009.

Blum,A.,and Fred, C. (2004). RepresentingWeb Services Management Informationin UDDI', 2004, Available at:http://soa.sys-con.com/node/45102 . Last accessed 17 February 2011.

Clark, K.P., Warnier M.E., Brazier F.M.T., and Quillinan, T.B.. (2010). Secure Monitoring of Service Level Agreements, 2010 International Conference on Availability, Reliability and Security, 2010.

Clement, L. (2005). Risks of running SOA without registry, Available at: http://www.looselycoupled.com/opinion/2005/clem-risk-gov0228.html. Last accessed 19 July 2011.

D'Mello,D.A., Ananthanarayana, V.S.,and Santi, T. (2008). A QoS Broker Based Architecture for Dynamic Web Service Selection, Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS), 2008.

Dai, Y.S., and Wang X.L. (1999). Optimal Resource Allocation on Grid Systems for Maximizing Layered Multi-Agent System for Distributed Multimedia Applications, Proceedings of the 1999 International Conference on Parallel Processing, 1999.

Dobson,C. (2006). Quality of service in Service Orinted Architecture, Available at: http://digs.sourceforge.net/papers/qos.html. Last accessed 11 November 2009.

Eunju and ChairK and Chair J.Y.(2005). OASIS Web Services Quality Model TC, Avaliable at: http://www.oasis-open.org/committes/tc_home.php?wg_abbrev=wsqm. Last accessed 2010.

Eyhab A.M., and Qusay H.M. (2007). QoS-based Discovery and Ranking of Web Services, In IEEE 16th International Conference on Computer Communications and Networks, 2007.

Gang, Y., Chanle, W., Jun, Y., and Shi C. (2009). A QoS-aware Model for Web Service Discovery,First International Workshop on Education Technology and computer Science, 2009.

Garofalakis,J., Panagis, Y., Sakkopoulos, E.,and Tsakalidis, A. (2006). Web Service Discovery Mechanisms: Looking for a Needle in a Haystack, Journal of Web Engineering, Rinton Press, 5(3):265-290, 2006.

Gwyduk, Y., Wei-Tek, T., Xiaoying, B., and Dugki, M. (2009). Design of a Contract-Based Web Services QoS Management System,29th IEEE International Conference on Distributed Computing Systems Workshops, 2009.

Hansen, B., Elliot G., Hatmaker, G., Scott, A.,and Scott, D. (2011). Scalability Issues, Available at: http://searchdatacenter.techtarget.com/definition/scalability. Last accessed 30 May 2011.

Heiko, L., Alexander, K., Asit, D.,and Richard, F. (2010) Web Service Level Agreement (WSLA) Language Specification, Available at: www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf. Last accessed 05 March 2010.

Hofen,J.,and Schulz, (2007). Web Service Middleware- An Infrastructure For Near Future Real Life Web Service Ecosystems, IEEE International Conference on Service Oriented Computing and Application (SOCA'07), 2007.

Hung,P.C.K., Li, H.,andJeng,J.J., (2004). WS-Negotiation: An Overview of research Issues, Proceedings of the 37th Hawail International Conference on System Sciences, 2004.

Jari, K., andAparna, S. (1998). Worth-Based Multi-Category Quality –of – Service Negotiation in Distributed Object Infrastructures, Software Technology Laboratory Hewlett-Package Company, July 1998.

Johnston, P. The JISC Information Environment Service Registry, Available at: http://www.ukoln.ac.uk/cd-focus/briefings/bp3/bp3.pdf. Last accessed 19 July 2011.

Jun, Y., Ryszard, K., Jian, L., Mohan, B. C., Suk, K.G.,andJianying, Z. Autonomous service level agreement negotiation for service composition. Available at: http://www.sciencedirect.com/science/article/pii/S0167739X0700026X. Last accessed 20 May 2010.

Khader,D., Padget J., andWarnier, M. (2009). Reactive Monitoring of Service Level Agreements, In Service Level Agreements in Grids Workshop proceedings, 2009.

Lee, Y., Oh, J., andHan, S. (2005). Enriching Quality and Fault-Tolerance of Web Services System, International Journal of Web Services Practices, Vol.1, No.1-2 pp: 153-157, 2005.

Li,F., Yang,F., Shuang,K.,and SuS. (2007). Peer-to-Peer based QoS Registry Architecture for Web Services, In Proceedings of the 7th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable systems, 2007.

Liu,K.,andFeng,Y. (2007). E-Commence Oriented Automated Negotiation Based on FIPA Interction Protocol Specification, Proceedings of the Sixth International Conference on Machine Learning and Cybernatics, Hong Kong, 19-22 August 2007.

