

1-1-2010

QoS-based semantic web service selection

Yijun Chen
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Chen, Yijun, "QoS-based semantic web service selection" (2010). *Theses and dissertations*. Paper 996.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

QOS-BASED SEMANTIC WEB SERVICE SELECTION

by

Yijun Chen

B. Sc. of Computer Science, Ryerson University, 2006

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the Degree of

1Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2010

@Yijun Chen 2010

DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Yijun Chen

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in parts, at the request of other institutions or individuals for the purpose of scholarly research.

Yijun Chen

QOS-BASED SEMANTIC WEB SERVICE SELECTION
M.Sc. Computer Science, 2010

Yijun Chen

Department of Computer Science

Ryerson University

ABSTRACT

This thesis discusses the dynamic web service selection in the semantic context with QoS constraints. The goal of this work is to investigate the mechanism of automated QoS-based semantic web service selection.

Semantic Web Service (SWS) aims to achieve the automation of web service tasks, such as service discovery, selection, composition and invocation. The task of semantic web service selection is further investigated through this thesis.

An architecture is proposed to achieve this task by considering QoS parameters. The QoS parameters are classified into dynamic and static attributes in the architecture. The dynamic attributes are evaluated and measured as an overall value by applying utility functions. This overall value can be modeled in the semantic context for the purpose of service selection.

Furthermore, the architecture directly models the static QoS attributes in the semantic context for service selection. Finally, an open SWS challenge scenario named hardware purchasing is used in several experiments in order to evaluate the proposed architecture.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Dr. Abdolreza Abhari for his supervision and support during my research work. Dr. Abhari not only transferred the knowledge to me, but also guided me towards the right way to do the research work. I do appreciate Dr. Abhari's assistance and his hard work.

I also would like to thank Dr. Alexander Ferworn. Dr. Ferworn taught me the research method, and guided me to choose the proper research topic.

Finally, I would like to thank my family, especially my wife Julie Ni, for their unconditional support and encouragement.

TABLE OF CONTENTS

Chapter 1

1.	Introduction.....	1
1.1		
	Motivation.....	2
1.2	Problem Statement.....	4
1.3	Scope, Assumption.....	6
1.4	Contributions.....	7
1.5	Thesis Structure.....	7

Chapter 2

2.	Background and Related Work.....	9
2.1	Background Information.....	9
	2.1.1 Limitation of the Current Web Service Technology.....	10
	2.1.2 Semantic Web Service Challenge.....	11

2.1.3	Introduction to WSMO.....	12
2.1.3.1	WSMO Ontology.....	12
2.1.3.2	WSMO Web Service.....	13
2.1.3.3	WSMO Goal.....	14
2.1.3.4	Service Grounding.....	14
2.1.3.5	Logic Reasoning.....	14
2.1.3.6	WSMO Service Discovery Mechanism.....	15
2.2	Related Work.....	16
2.2.1	Web Service Selection.....	16
2.2.2	QoS-based Web Service Selection.....	17
2.2.3	Modeling QoS in WSMO.....	19
2.2.4	Comparison between WSMO and other SWS Frameworks	20
2.3		
	Summary.....	23

Chapter 3

3.	Utility-based QoS Brokering Model for Semantic Web Service Selection.....	24
----	---	----

3.1	Evaluate QoS Information with A Utility Function.....	25
3.1.1	Utility Function Regarding to Response Time.....	27
3.1.2	Utility Function Regarding to Throughput.....	28
3.1.3	The Overall Utility Function.....	30
3.2	Utility-Based QoS Brokering Architecture for Semantic Web Service Selection	31
3.3	Overview of the SWS Challenge Scenario - Hardware Purchasing.....	33
3.4	Model QoS Constraints in the WSMO Semantic Context	34
3.5.	Modeling Static QoS Parameters in the Proposed Architectur.....	41
3.6	Summary.....	42

Chapter 4

4.	Evaluation of the Model.....	43
4.1	Experiment Environment.....	43
4.1.1	Evaluation Objective.....	44
4.1.2	Experiment Components.....	44
4.2	Evaluating Utility Functions.....	46
4.3	Evaluating the QoS Broker	49
4.3.1	Evaluating QoS Collector	49
4.3.2	Evaluating Utility Analyzer	50

4.4	Evaluation of Utility-Based QoS Broker in WSMO Semantic Service Selection.	51
4.4.1	Experiment Environment	51
4.4.2	Simulation 1: Service Selection Algorithm with a Base QoS Utility Value.....	53
4.4.3	Service Selection Algorithm with the Criteria for the Service with Highest QoS Value Selected.....	55
4.5		
	Summary.....	60

Chapter 5

5.	Conclusion and Future Work.....	61
5.1	Conclusion.....	61
5.2	Future Works.....	63
	Appendix.....	65
	References.....	76

LIST OF TABLES

Table 2.1	Synonyms comparison.....	15
Table 2.2	Example list of QoS attributes from [23].....	18
Table 3.1	Example of QoS information	29
Table 3.2	Examples of the products offered by the service vendors.....	35
Table 4.1	Result of Service Selection for Test Case one	54
Table 4.2	Result of Service Selection for Test Case Two.....	55
Table 4.3	Result of Service Selection for Test Case Three	56
Table 4.4	Service Selection Result for Three Request Sequence	59
Table A.1	Product Information Provided by All Vendors.....	65
Table B.1	Experiment Result: the QoS information for Service Bargainer.....	69
Table B.2	Experiment Result: the QoS information for Service Hawker	69
Table B.3	Experiment Result: the QoS information for Service Rummage.....	70
Table C.1	Utility for Service Bargainer.....	72
Table C.2	Utility for Service Hawker.....	72
Table C.3	Utility for Service Rummage	72

LIST OF FIGURES

Figure 1.1	Dynamic Service Selection/Invocation Scenario	3
Figure 2.1	Ontology toward semantic web and semantic web service.....	21
Figure 2.2	RDF, OWL toward WSMO semantic.....	22
Figure 3.1	Pseudo code Web Service Selection by Evaluating the QoS Information	25
Figure 3.2	Algorithm to Retrieve Service Response Time	28
Figure 3.3	Algorithm to retrieve service throughput.....	30
Figure 3.4	The QoS broker merged with WSMO/WSMX architecture.....	31
Figure 3.5	The general algorithm for functional discovery.....	36
Figure 3.6	A new concept, QoSValue	36
Figure 3.7	An QoSValue instance for service Bargainer.....	37
Figure 3.8	An QoSValue instance for service Hawker.....	37
Figure 3.9	An QoSValue instance for service Rummage.....	37
Figure 3.10	Description of the Goal for Notebook purchasing	38
Figure 3.11	Capability for the Goal of Notebook Purchasing with QoS Selection Criteria.....	39
Figure 3.12	Capability for the Goal of Notebook Purchasing with QoS Selection Criteria	40
Figure 3.13	Definition For rankingcriteria Variable	41
Figure 3.14	QoSPrice Concept and its instance for Service Bargainer	41
Figure 4.1	Utility Function Vs Response Time for Service Bargainer.....	47
Figure 4.2	Utility Function Vs Response Time for Service Rummage.....	47
Figure 4.3	Utility Function Vs Response Time for Service Hawker.....	48
Figure 4.4	Utility Function Vs Throughput for Service Bargainer.....	48

Figure 4.5	Utility Function Vs Throughput for Service Rummage.....	49
Figure 4.6	Utility Function Vs Throughput for Service Hawker.....	49
Figure 4.7	Experimental Execution Flow	53
Figure 4.8	Service Selection Sequence A in WSMO Environment.....	57
Figure 4.9	Service Selection Sequence B in WSMO Environment.....	58
Figure 4.10	Service Selection Sequence C in WSMO Environment	58
Figure 4.11	Service Selection Sequence in Semantic Environment ($\sigma=0.005$).....	59
Figure 4.12	Service Selection Sequence in Semantic Environment ($\sigma=0.01$)	60
Figure B.1	Implemented QoS response-time collector in Java.....	68
Figure C.1	Implementation of The utility calculator (service Bargainer) in Java	71
Figure D.1	Capability Definition for Notebook Purchasing	74
Figure D.2	Service Selection result Shown in WSMX Message Window.....	75

LIST OF ACRONYMS

ACRONYMS

DEFINITIONS

WSDL	Web Service Description Language
SOAP	Simple Object Assess Protocol
UDDI	Universal Description, Discovery and Integration
SOA	Service Oriented Architecture
UDDI	Universal Discovery, Description and Integration
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Execution Environment
XML	Extensible Markup Language
OWL	Web Ontology Language
RDF	Resource Description Framework
GUI	Graphical User Interface
API	Application Programming Interface

CHAPTER 1

INTRODUCTION

In the last decade, web service has been the most successful and popular technology in distributed computing [1][2]. Web service is a distributed technology that communicates based on HyperText Transfer Protocol (HTTP). It is described using Extensible Markup Language (XML), and transported via Simple Object Access Protocol (SOAP). Due to these characteristics, web service has been widely adopted and applied in the IT industry.

Consequently, more and more web services have been created and published in order to achieve users' requirements. For example, Google provides its web service APIs to allow users to access its resources. Amazon has its own service APIs for users to search its online bookstore. Many E-commerce service APIs are published to achieve various functionalities. However, developers and researchers have realized the difficulty of discovering and reusing certain web services to fulfill their tasks in such a large scale publication of web services.

Furthermore, current web service techniques allow service consumers to discover and bind services only at design time. Dynamic service selection during run time is still challenging [3]. This thesis discusses this challenge and proposes a potential solution based on the semantic context. In the first chapter, background information of the thesis is presented and organized as follows. Section 1.1 introduces the motivation of this work. Section 1.2 discussed the problem

statement. Section 1.3 defines the assumption and scope. Section 1.4 states the contributions of this work. And finally Section 1.5 presents the thesis structure.

1.1. MOTIVATION

With the boom of web service technologies, web services have been created and published in a large-scale manor. However, this large amount of web services produces several unpredictable issues. Many services provide similar functionalities. When developers and researchers try to find specific services to fulfill their tasks, they could be faced a long list of web services in the search result. It is difficult for the current web service technologies to find a suitable web service in a list of similar web services. One of the main reasons is that current web technologies are built upon syntactical techniques, and syntactic-based search mechanisms cannot differentiate web services and the related QoS information.

Furthermore, in the current service oriented architecture (SOA), many tasks are still executed manually [4]. For example, in the Java language, if a developer wants to invoke a web service, he/she is required to know the service address in the design stage. He/she needs to initially create a web service reference via a Web Service Description Language (WSDL) file, define an instance of this web reference, and invoke the web service functionalities.

However, in many cases, web services need to be invoked on the fly. Figure 1.1 shows the scenario of web service selection and invocation at the run time. At the beginning, the developer does not know which service is selected, as services are determined by the running result of the

previous service in the decision box. As we stated earlier, if the developer wishes to select and invoke web services, he/she first has to create the corresponding web service references manually. Even if the numbers of web services are in the hundreds, there are still countless manual works to complete.

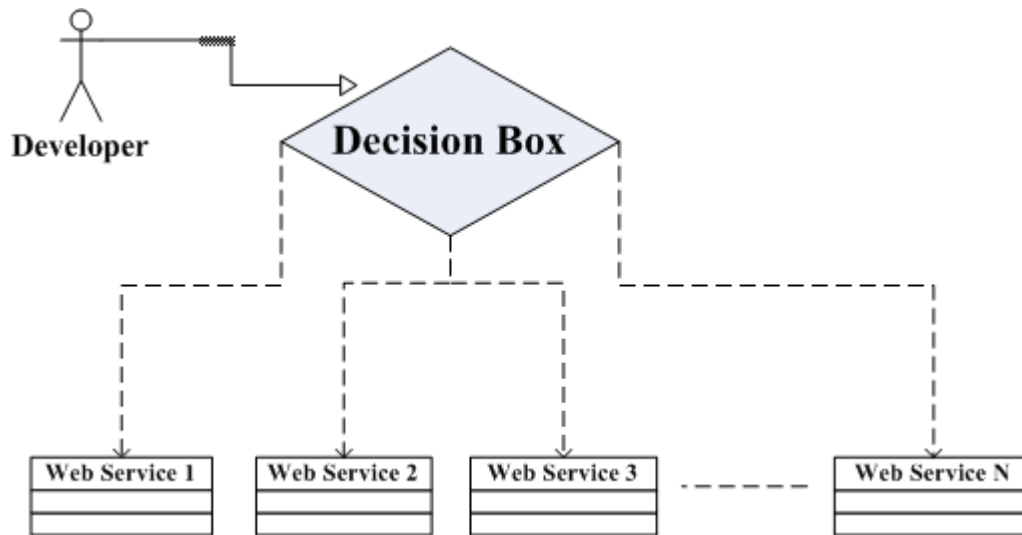


Figure 1.1 Dynamic Service Selection/Invocation Scenario

Evidently, it is difficult to implement such scenarios in the current SOA framework. Thus innovative modes must be structured in order to avoid such manual works. These limitations in the current web service technologies propel many researchers to find their solutions. In this work, we propose a model to retrieve web services QoS information and later, based on that information web services are automatically selected on the running time. This concept of automatic selection -based on the QoS information - is regarded as the main source of motivation for this thesis.

1.2. PROBLEM STATEMENT

As we discussed earlier, the current web service technologies are insufficient in supporting dynamic service selection and can not maximize the automation tasks. To achieve the automation of web service tasks, one of the major approaches is to make web services become more meaningful, namely by adding semantics to web services.

As the concept of Semantic Web Services (SWS) blooms, more attention is given to its related research topics. Bachlechner in [5] discussed the technology roadmap toward SWS. The concept of SWS is regarded as the next generation of web technology. The main goal of SWS is to enhance the automation level of web services [4]. More specifically, the SWS aims to achieve automation of service discovery, selection, composition and invocation. Many researchers endeavour the study of how to implement semantic tasks to web services, and some related solutions have been addressed including [6] and [7].

This is the major problem addressed in this thesis. To achieve dynamic service selection in the semantic context, we need to find a way to achieve semantic-based automated tasks. Semantic-based automated tasks involve the following issues:

- Add semantics to web services.
- Describe the consumer's request in semantic language.
- Implement the logic, which makes the above two items understand each other.
- Execute the semantic solution and retrieve the correct results.

Furthermore, we introduce the Quality of Service (QoS) constraints in our semantic solution. The QoS information is an important factor in dynamic service selection mechanism that is considered in [8] and [9]. Applying the QoS in service selection mechanism can greatly enhance the accuracy of the service search results. At the early stage of web service technologies, most researchers are concerned with the functionalities and interfaces of web services as mentioned in [10]. However, with the maturity of web service technologies, along with an increase in provided services, consumers are more likely to choose the better quality services.

Another problem is how to deal with QoS information in our semantic solution. The QoS may be impacted by many factors, including network performance, the service hosting architecture, and service performance as it was used in [10]. For example, the service availability, response time, service charge, and service reputation are all samples of QoS parameters. So the question is: how can we apply these QoS attributes to the dynamic service selection algorithm in the semantic context? This problem can be separated into the following items:

- How to retrieve and evaluate these QoS attributes.
- How to model these attributes in semantic.
- How the modeled QoS information can impact the service selection.

The above issues have been hot research topics in recent years. Many researchers investigate and endeavour to find their own solutions. It is the objective and core task of this work to find a solution for the above three questions.

1.3. SCOPE AND ASSUMPTIONS

The Semantic Web Service (SWS) is designed to minimize human intervention as much as possible. The objective is not only to achieve all core tasks related to the web services such as service discovery, selection, composition and execution, but also to support the automation of these tasks. However, it is difficult to cover all the issues regarding the SWS in this work. In the following section, we would like to introduce our research scope, assumptions of this work.

Currently, there are several approaches proposed to provide semantic support to the web services as explained in [11], for example, Resource Description Framework (RDF) [12] and [13], Web Ontology Language for Services (OWL-S) [14] and [15], and Web Service Modeling Ontology (WSMO) [17]. OWL-S and WSMO are the most prominent frameworks. Efforts in the SWS research focus mostly on the topic surrounding these two approaches as stated in [18]. In this case, this scope is defined in the area of WSMO framework in order to achieve the semantic task.

Furthermore, we limit our solutions to the semantic tasks in the area of service selection. Thus, other tasks such as service discovery, composition and invocation are not included. For the purpose of evaluating our proposed solution, we make use of one of the Semantic Web Service (SWS) Challenge scenarios available for all researchers named hardware purchasing [20], as our experimental test case. This scenario is used to provide a QoS-base semantic solution for dynamic web service selection. The scenario provides three web services for users to test their semantic solutions for hardware purchasing problem.

The main objective of this research is to provide a solution to achieve the task of dynamic service selection in the semantic context based on the QoS information. To achieve this goal, there are two basic assumptions in our approach. First, we assume the static QoS information provided by service providers that are retrieved in our proposed architecture is trustworthy. Second, we assume the services that are provided by the scenario of hardware purchasing test case are always available.

1.4. CONTRIBUTIONS

In this thesis, we investigate the QoS-based semantic solution for dynamic service selection. A simple architecture is proposed to achieve this task. Also, our experiment demonstrates this proposed architecture to fulfill the semantic tasks. The contribution of our research can be stated as following:

- Design a QoS brokering architecture to collect the QoS information.
- Evaluate the QoS information by applying utility functions.
- Model the QoS information in our semantic solution.
- Implement the service selection mechanism based on the QoS constraints for WSMO environment.

1.5. THESIS STRUCTURE

For the thesis structure, the rest of the parts are organized as follows. In Chapter 2, background and related works will be discussed. In the background information, certain limitation of the

current web service technology and some core concepts in the WSMO framework are introduced. Subsequently, other related works are also introduced. Related works including QoS-based web service selection, semantic web services and the challenges, and some related solutions provided by WSMO for semantic web services will be explained next. In Chapter 3, an architecture and semantic-based model is proposed, while in Chapter 4, this architecture is evaluated based on the test case of hardware purchasing. In Chapter 5, this thesis is concluded and our future work is introduced.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter will discuss background information and related works. In the background information, the limitations in the current web service technology will be presented, as well as several basic concepts in WSMO will be introduced. In the related works, various early works in the service selection will be reviewed first. Then, the related works in semantic context will be introduced.

2.1 BACKGROUND INFORMATION

With the upgrade of network technology and more complicated distributing environment, the current web service technology can not meet the users' requirement in some areas. There are several factors that play the roles in causing the limitation of the current technology. Consequently, the Semantic Web Service (SWS) has been proposed to solve these shortages. In this section, we present some background information on this work. We first discuss the limitation of the current web service technology. Then we introduce one of the semantic solutions, WSMO, and discuss its basic concepts and principles.

2.1.1. Limitation of the Current Web Service Technology

Web service is supposed to be the outstanding middleware solution for enabling the development of distributed software applications. Web service builds upon some standard technologies. Service interfaces and data are described in Web Services Description Language (WSDL) [19]. The messages delivered between service providers and consumers meet the standard protocol, Simple Object Access Protocol (SOAP) [21]. The communication is via the Hypertext Transfer Protocol (HTTP). Also, the Universal Description Discovery and Integration (UDDI) [22] provide a platform, which web services can register and consumers can find services by their registry information.

By means of the above outstanding characteristics, web service is becoming very popular in various distributed applications. One of the results is that web services are created and published rapidly in a very large scale basis. As we discussed earlier, the huge number of web services cause problems. For example, consumers could not find the suitable web service meeting their requirement; also, if the interface of a web service needs to change due to any new requirements, the consumer could not be notified immediately, thus causing the failure when invoking the web service.

There are some reasons responsible for the above problems. The major one is that web service has an inherent drawback. Current technology, such as UDDI (offering the service for web service registry and discovery) and WSDL (offering the interface for service consumers

accessing the web service), only provide syntactical support. Syntactical support only contains limited information so that service consumers can't find the exact suitable services based on their requirement.

According to Cooney et al. [23], web service was designed in the static mode and the service resided at the network endpoint for a long run at the beginning. When the service interface is modified with updated business logics, the web service endpoint has to be renewed manually. Consumers cannot be automatically notified with the modification, and they have to manually update the service reference which points to the web service endpoint. If consumers failed to update the reference, it could cause the invocation failure.

2.1.2. Semantic Web Service Challenge

The Semantic Web Service Challenge [20] was initially launched by Stanford University (USA) in the year 2006. It provides a series of workshop, problem scenarios, and testbeds for participants to do research on the topic of semantic web services. The problem scenarios are the open problems, which serve as the basis for the examination and comparison of approaches addressing the challenges [24].

Margaria in [25] introduces complexity in the domain of the SWS challenge. The authors analyze the major problem in semantic web service in which there is no common standard to compare and classify these frameworks in terms of their abilities and shortcomings. In [25], an example is given to describe two problem scenarios: mediation and discovery.

2.1.3. Introduction to WSMO

In this section, the brief explanation of WSMO is provided. We use the test case scenario of this thesis as the example to make the definition more clear. WSMO consists of four main elements: ontology, web service, goal, and mediator. These elements constitute the WSMO solution for achieving semantic web service tasks. Ontology can be considered as the basic building block. It extracts the data and models from the service providers and formulates the semantic context. Goal and web service are based on the ontology to add the semantic. However, for the challenge scenario of hardware purchasing, we don't involve the mediator since all the service providers conform to the same message format and there is no need for the data or message format transformation by use of mediator. The following briefly discusses three elements, including a simple introduction of some core concepts in the semantic technology.

2.1.3.1. WSMO Ontology

Originating from a philosophical terminology, ontology in computer science means a specification of a shared conceptualization [26]. In the semantic web, ontology acts as a central enabling technology. It provides knowledge representation and allows machine understanding of the knowledge via the links including the terms in the ontologies.

Consequently, ontology is also the core element in WSMO. First of all, ontology in WSMO can be regarded as data models and instances, which is responsible for all the resource description

and data exchange between the WSMO elements. Secondly, ontology describes the relationship between these data models and functions involved in the domain. Thirdly, the ontology use a mediator to communicate with other domains through providing data mapping between domains. Furthermore, ontology is organized hierarchically. Similar to the traditional class definition in object-oriented design, an ontology can inherit and/or be inherited by other ontologies.

2.1.3.2. WSMO Web Service

The work [27] explained WSMO web service as a computational entity to describe all aspects of a service, mainly including non-functional properties, functionalities, and interfaces. By invoking this web service, users can achieve the goal they predefined. In the test case that used in this work, there are three traditional web services provided by three different product vendors for the hardware purchasing scenario. These three web services comprise of Bargainer, Rummage, and Hawker. The corresponding WSMO web services need to be developed on the top of three original web services.

There are two main elements to fulfill WSMO web services functionalities, capability and interface. Capability describes the web service according to its functionality, and defines the pre/post condition or status of the functionality. Interface provides the real implementation of the web service functionality. It is mainly achieved by two operations: choreography and orchestration. Choreography provides communication between the web service and the client, while orchestration interacts with other web services for the purpose of achieving capability.

Also, web service lists all the product instances enabling the request to find all those that matched the given requirement. The instances can be stored in another independent ontology

2.1.3.3. WSMO Goal

The WSMO goal is derived from the consumer's requirement. For the case of hardware purchasing scenario, it describes the product the consumer wants to purchase. The general rule to define this product is that it should be offered by all service providers. Thus, the QoS constraints can be tested on the service selection.

2.1.3.4. Service Grounding

.