Lomuscio, A., Wooldridge,M., andJenningsN. (2001). A Classification Scheme for Negotiation in Electronic Eommerce,Lecture Notes in Computer Science Agent Mediated Electronic Commerce, The European Agent Link Perspective page 19-33, 2001.

Longworth,D. (2005). Registry holds the keys to SOA success. Available at: http://www.looselycoupled.com/stories/2005/registry-infr0131.html. Last accessed 19 July 2011.

Marco, C., andBarbara, P. (2005). An Architecture for Flexible Web Service QoS Negotiation, Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05), 2005.

Mareno, M.,andRaffeala, M. (2010). QoS Analysis for Web Service Applications: a Survey on Performance Oriented approaches from the Architecture Point of View", Technical Report UBLCS-2010-05, February 2010.

Massimo, P., Takahiro, K.,andTerry R. P., and Katia, S. (2002). Semantic Matching of Web Services Capabilities, The Semantic Web — ISWC 2002 Lecture Notes in Computer Science,Volume 2342/2002, 2002.

Menasce,D., andDubey, V. (2007). Utility Based QoS Brokering In Service Oriented Architecture, IEEE International Conference on Web Services (ICWS), 2007.

Michlmayr,A., Rosenberg F., Leitner,P.,andDustdar S. (2009). Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection, Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing,November 2009.

Netra, P., andArpita, C., (2010). Ranking Web-services Based on QoS for Best-fit Search, International Journal of Computer Science and Communication, Vol. 1, No. 2, pp. 345-349, July-December 2010.

Niko, T.,andShanika, A. (2005). Automatic Measurement of a QoS Metric for Web service Recommendation,In Proceedings of the 2005 Australian Software Engineering Conference, 2005.

Raimondi, J.S.,andWolfgang, E. (2008). Efficient Online Monitoring of Web-Service SLAs, Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering 2008.

Rajendran,T., andBalasubramanie, P. (2009). An Efficient Framework for Agent Based Quality Driven Web Service Discovery, International Conference on Intelligent Agent & Multi-Agent Systems, 2009.

Rajendran,T., andBalasubramanie, P. (2010). An Optimal Broker Based Architecture for Web Service Discovery with QoS Characteristics, International Journal of Web Services Practices, Vol.5, No.1 (2010), pp. 32-40, 2010.

Refael, K.T., andCarlos, B.W. (2006). An Architecture to provide QoS in Web Service, Proceedings of the 6th IEEE International Conference on Web services, 2006.

Serhani, M.A, Dssouli, R., Sahraauni, H., Benharref,A., andBadidi, M.E. (2005). QoS Integration in Value Added Web Services, Second International Conference on Innovations in Information technology, 2005.

Serhani,M.A., andDssouli, R. (2005). A QoS Broker Based Architecture for Efficient Web Services Selection, Proceedings of the IEEE International Conference on Web Services, 2005.

Shuping, R. (2003). A Model for Web Service Discovery with QoS, Third Annual ACM Conference on Electronic Commerce, 2003.

Skene, J., Lamanna,D.D., andEmmerich W. (2004). Precise Service Level Agreements, In Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004.

Tian, M., Gramm, A., Ritter, H., andSchiller, J. (2004). Efficient Selection and Monitoring of QoS aware Web services with the WS-QoS Framework, Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04) 2004.

Vuckovic M., Nevera, S., and Stefanovic, T.S. (2004). Quality of Experience of mobile service, Nokia code: 11212 – 1004 Indivisual/Libris, 2004.

OASIS (2003). UDDI Specification TC, Avaliable at: http://www.oasis-open.org/committes/uddi-spec/doc/tcspecs.html. Last accessed 2010.

Wang,S.J., andChen,H. (2008). A Web Service Selection Model Based on Measurable QoS Attributes of Clients Sides International Conference on Computer Science and Software Engineering, 2008.

Yong, W., Li., andGuiping, D.(2008). QGWEngine: A QoS-aware Grid Workflow Engine, IEEE Congress on Services, 2008.

Zhou, C., Chain, andL.T., Bilhanan, S., Lee B.S. (2003). UX-An architecture Providing QoS-aware and Federated Support for UDDI, In proceedings of the first International Conference on Web Services (ICWS03), 2003.

Zhou,C., Chia,L.T., and Lee,B.S. (2005). Semantics in Service Discovery and QoS Management, IT Professional archive Volume 7, Issue 2, March 2005.

Ziqiang X., Martin, P., Powley, W., and Zulkernine, F. (2007). Reputation Enhanced QoS-based Web Service Discovery, Proceeding of the IEEE International Conference on Web Services ICWS 2007 (2007)