Service grounding acts as glue between WSMO web service and the original web service introduced in [29]. Data models and functionalities are described by WSDL and the input/output message formats are defined in the XML schemas, which are regarded as syntactic description of web services. However, WSMO is a semantic solution of web service. The semantic description is implemented by means of ontology. Ontology consists of the elements like properties, concepts, and so on. These elements are evolved from WSDL and its XML schema when designing the ontology. Kopeck in [29] explained that the procedure of mapping the semantic description to the syntactic elements is defined as service grounding.

2.1.3.5. Logic Reasoning

The logical reasoning behind the WSMO is based on the description logic. More accurately, the description logic for WSMO is a subset of First-Order logic (FOL) as explained in [30] and [31]. Description logic is a family of decidable ontology language, facilitating provision of reliable and consistent reasoning. Hodges in [31] introduced the concept of FOL and syntax as well as its applications. Similar to the description logic in OWL, WSMO has its own DL – WSML-Description Logic (WSML-DL). Currently, the WSML-DL logical expression is restricted to the style of FOL. The paper [32] gave distinct principles and examples of FOL. Also, for the purpose of a more simplified understanding of the basic elements of description logic, we conclude a synonym comparison chart for clarification (Table 2.1). In comparison with the traditional programming language, the concept can be mapped into class; meanwhile the role can be mapped into property and the individual can be mapped into object.

Table 2.1 Synonyms comparison

Traditional Language (Java)	Description Language	WSML DL
class	concept	concept
property	role	relation
object	Individual	instance

2.1.3.6. WSMO Service Discovery Mechanism

WSMO presented some distinguished characteristics in the service discovery mechanism, which is further explained in [33]. It has multiple discovery engines which can support different search

levels. WSMO supports keyword discovery, functional discovery, and instance based discovery, as this is also further explained in [34]. Keyword discovery, also referred to as non-functional properties discovery, performs the search by the non-functional properties defined in the service description. Toma in [16] presents how to model QoS characteristics as non-functional properties in WSMO. Shafiq et al. describe in [33] that functional discovery plays a key role in WSMO service discovery and selection since it is based on the descriptive logic. Finally, Shafiq et al. conclude that different search engines can be organically applied to this complicated goal discovery.

2.2. RELATED WORK

The SWS technology is regarded as the next generation of web technology. The topic of semantic technology achieving dynamic web service selection attracts many researchers and organizations. In this section, initially the similar work on current web service selection methods are examined in close detail, along with the subsequent explanation of the research status and related works in the semantic web service selection.

2.2.1. Web Service Selection

Web service selection in SOA environment has been a hot research topic in recent years. A few known frameworks, such as workflow (BPEL4WS) [35], dataflow [36], and process-based [37] have been implemented in this area.

However, the above methods have their limitations, which cannot meet the complexity in dynamic service selection. Web service selection is one of the steps in the service composition. Guilan et al. [35] discuss a workflow model applying service composition by using the Business Process Language for Web Services (BPEL4WS). Although the framework proposed in [35] can achieve the automatic task to some extent, it is not designed for task automation and cannot really solve the problem.

Furthermore, users cannot select service on demand since the flow or processes define service selection in advance. This point also has been reported in [35] and [37]. Consequently, service discovery and selection is static and inflexible. Moreover, a service selection method proposed by Tsai et al. in [38] cannot promise to find suitable services in order to achieve the user's request. These drawbacks promote several new mechanisms for dynamic service selection mechanisms, of which the SWS is the most prominent one.

2.2.2. QoS-based Web Service Selection

The QoS information describes the quality of web services. It is an important consideration when the consumer makes decision on service selection. Normally, the QoS attributes can be classified in two categories: dynamic and static - as described in [28]. Li et al. explain in [28] that dynamic attributes could be changed in the execution time, for example response time and throughput; static attributes are defined by service providers before service executions and are usually not updated during the execution. Table 2.2 presents some example attributes by this classification.

These static QoS attributes are relatively easy to process in the context of service selection; while the dynamic ones are more complicated.

Table 2.2 Example list of QoS attributes from [28]

	QoS Attributes
Dynamic	Availability, response time, throughput, reputation, etc.
Static	Stability, capacity, accuracy, security, price, etc.

The QoS-based web service selection has been discussed for a long period of time. There are many approaches proposed in order to achieve this target. Yu et al. in [39] introduces the QoS brokering architecture which manages the QoS constraints in distributed services. The system provides the main functions such as service discovery, planning, and service selection.

The work in [9] also discusses the QoS brokering system. The author introduces utility functions to evaluate the QoS information. The broker is designed to fulfill the tasks, such as collecting QoS information, applying the utility functions to the QoS information, and selecting the suitable services. This work presents the preliminary idea to introduce such brokering systems into the semantic environment.

Currently, there are some measuring techniques for evaluating the QoS information. One solution is the QoS matrix as explained by Li et al. in [28]. In this method, a QoS matrix is

formulated by applying the collected QoS values, and the overall value is measured through the matrix calculation. Another solution is utility functions introduced by Menasce in [9]. It also first collects all the QoS information, retrieves the utility value for each QoS parameter, and merges all the independent values into an overall one.

Utility functions have been applied in the distributed autonomic computing system for a period of time (see [43] for details). Utility function can achieve self-optimization, which is the most attractive character. There are a couple of ways to get utility functions. The function could be designed by an expert, specified in an agreement, or derived from another utility function as described by Walsh et al. in [43]. The utility functions we used in this thesis are similar to those presented in [9].

2.2.3. Modelling QoS in WSMO

Toma et al. in [16] introduced some basic steps of how to model QoS constraints in the WSMO environment. They acknowledge that there are two main challenges: the first is concern with how to model QoS information, and the second is about how to attach the QoS attributes to WSMO services and goals. The authors also discuss other possible approaches. Essentially, [16] suggests model QoS attributes as non-functional properties, and implements the attaching logic by using some WSMO elements, such as relation, concept and capability. The author doesn't implement any of these approaches.

The topic presented in article [16] is very similar to the topic of this thesis. The main difference rests in the fact that we classify the QoS information into dynamic and static attributes in this work. Dynamic attributes are evaluated measured as an overall value by applying utility functions. The overall value will be modeled in the semantic context for the purpose of service selection. Also the static QoS attributes will be directly modeled in the semantic context for service selection. Since there are many research works in the area of Semantic Web Service based on different SWS frameworks, in the next section, we discuss them and explain the differences among three frameworks in details.

2.2.4. Comparison between WSMO and other SWS Frameworks

The SWS research and development have received much attention in the past few years. Many researchers endeavour the study of how to implement the semantic tasks to web services. Bachlechner in [5] introduced the technology roadmap toward to SWS, and he also acknowledged that the SWS had not been adopted by industrial segments at the time he addressed this paper in the year 2008.

Currently, several frameworks have been developed to help provide the environment of building semantic projects. As mentioned in Chapter 1 of this thesis, Resource Description Framework (RDF) introduced in [12] and [13], Web Ontology Language for Services (OWL-S) introduced in [14] and [15], and Web Service Modeling Ontology (WSMO) [17] are most outstanding frameworks. These concepts will be explained in further detail in the following paragraphs.

RDF is W3C standard metadata data model. The most distinct feature of RDF is considering the user independence. With the updating of metadata schemas in the server side, the clients do not need to change the data usage and thus, will be kept unaffected [40]. This mechanism achieves the independent implementation between the data providers and consumers. Also, RDF provides a graphic view in the metadata model design. The metadata resource is represented as a graph node. The two nodes can be linked by an edge which describes the existing relationship between them. This feature provides users a visual explanation of the model design and can be easily understood. The above characteristics give RDF its own set of unique advantages in the semantic domain.

OWL Web Ontology Language is also a W3C standard language for the purpose of constructing web ontologies [15]. Web ontology describes the entities in the web along with the relationship among these entities. The entities contain the concepts and relations extracted from the web, covering functionalities, along with content the web provides. Ontology could contain the definition of properties, functionalities, classes, and instances transformed from the entities.

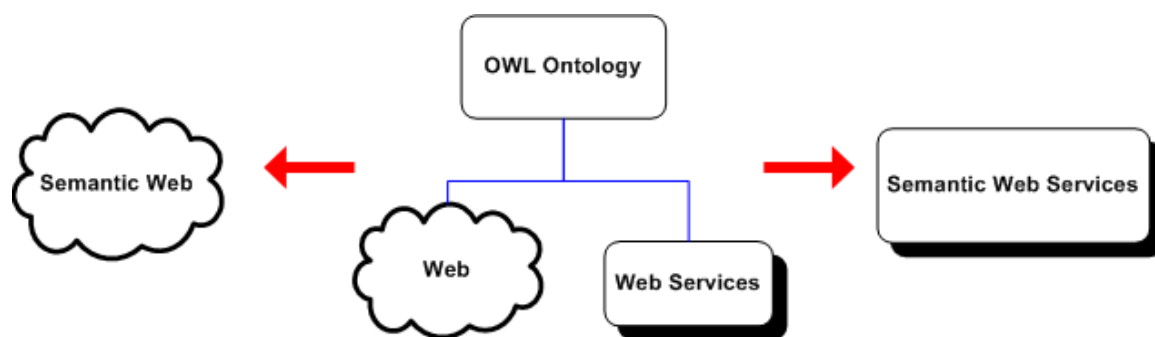


Figure 2.1 Ontology toward semantic web and semantic web service

The above figure shows the OWL ontology along with its relation to the web and web service, which uses the semantic context and produces semantic web services.

The main part of the OWL ontology is Description Logic (DL) as explained by Baader et al. in [41]. Description Logic is a knowledge representation language. It's normally applied in the domain of knowledge representation, which can add logical reasoning on the concept of the knowledge base domain. Thus, domain elements can be understood, reasoned, and processed in the computing environment. Consequently, this logic reasoning ability of DL is also applied for OWL ontology and semantic web service by providing logical formalism.

However, by comparing the pros and cons of these frameworks, we decided to choose WSMO to implement the proposed solution. WSMO is a combination of OWL and RDF. It takes the metadata model design from RDF and ontology design based on description logic from OWL. This combination makes WSMO to be the most prominent approach in the domain of semantic web service. Figure 2.2 shows what WSMO has received from other semantic technologies.

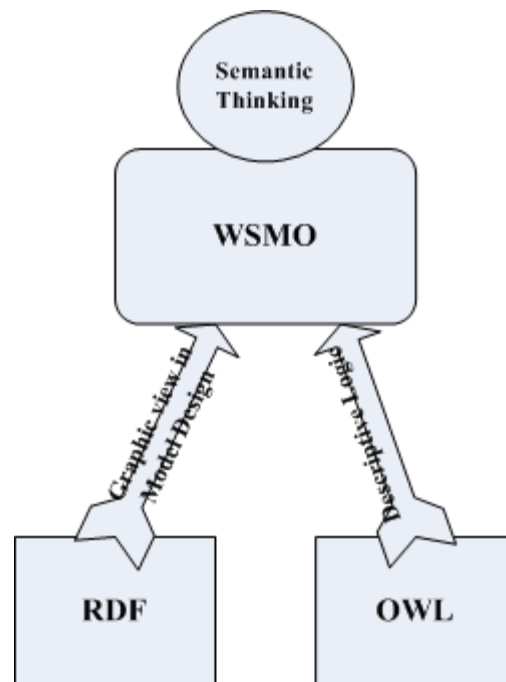


Figure 2.2 RDF, OWL toward WSMO semantic

Also, another advantage of WSMO is that it not only provides a framework, but that is also provides an execution environment named as WSMX [42]. This environment allows for the execution and monitoring of semantic applications.

2.3. SUMMARY

In this chapter, we discussed the background information and related works of this thesis. The background part discussed the limitations in the current web service technology. Then, it introduced the concept of semantic web service along with its challenges. Also, it introduced the ontology-based semantic web service solution (i.e., WSMO), its formalism language (i.e., WSML), and the execution environment (i.e., WSMX). The related works reviewed various early works in the service selection. In addition, it introduced some works in QoS-based web service selection. It discussed WSMO semantic framework and present how to model QoS information in this framework. Finally, it compared WSMO with other semantic frameworks.

CHAPTER 3

UTILITY-BASED QOS BROKERING MODEL FOR SEMANTIC WEB SERVICE SELECTION

In traditional service oriented architecture, the Quality of Service (QoS) attributes play an important role in dynamic web service selection. Currently, there are many solutions to address how the QoS information can be retrieved and applied into the algorithm of service selection. In this section, a utility-based QoS brokering architecture will be proposed for dynamic web service selection. This brokering system will then be further investigated for semantic environment. The first aspect under examination would be how utility functions can evaluate the QoS information, followed by an introduction to the brokering system and how the QoS information is conveyed. Finally, the main part of this work is concentrated on the implementation of semantic web service selection based on the QoS information.

The remaining part is organized as following: utility functions will be discussed with a focus on evaluating QoS attributes in section 3.1. A utility-based QoS brokering architecture will be presented in section 3.2, and finally, we focus on how to implement this QoS broker in the semantic context in section 3.3.

3.1. EVALUATING QoS INFORMATION WITH A UTILITY FUNCTION

In this section, the emphasis relies heavily on the dynamic QoS attributes. The following section presents how we evaluate the QoS attributes on service selection. The general idea to measure the QoS information is to collect all QoS values for each service, merge them into an overall value, and then compare the overall value to see which provided service meet the consumer's requirement. The algorithm is described as following:

```
/*Algorithm: web service selection based on the QoS information */  
  
function FindServices()  
  
  initialize all services in the service repository  
  
  for each service  
    for each involved QoS attribute  
      retrieve its QoS value and apply its utility function  
      calculate the utility value  
    end For loop  
  
  calculate the overall QoS value by overall utility function  
  
  if the overall value meets the consumer's requirement  
    add the service to the selected list  
  
  end for loop  
  
  return the selected service list
```

Figure 3.1 Pseudo code of the Web Service Selection by Evaluating QoS Information

From the algorithm, we can see that the measuring technique is the key point. Usually these QoS attributes are measured by using utility functions. Utility functions can reflect the real status of the QoS information. Each QoS attribute is supposed to be measured by a function. The utility function finally produces a utility value, which can be used as the overall values of the total QoS attribute. The relationship between the QoS attribute and its function is either monotonically increasing or decreasing. For example, the utility value is decreasing while the response time increases, or the value is increasing when the throughput increases. In this thesis, we take these two dynamic QoS parameters as examples to help validate the utility function.

Also, these two independent utility values will be later applied in the overall utility function with their own weight for their overall value. The overall value is finally used for service selection. The reason why we need this overall value is that the service selection cannot be based on each single QoS attribute. For example, from the services presented in Table 3.1, it is hard for the consumer to make the decision on which service is the best one. Also, if there are more attributes involved, the situation becomes more complicated. Thus, the overall QoS value is introduced to evaluate all the involved QoS information and produce the total value.

Table 3.1 Example of QoS information

Service	Response Time	Price	Reputation
Service A	Fast	Expensive	good
Service B	Medium	Not Cheap	good
Service C	Slow	Cheap	Very good

Moreover, this overall utility function can be easily extended by adding with other QoS parameters. Each QoS attribute has its weight contributed to the overall value. These issues will be discussed in further detail. In this thesis, we'll investigate how utility function can be measured the QoS information and applied in the semantic context.

3.1.1. Utility Function Regarding to Response Time

As we mentioned before, we used the same utility functions as presented in [9] for this work. The utility function should decrease when the response time increases. The function, $U(r)$, is defined as:

$$U(r) = 100 * (e^{-r} + e^{-(r+\beta_r)}) / (1 + e^{-(r+\beta_r)}) \quad 3.1$$

The above formula is the utility function for response time from [9], where r is the response time, which is measured by running the service. And β_r is the Service Level Agreement (SLA) value for response time. Mei and Meeuwissen in [44] explained that the SLA is a concept to get QoS guarantees between service providers and consumers at the network level. We can regard the SLA as a measure standard, which can be used to evaluate the service quality. If the real value is close to this SLA value, the service can be considered in good quality. The SLA value can then be negotiated between the service consumer and service providers. To simplify the situation, the SLA value is defined as 1 sec in our scenarios.

The response time is defined as the time a service takes to complete its task as explained in [45]. The algorithm to retrieve the service response time is presented as following figure.


```

set serviceStartTime = the current computer time;

start the service;

invoke the service functionalities;

get the service result;

stop the service;

set serviceStopTime = the current computer time;

serviceResponseTime = serviceStopTime- serviceStartTime ;

return serviceResponseTime;

```

Figure 3.2 Algorithm to Retrieve Service Response Time

3.1.2. Utility Function Regarding to Throughput

The utility function should increase when the throughput increases. The function, $U(t)$ is defined as:

$$U(t) = (100 * (1 + e^{\beta t}) / (e^{\beta t} * (1 + e^{(-t + \beta t)}))) - 100 / e^{\beta t} \quad 3.2$$

The above formula is the utility function for throughput from [9], where t is the throughput monitored from each service. The β_t is the throughput SLA. In our case, the SLA value is defined as 1 kbyte/ses.

The throughput is defined as processing rate per service request. Zuquim et al. in [45] indicated that measuring service throughput is very complicated since the throughput could be impacted with many factors, such as latency, network traffic, hardware limitation, etc. Also in some cases,

the unit of throughput is measured as transactions/second. But in most of cases, the throughput is measured as byte per second. In our research, the later one will be used.

To simplify the situation, we simply define the throughput as following formula:

$$\text{Throughput} = \text{The size of service response} / \text{response time} \quad 3.3$$

The above formula describes the throughput calculation. And the algorithm to retrieve service throughput is described as the following:

```
calculate serviceRequestBufferSize;  
set serviceStartTime = the current computer time;  
start the service;  
invoke the service functionalities;  
get the service result;  
stop the service;  
set serviceStopTime = the current computer time;  
calculate serviceResponseBufferSize;  
serviceResponseTime = serviceStopTime- serviceStartTime ;  
throughput = serviceResponseBufferSize/ serviceResponseTime  
return throughput;
```

Figure 3.3 Algorithm to retrieve service throughput

3.1.3. The Overall Utility Function

The overall utility function is presented as following:

$$U(o) = W(r) * U(r) + W(t) * U(t) + W(x) * U(x) \quad 3.4$$

Where $W(r)$, $W(t)$ are the weights assigned to response time and throughput and $W(x)$ is the weight for any additional dynamic parameters. The relationship and requirement as defined as following:

$$\begin{aligned} W(r), W(t), W(x) &\in (0,1) \\ W(r) + W(t) + W(x) &= 1 \end{aligned}$$

This overall function can be extended when there are more QoS attributes added.

3.2. UTILITY-BASED QOS BROKERING ARCHITECTURE FOR SEMANTIC WEB SERVICE SELECTION

There are many ongoing solutions for semantic web service proposed. In this thesis, we have chosen one of the most prominent frameworks referred to as Web Service Modelling Ontology (WSMO). We use the same utility functions described in the previous to evaluate the QoS information in semantic environment. The architecture is shown in Figure 3.4.

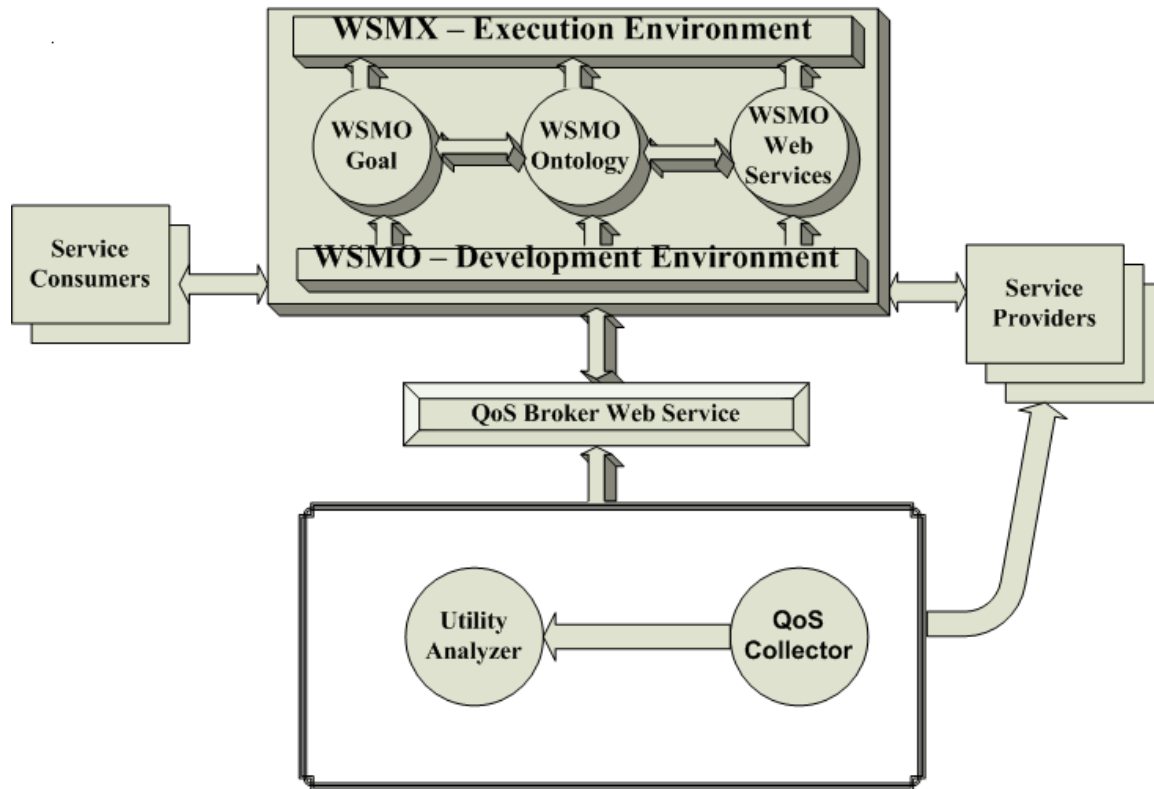


Figure 3.4 The QoS broker merged with WSMO/WSMX architecture

The ultimate goal of this architecture is to achieve the QoS-based dynamic service discovery and selection mechanism in the WSMO semantic environment. The architecture is supposed to discover and select among WSMO web services, which have already been deployed in the WSMO framework with semantic description.

The task of the broker is to collect the QoS information and analyze the overall utility value. The utility value is supposed to provide to WSMO solution seamlessly. In this proposed architecture, the utility value is packaged and published as a web service. Thus, the WSMO elements can retrieve the value from this web service. And finally, the WSMO/WSMX architecture is designed to achieve the task of service selection based on the overall utility value.

Utility functions play an important role in evaluating the QoS information. However, simply the function alone is not sufficient enough. A middle layer is necessary in order to convey the utility value between the service providers and consumers. The goal of this brokering architecture is to collect the QoS information, analyze the utility value and deliver the result to the consumers. The general architecture consists of two main parts: QoS Collector and Utility Analyzer, as presented in Figure 3.4.

The QoS Collector is designed to retrieve all QoS value from each web service. In this case, we only collect the response time and throughput. The Utility Analyzer is designed to retrieve the overall QoS value. The analyzer first receives each QoS value from the collector, and applies them to the respective utility function. Finally, the overall value is calculated based on the assigned weight for each QoS attribute defined in the overall function.

To validate this proposed architecture, we select one of the SWS Challenge scenarios named hardware purchasing, as our experimental environment. The scenario has three service providers, presenting as traditional web services. Each provider offers a few hardware products for the consumers to purchase. Also, all the semantic tasks are implemented and executed in the WSMO/WSMX architecture. In the test cases, the goal is defined as finding a product, which is offered by all the services. The product and its service will be only impacted and selected by its QoS information analyzed by the broker. Under this circumstance, the architecture can be validated for semantic web service selection with QoS constraints that will be explained in Chapter 4.

3.3. OVERVIEW OF THE SWS CHALLENGE SCENARIO - HARDWARE PURCHASING

The scenario of hardware purchasing in the SWS Challenge includes a series of web service providers. It provides the participants an experimental environment to test their solutions, while emphasizing on the topic of semantic service discovery and simple composition. Each service provider offers its own array of products, which are identified by a unique product ID. Also, in order to avoid complexity, we define all services to allow ordering a single product for each transaction. The three service vendors are addressed and described as following:

1. Vendor Rummage, <http://sws-challenge.org/shops/Rummage.wsdl>
2. Vendor Hawker, <http://sws-challenge.org/shops/Hawker.wsdl>
3. Vender Bargainer, <http://sws-challenge.org/shops/Bargainer.wsdl>

The following table presents the examples of the product offered by the above vendors.

Table 3.2 Examples of the products offered by the service vendors

(For the detailed information, please see the Appendix A.)

Product Name	Product ID	Provider	Description
Notebook	00000001	Rummage	13” flat screen; 1.83 GHz Intel Core Duo;Memory : 512 MB DDR2 – SO-DIMM;HDD : 60 GB; color : white; price:1,099.00

Web_cam	00000011	Rummage	VGA 640 x 480;price:149.00
Notebook	00000004	Bargainer	13” flat screen; 2.0 GHz Intel Core Duo; Memory: 1GB DDR2– SO-DIMM;HDD: 100 GB; color : white; price:1,449.00
Docking_station	00000016	Bargainer	docking station for X41 (00000008); price:269.95
Notebook	00000003	Hawker	13” flat screen;2.0 GHz Intel Core Duo; Memory : 512 MB DDR2– SO-DIMM; HDD: 80 GB; color: white; price: 1,349.00
Accessory	00000015	Hawker	Neoprene sleeve; price: 29.95

3.4. MODELING QOS CONSTRAINTS IN THE WSMO SEMANTIC CONTEXT

In the traditional SOA environment, the QoS information is normally registered in the UDDI entry, and can be retrieved by the consumers or the brokering systems. Li and Zhou explained in [28] that such mechanisms based on the UDDI have been proved inefficient due to the fact that their discovery results always contain irrelevant and unsatisfied items. This problem is caused by

the lack of semantic support. The WSMO framework has been proposed to solve this issue by its semantic characteristic.

As we introduce the WSMO service discovery mechanism in Chapter 2, the WSMO architecture provides both functional and non-functional service discovery and selection mechanism. The static QoS information is defined by non-functional property discovery. In the WSMO definition, non-functional properties are described as cost-related and charging-related properties of a web service, such as network-related QoS, security, cost, performance, reliability, etc. The task of processing static QoS attributes can be achieved by WSMO keyword discovery (also non-functional properties discovery)

Dynamic QoS attributes have more impacts on the service selection. Things become more complicated due to the characteristic of dynamic. The dynamic QoS attributes can be implemented in WSMO functional discovery. Through out this work, we focus on the functional discovery of dynamic QoS attributes.

We propose the functional discovery model based on the WSMO framework in order to solve the issue revealed in Figure 3.4. In this proposed architecture, the QoS broker acts as a middle layer between the service provider and WSMO framework. The main task is to collect the QoS information, and to further analyze it with the utility function. The broker is published as a web service, and the WSMO solution can retrieve the QoS value from this web service.

To achieve this functional discovery, we add the QoS logic by extending the existing semantic

solution for hardware purchasing. The general algorithm of implementing this functionality is described as following:

create a WSMO concept to define the overall QoS value
instantiate this concept for each service.
define a WSMO goal to implement the discovery and selection algorithm

Figure 3.5 The general algorithm for functional discovery

First of all, we introduce a new concept, QoSValue, into the ontology, named ProductOntology. The Concept is a WSML element, which describes the data type used in the application. The concept is very similar to the class/interface defined in the Java language. The WSML code of this new concept is presented as the following:

```
concept QoSValue  
  qos_value ofType _decimal
```

Figure 3.6 A new concept, QoSValue

The concept name is QoSValue; and it contains one data, qos_value, which is the data type in decimal format.

Also, it is necessary to define an instance of this QoSValue concept for each service. The instance is also a WSML element, which stands for a real value of the concept. Normally, the instances are defined in each WSMO web service. In this case, an instance of the concept QoSValue is added to each service as following:

```
instance Bargainer_QoSValue memberOf po#QoSValue  
po#qos_value hasValue 81.29
```

Figure 3.7 An QoSValue instance for service Bargainer

```
instance Hawker_QoSValue memberOf po#QoSValue  
po#qos_value hasValue 79.50
```

Figure 3.8 An QoSValue instance for service Hawker

```
instance Rummage_QoSValue memberOf po#QoSValue  
po#qos_value hasValue 82.45
```

Figure 3.9 An QoSValue instance for service Rummage

The string `po#` stands for the ontology of ProductOntology, which means the concept is located in this ontology, and that this pre-string can guide the system to find the concept. Each instance has an attribute of `qos_value`, which inherits from the concept, `QoSValue`. Also, the attribute is assigned a decimal value. This decimal value is the overall QoS value. It is supposed to retrieve from the QoS broker web service. To simplify the situation, we hardcode this value to make it easy to conduct the experiments. In the real system, the QoS broker web service is providing this value that should be automatically feed to the `QoSValue` instance for each service. For example, the Java API for WSMO can be used to achieve this automation. .

A WSMO goal is defined to implement the discovery and selection algorithm. In this WSMO

goal, we define the consumer's requirement for the product, which is the notebook that described in Figure 3.10.

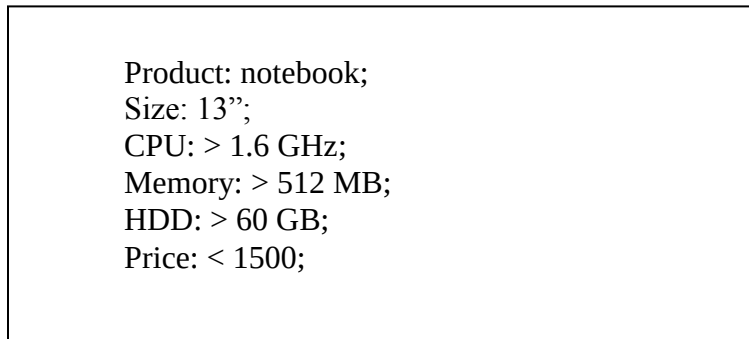


Figure 3.10 Description of the Goal for Notebook Purchasing

The goal requirements combine with the QoS constraint. The WSML element, Capability, is responsible for fulfilling this functionality. The capability defines the states before/after the service execution, which are pre-condition/assumption and post-condition/effect respectively. . In this case, we define the post-condition, which implement the logic of selection criteria. Service consumers can identify which service can meet the selection criteria of notebook and QoS information. Figure 3.11 shows the definition of capability for the goal. In the definition of “nfp” (non-function property), two variables (?p and ?q) are defined respectively for concept of Notebook and QoSValue. Also, in the definition of postcondition, the selection criteria are setup. In this case, the QoS criteria is set to be larger than 80. So, the service selection result is supposed to show all services meeting this requirement: Bargainer, Rummage and Hawker. An experiment of this case will be made in the next evaluation chapter.

```

capability GoalA1Capability
nfp
_ "http://www.wsmo.org/goal/discovery/instancebased/mainElements" hasValue
{"?p","?q"}

endnfp
/*goal: Product: notebook;
   Size: 13"; CPU: > 1.6 GHz;Memory: > 512 MB;
   HDD: > 60 GB; Price: < 1500;
*/
postcondition
definedBy
(
    ?p[po#name hasValue "Mac Book 13",
        po#processorGHz hasValue ?procGhzX,
        po#price hasValue ?priceX,
        po#hddGB hasValue ?hddGBX, po#memoryMB
hasValue ?memMBX] memberOf po#Notebook
        and ?procGhzX >= 1.6
        and ?memMBX >= 512
        and ?hddGBX >= 60
        and ?price < 1500

    and

    ?q[po#qos_value hasValue ?qosValue] memberOf po#QoSValue
    and ?qosValue > 80
)

```

Figure 3.11 Capability for the Goal of Notebook Purchasing
with QoS Selection Criteria
(services are selected by comparing with the base value)

Furthermore, we present another algorithm, which makes the service with higher QoS utility value is selected. The WSMO framework provide a functionality named rankingcriteria, which allows developer to define the criteria either high better or lower better. This functionality is defined in the capability/nfp as well. Figure 3.12 shows the implementation of this algorithm.

```

capability GoalA1Capability
nfp
_ "http://www.wsmo.org/goal/discovery/instancebased/mainElements" hasValue
{"?p", "?q"}
_ "http://www.wsmo.org/goal/discovery/instancebased/rankingcriteria" hasValue
{"?qosValue-HigherBetter")

endnfp
/*      Product: notebook;
      Size: 13";CPU: > 1.6 GHz;Memory: > 512 MB;
      HDD: > 60 GB;Price: < 1500;
      */

postcondition
definedBy
(
    ?p[po#name hasValue "Mac Book 13",
        po#processorGHz hasValue ?procGhzX,
        po#price hasValue ?priceX,
        po#hddGB hasValue ?hddGBX, po#memoryMB
hasValue ?memMBX] memberOf po#Notebook
        and ?procGhzX >= 1.6
        and ?memMBX >= 512
        and ?hddGBX >= 60
        and ?price < 1500

    and

    ?q[po#qos_value hasValue ?qosValue] memberOf po#QoSValue
    and ?qosValue > 80
).

```

Figure 3.12 Capability for the Goal of Notebook Purchasing with QoS Selection Criteria

(Service with highest QoS utility value is supposed to be selected)

In this algorithm, we keep the selection criteria defined in Figure 3.12 unchanged and add the rankingcriteria variable to be “?qosValue-HigherBetter”. The rankingcriteria variable consists of two parts, shown in the following figure.

```
rankingcriteria variable = ? instance_variable_name + “-HigherBetter/LowerBetter”
```

Figure 3.13 Definition For rankingcriteria Variable

The selection result is expected with the service in the highest QoS utility value. An experiment for this algorithm is also made in the next evaluation chapter.

3.5. Modeling Static QoS Parameters in the Proposed Architecture

The algorithm described in Section 3.4 can also be extended to apply the static QoS constraints (e.g. service price). Same to the procedure shown in Figure 3.5, the WSMO concept of QoSPrice should be created similar to Figure 3.6. Also, the corresponding instance of QoSPrice for each service need to be created similar to Figure 3.7 and set in each WSMO web service. An example is given in Figure 3.14, including the concept of QoSPrice and the corresponding instance for service Bargainer.

```
concept QoSPrice
  qos_price ofType _decimal

instance Bargainer_QoSPrice memberOf po# QoSPrice
po#qos_price hasValue 0.15
```

Figure 3.14 QoSPrice Concept and its instance for Service Bargainer

Then, the goal can be extended by adding QoSPrice criteria, which is same as stated in the algorithm Figure 3.16. The only difference is that the QoS price is provided by service providers,

which is published in registry and can be retrieved from that registry (i.e., UDDI), while those dynamic QoS attributes need to be measured in the execution time and will be published by QoS broker service as the overall utility value. When the combination of both dynamic and static QoS parameters are needed, the algorithms of Figures 3.11 and 3.12 can simply be extended with the "and" keywords to combine all section criteria in the Goal definition.

3.6. SUMMARY

In this chapter, we introduce our proposed solution to semantic service selection by applying utility-based on the QoS information. The utility function in evaluating the QoS attributes is first discussed. Also the QoS brokering architecture to collect and analyze the QoS information is presented. Then, this QoS broker in semantic environment (WSMO) is presented. Finally, modeling QoS parameters in WSMO context is introduced and two algorithms implementing the selection criteria are presented. In the next chapter, we'll make a series of experiments to evaluate the proposed architecture.

CHAPTER 4

EVALUATION OF THE MODEL

In this section, we evaluate the proposed architecture presented in the previous chapter. The evaluation is to demonstrate that the architecture can achieve the dynamic service selection by applying utility-based QoS information. Also, the evaluation is divided into three parts:

- Evaluating utility functions regarding to the QoS attributes.
- Evaluating the QoS broker in two parts: QoS collector and utility analyzer.
- Evaluating the architecture for semantic web service election in WSMO environment.

Among these three parts, the first two evaluations on the QoS broker will be briefly presented and the last one, the proposed architecture in WSMO environment, will be emphasized. In the following section, the evaluation environment is introduced in Section 4.1, and the above evaluations are conducted in Section 4.2, 4.3, and 4.4 respectively. Finally in Section 4.5, we summarize the whole evaluation.

4.1. EVALUATION ENVIRONMENT

In this section, we discuss the evaluation environment for the utility function applied in QoS-based dynamic service selection in WSMO environment.

4.1.1. Evaluation Objective

The objective of the evaluation is to verify the following issues:

- All the QoS information can be collected.
- Utility functions can measure the value for each QoS attribute properly.
- The overall utility value can be calculated by the function.
- The suitable services can be found and selected based on the overall value in WSMO semantic solution.

To achieve the above goals, we design a series of test cases. Also, to simplify the process and implementation, we apply two typical dynamic QoS attributes in the experiment, response time and throughput,

4.1.2. Experiment Components

The evaluation system involves four components, which are described as the following:

1. **Service Providers:** as it was introduced in the previous chapters, we have chosen one of the scenarios in the SWS challenge, hardware purchasing, as the experimental case. Also, we use the existing solution and extend it with QoS constraints. As mentioned before, the scenario contains three service providers: Rummage, Hawker, and Bargainer.
2. **Service Consumer:** the service consumer is simulated using a java application. The consumer application is implemented in JAX-WS 2.0 and executed in Tomcat 6.0. The application runs in an independent machine, which can send the request to QoS broker,

which the broker runs on the same machine. The service consumer also can be simulated in SoapUI. The SoapUI is a web service simulation tool, which provides a GUI interface to invoke web services, and the result is supposed to be presented in the SoapUI console.

3. **Utility function:** we introduce the utility function in chapter three. In this experiment, the utility is supposed to apply with two QoS attributes, response time and throughput. As mentioned before, when the response time is increasing, we expect for the utility function to decrease. Also we expect for the utility function to be increasing with the increase of throughput. Therefore, we measure these two dynamic attributes and pass them on as the QoS parameters in the experiments.

The overall QoS value will be calculated by these two attributes. The formula is presented in 3.4. The overall utility value will be used in the next evaluation for service selection.

4. **QoS Broker:** the QoS broker is also a java application, developed in the Netbeans 6.5, and deployed in Tomcat 6.0. The broker is represented as a web service to the consumer. The main functionality of the broker is to receive the service request from the consumer, analyze the QoS factors with utility function, select the service that meet the QoS constraints, and send the selected service back to the consumer.

4.2. EVALUATING UTILITY FUNCTIONS

This evaluation is to demonstrate the utility functions that have been introduced in Chapter three can fulfill the following tasks:

- Generate the overall QoS value by measuring each network QoS attribute.
- The overall value can reflect the real QoS status in the current service execution.

The main purpose of the first experiment is to verify if the relation between service response time and its utility function is decreasing. The utility function for response time is defined in formula 3.1. For the reason of stability, we collect response time for each service for 50 execution times through the QoS collector. The data collected is presented in Figure 4.1 -4.3.

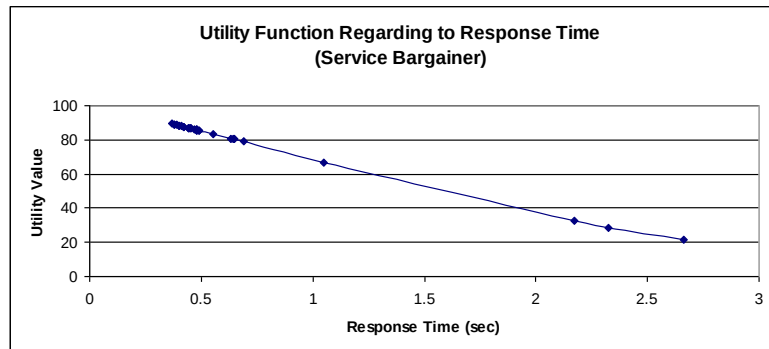


Figure 4.1. Utility Function Vs Response Time for service Bargainer

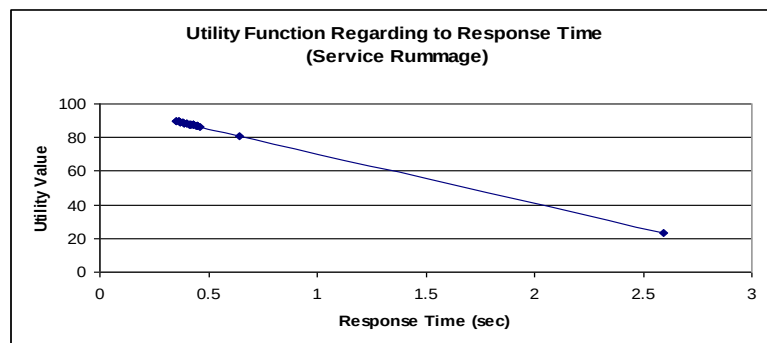


Figure 4.2 Utility Function Vs Response Time for service Rummage

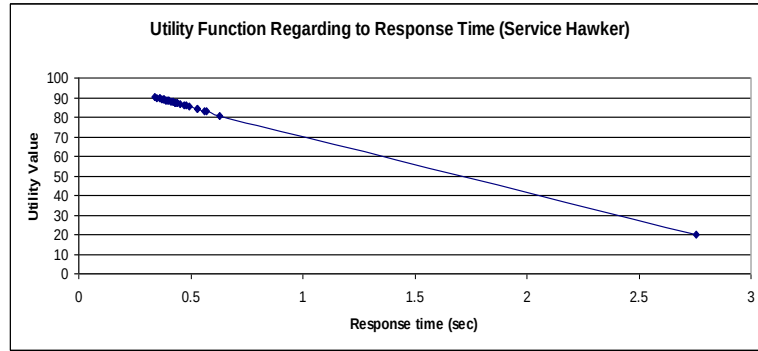


Figure 4.3 Utility Function Vs Response Time for service Hawker

As we see the results, the utility value decreases while the response time increases, which is the same as we expected. Also, the graph shows that where the response time is close to 1 sec (since we setup the SLA value to be 1 sec), the measured utility values converge to a small range (e.g. 80-90). While the response time is far further from 1, the utility value decreases or increases exponentially. It is one of the advantages of utility function, which can reflect the real status of the response time.

The purpose of the second experiment is to check if the utility value is monotonically increasing with the increase of the throughput. The utility function for response time is defined in formula 3.1. The data collected is presented in Figure 4.4-4.6.

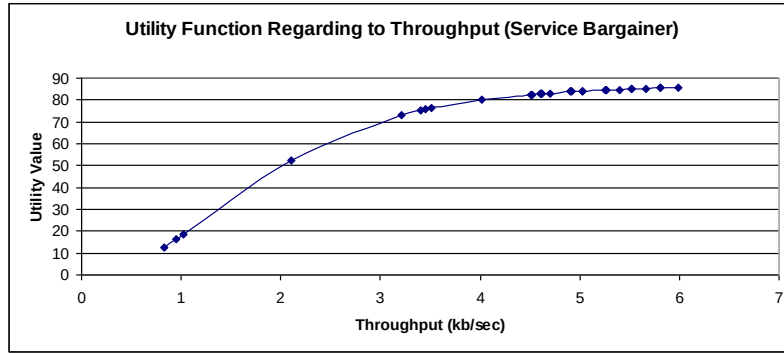


Figure 4.4 Utility Function Vs Throughput for service Bargainer

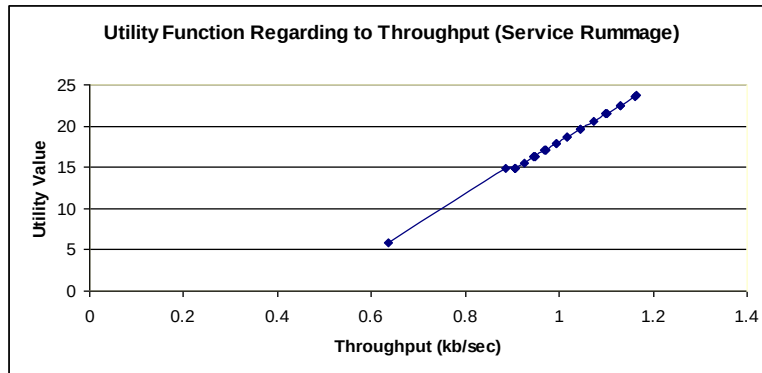


Figure 4.5 Utility Function Vs Throughput for service Rummage

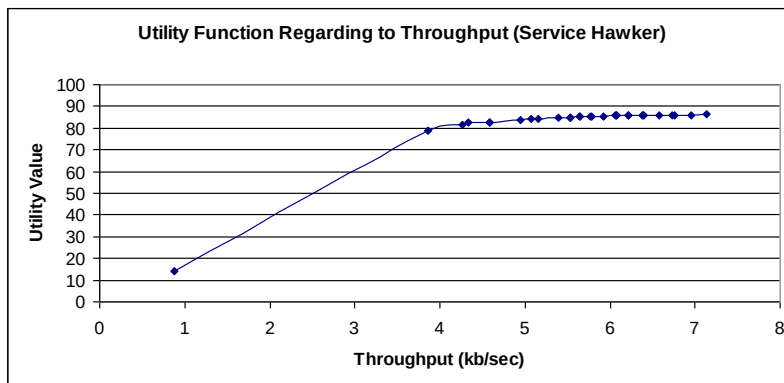


Figure 4.6 Utility Function Vs Throughput for service Hawker

As we see the result, the utility value presents monotonically increasing when the throughput increases. We can say this experiment helps achieve our expectation, and that the utility function for throughput can reflect their relationship properly.

4.3. EVALUATING THE QOS BROKER

The QoS broker is implemented in the Netbeans with JDK 6 and AX-WS 2.0, and it's deployed in Tomcat 6.0. The broker is presented as a web service. Service consumers can invoke the service by creating a web service reference. The architecture of the broker is presented in the 3.10. Also, the QoS broker and service consumer run on the same computer, which is Dell Inspiron 9200. The computer is configured with 1.8 Hz Intel Pentium processor and 1.25 GB of RAM. The network adapter is Intel PRO/wireless LAN 2100.

The evaluation is to demonstrate that the broker can collect the QoS attributes from the service providers, retrieve and measure the overall QoS value, and find the suitable service. Also, it demonstrates that the result can be invoked and delivered to the consumer. The evaluation is divided into the following two parts to fulfill the mentioned tasks.

4.3.1. Evaluating QoS Collector

The QoS collector is responsible for collecting various QoS values. This experiment collects response time and throughput for each service, and retrieved that five times for each. Appendix B presents the implemented algorithm and detailed experiment result.

The result in Appendix B shows the response time and throughput for each service provider. In the first execution of each service, it takes more time to get the response since the first connecting to the service occupies more time than others. In the following executions, the response time is stable and consistent. Also the throughput is calculated based on the response time and the buffer size of service response.

4.3.2. Evaluating Utility Analyzer

The utility analyzer is part of the QoS Broker, which is responsible for retrieving the utility value for each QoS attribute. The goal of the experiment is to demonstrate that the utility analyzer can evaluate the QoS attribute from service providers and output the utility value for the service selection. Appendix C presents the implemented algorithm and detailed experiment result.

The result in Appendix C shows that the utility analyzer can fulfill its task as we expected. The utility value can be retrieved and reflect the real response time in service selection. The less response time consumed, the higher utility value is calculated, and the more probability of service is selected. The next experiment is designed to verify this assumption.

4.4. EVALUATION OF UTILITY-BASED QOS BROKER IN WSMO SEMANTIC SERVICE SELECTION

In this section, we evaluate the proposed WSMO semantic service selection architecture based on the QoS information. A series of simulations are setup to be executed in the environment of WSMX. The goal of these simulations is to demonstrate that the proposed solution reaches our pre-defined objective, which is that the QoS information can be achieved in WSMO environment and play an important role in the semantic web service selection.

This evaluation is to extend the previous experiment in service selection in the semantic environment. In this section, the service selection mechanism will be implemented in semantic description language. The two selection algorithms defined in Figure 3.11 and 3.12 will be implemented and evaluated in the experiment. Also, to simplify the complexity, we only consider the utility value of response time in the selection algorithm. Finally, we would like to emphasize that this experiment extends the existing solution of hardware purchasing in the SWS-challenge by applying QoS constraints.

4.4.1. Experimental Environment

The evaluation is designed to demonstrate the QoS value can affect the service selection in the WSMO semantic environment. The experiment consists of examining four elements: service providers, QoS broker, WSMO/WSMX architecture, and a service consumer. The architecture is presented in Figure 3.4. There are three open source software applications need to complete this

evaluation: WSMO studio, WSMX execution environment, and SoapUI simulating as a service consumer.

First of all, the development environment WSMO studio and the semantic execution environment should be downloaded and installed. The development environment, WSMO studio, was first released by the WSMO research group in 2007 [46]. It is a GUI tool for developing WSMO projects with semantic web service annotations. It is an open source project built upon Eclipse platform, which can be extended by any third party. The studio can check the WSMML syntax while writing the code, which can help prevent the syntax errors in the editing stage.

As a semantic execution environment, WSMX provided the environment to process WSMML files. The execution environment can be accessed via a web service provided by WSMX. The functionality, AchieveGoal, in the web service is responsible for fulfilling the execution task. The steps involving with the evaluation are described as following:

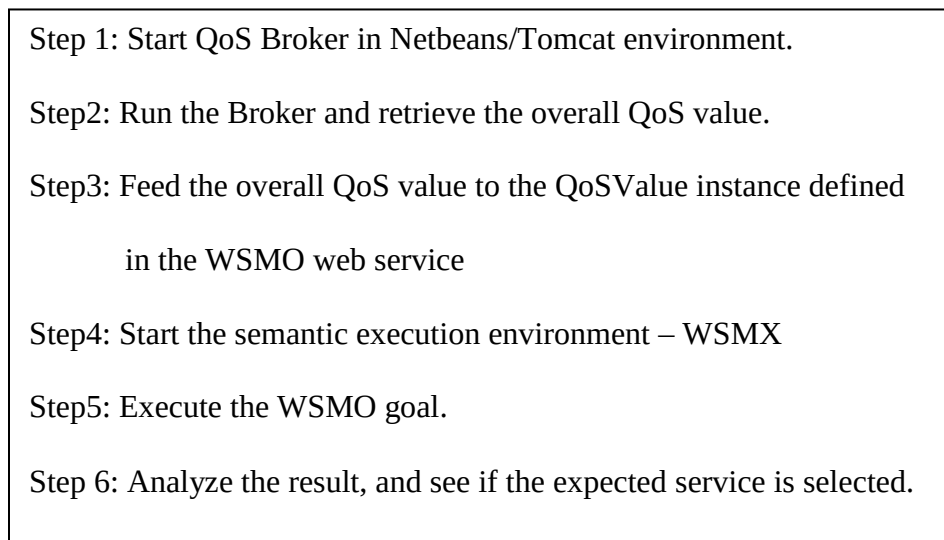


Figure 4.7 Experimental Execution Flow

The service consumer for this evaluation is simulated in SoapUI as we stated before. It is an open source freeware. Also, please note that we need step3 since the overall QoS utility value retrieved in the QoS broker was not seamlessly fed to the selection algorithms in WSMO. The overall utility value is manually updated to the semantic solution in this work.

The simple test for semantic web service solution was performed according to the above execution flow shown in Figure 4.7 without considering any QoS constraints. The scenario and test case result is shown in Appendix D.

4.4.2. Simulation 1: Service Selection Algorithm with a Base QoS Utility Value

For all the experiments of this section we used the goal in the way that all services meet the requirement. In the first experiment, the algorithm presented in Figure 3.11 will be evaluated. The goal of the experiment is to demonstrate the service selection can be impacted by the base QoS utility value defined in the selection criteria. When the semantic solution is not configured with QoS constraints, all the service providers meet the requirement and all the services are supposed to be selected. However, after we add the QoS constraints, only the services which meet the selection criteria can be selected. Therefore, we know which service selection can be affected by the QoS constraints. We will setup three test cases to demonstrate this hypothesis.

In the first test case, a base value is setup for the selection criteria on purpose which should result in selecting all the services. As the result of this test is shown in Table 4.1, all the services are selected.

Table 4.1 Result of Service Selection for Test Case one
(For Selecting Service with the Utility Value Greater Than 80)

Service Providers	The Overall Utility Value	Selection Criteria	Service Selected
Bargainer	87.343	> 80	selected
Hawker	83.169	> 80	selected
Rummage	88.004	> 80	selected

As we can see in Table 4.1, all services are selected based on the selection criteria. Now, in the test case two, we change the selection criteria to be greater than 85, the services with the overall utility value beyond this value should be selected. Table 4.2 shows the result of this test case. The service Bargainer and Rummage are selected since they have the utility value greater than 85. The result is same as we expected.

Table 4.2 Result of Service Selection for Test Case Two
(For Selecting Services with the Utility Value Greater Than 85)

Service Providers	The Overall Utility Value	Selection Criteria	Service Selected
Bargainer	87.343	> 85	Selected
Hawker	83.169	> 85	Not Selected
Rummage	88.004	> 85	Selected

In the last the test case, the selection criteria is setup to be greater than 88 in order to make the service with the highest utility value to be selected. Table 4.3 presents the result of this test case. The service Rummage is selected since it has the highest utility value. The result meets the expectation we made before.

Table 4.3 Result of Service Selection for Test Case Three:
(For Selecting Services with the highest Utility Value)

Service Providers	The Overall Utility Value	Selection Criteria	Service Selected
Bargainer	87.343	> 88	Not Selected
Hawker	83.169	> 88	Not Selected
Rummage	88.004	> 88	Selected

4.4.3. Simulation 2: Service Selection Algorithm with the Criteria for the Service with Highest QoS Value Selected

In this experiment, we evaluate the second service selection algorithm in WSMO context, which presented in Figure 3.12. The goal of this experiment is to validate this algorithm and demonstrate the rank criteria defined in the algorithm can work for the service selection. This algorithm is extended the algorithm in Figure 3.11 by adding the ranking criteria with highest QoS utility value.

To achieve this goal, we design three test cases by executing the selection mechanism 50 times. In the first test case, we do not set any extra factor to impact the result of service selection, and expect that the service selection is reflected by real utility value of the involved services. In the second case, we introduce a selection factor (α) to the service Bargainer, which can make this service selected more. In the third case, we adjust this selection factor so we can see if the affected service will be selected much more than the other two services in the long run.

In the first test case, the service request sequence is executed three times. Each sequence executes the selection 50 times. Figure 4.8-4.10 presents the selection result. This result shown in figures reflected the real status of the utility value for each service, and it is as we expected. Also in each execution time, the utility value of each service is different.

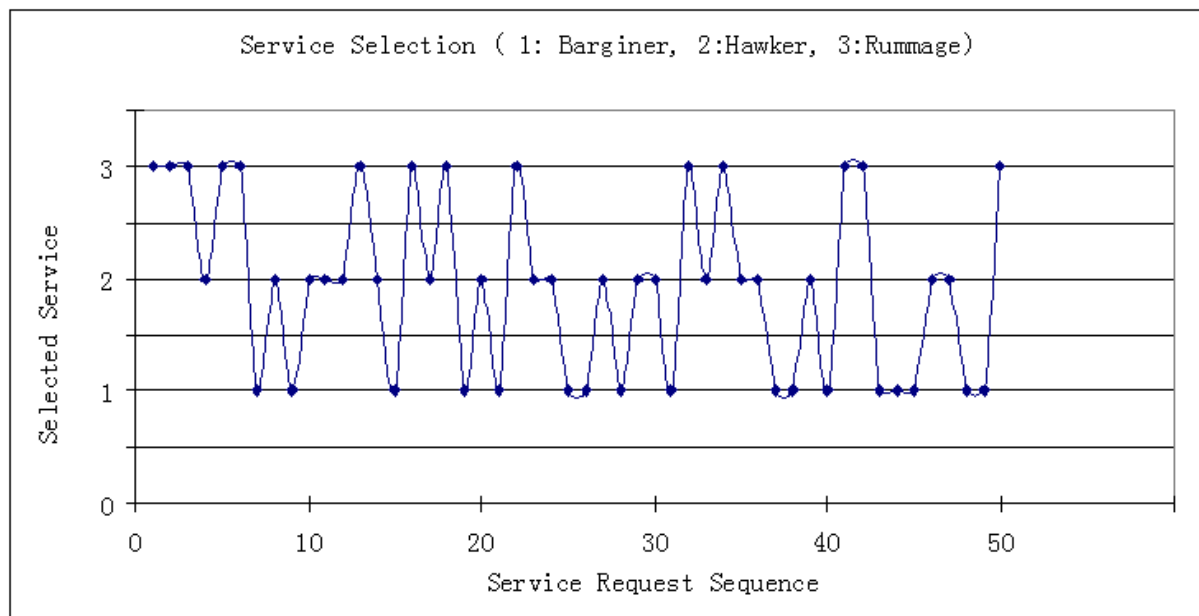


Figure 4.8 Service Selection Sequence A in WSMO Environment
(without extra selection factor)

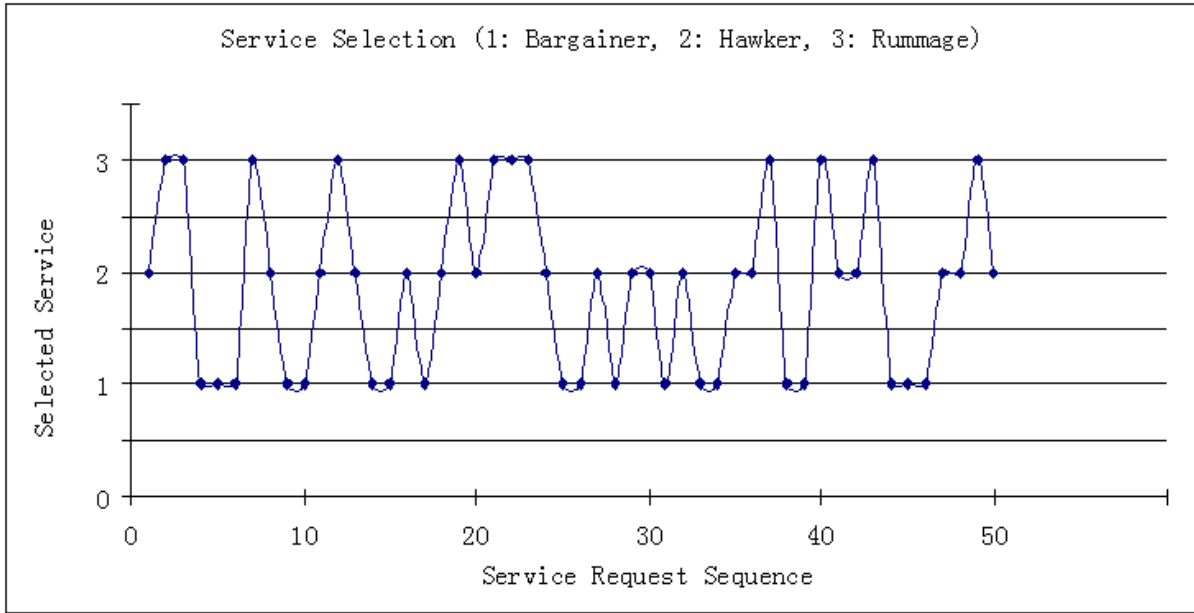


Figure 4.9 Service Selection Sequence B in WSMO Environment
(without extra selection factor)

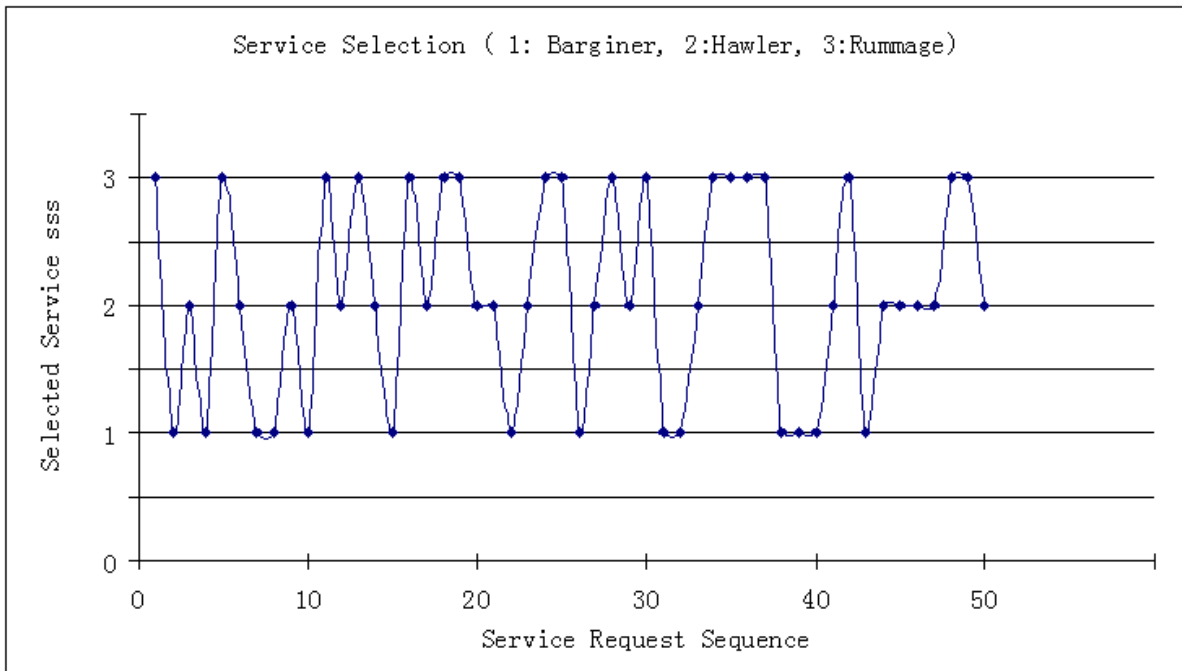


Figure 4.10 Service Selection Sequence C in WSMO Environment
(without extra selection factor)

In the long run the number of selection between three services is random. Table 4.4 concludes the service selection number the above three request sequences and shows that the service selection in the long run is random.

Table 4.4 Service Selection Result for Three Request Sequence

Service Request Sequence	Service Rummage Selected	Service Hawker Selected	Service Bargainer Selected
Sequence A	14	19	17
Sequence B	12	19	19
Sequence C	18	18	14

In the second test case, an extra selection factor α is introduced. In this case, we set α equal 1.005, and only add the factor to the service Baigainer. For the each execution, the utility value of service Bargainer multiplies with the value of α . Thus the service Bargainer is expected to be selected more than the others. Figure 4.11 presents the result of this test case. The result shows that the service Rummage is selected 13 times, the service Hawker is selected 16 times, and the service Bargainer is selected 21 times. This result meets our expectation.

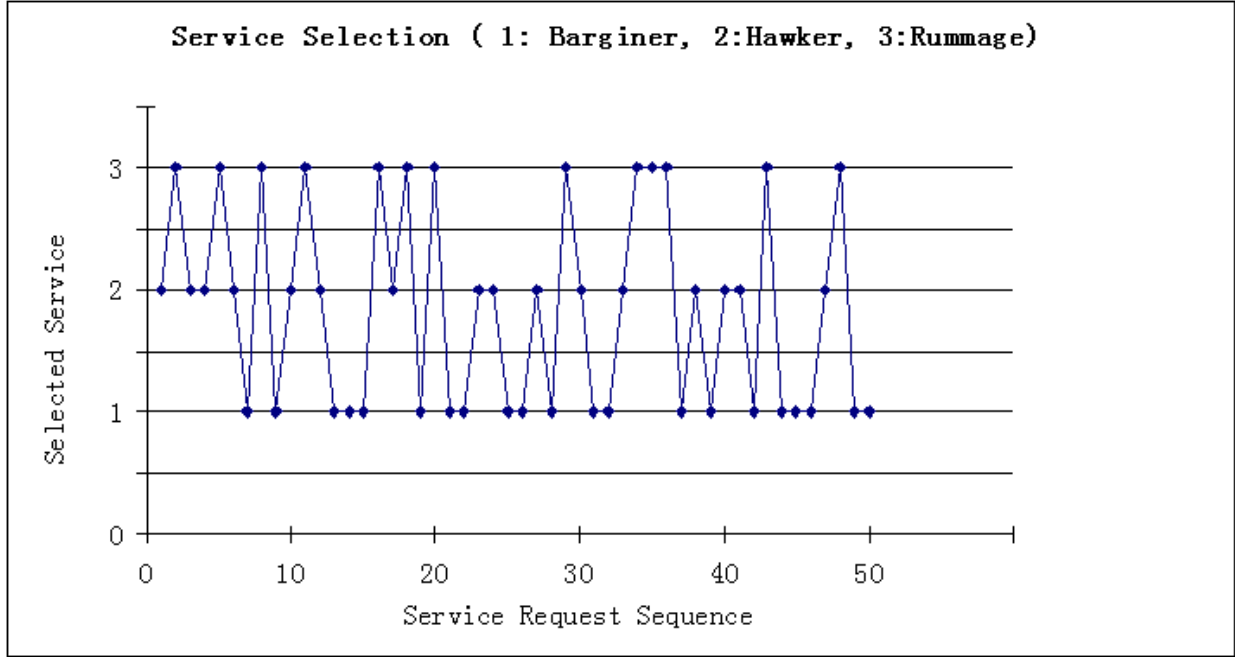


Figure 4.11 Service Selection Sequence in WSMO Environment

(with extra selection factor $\alpha = 1.005$)

In the third test case, this extra factor α is adjusted. The value of α is increased to be 1.01. Also similar to the second test case, only the service Bargainer is impacted with this factor. Thus, in this case, we expect the service Bargainer will be selected much more than others. Figure 4.12 shows the result that the service Rummage is selected 7 times, the service Hawker is selected 19 times, and the service Bargainer is selected 24 times. The result is same as we expected.

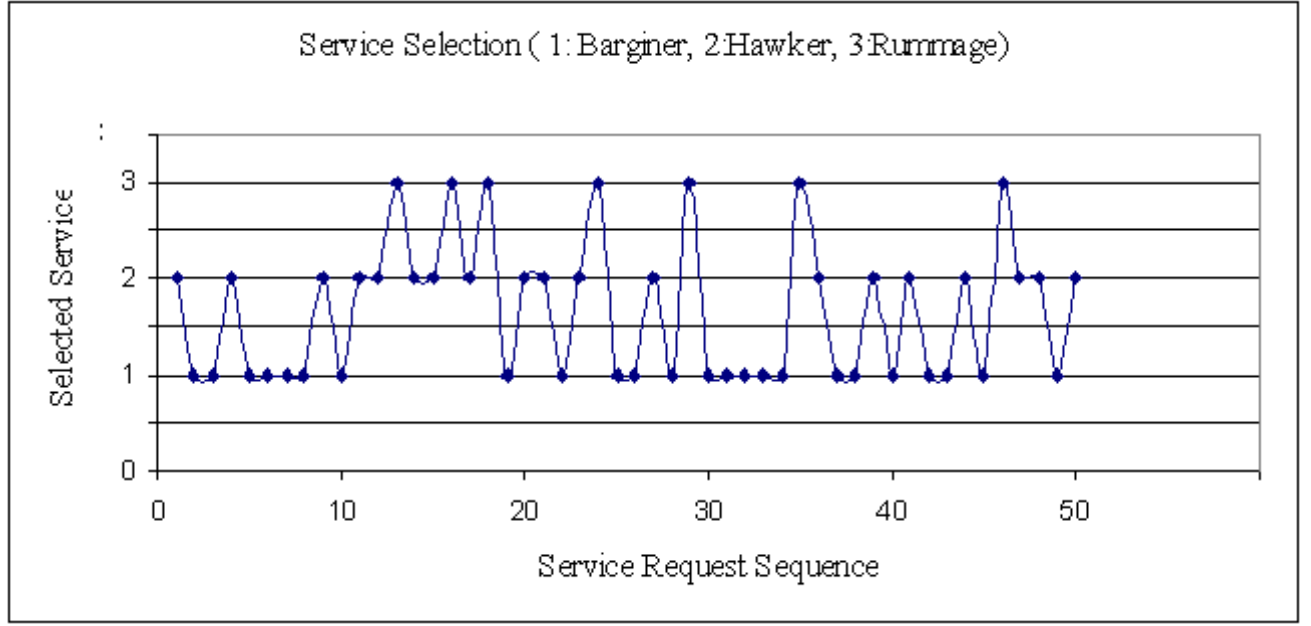


Figure 4.12 Service Selection Sequence in WSMO Environment
(with extra selection factor $\alpha = 1.01$)

4.5. SUMMARY

In this section, several experiments are conducted in order to evaluate our proposed architecture for QoS-based dynamic service selection. The utility functions regarding to the QoS information was first evaluated. The experiments demonstrate the utility functions can measure the QoS information for the service selection properly. Also, the service selection in the WSMO semantic environment was evaluated. Two experiments were conducted to demonstrate feasibility and efficiency of the semantic-based selection algorithms proposed in Chapter three. By analyzing the experiment results, we can verify that the proposed architecture can achieve the tasks of dynamic service selection based on the QoS parameters in the semantic context.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This chapter provides a conclusion for this work, and discusses some future works.

5.1. CONCLUSION

The topic of Semantic Web Service (SWS) has drawn great interest from both academia and the IT industry in recent years. There are several SWS challenges proposed by organizations who conduct the research on SWS. These organizations offer a series of SWS challenge scenarios to attract the researchers who will help contribute to their solutions.

This thesis discussed the topic of utility-based QoS for dynamic web service selection in the semantic context and proposed the architecture to achieve this task. It introduced utility functions to measure the QoS information. It also implemented a QoS broker to collect the QoS information and analyze its overall utility value. It applied this QoS broker into the WSMO environment to achieve the semantic service selection task based on the retrieved QoS value.

The proposed architecture with utility functions is designed to employ more QoS parameters. The dynamic parameters (i.e. response time and throughput) are used and tested in this work.

However, the other dynamic QoS attributes (e.g. availability, reputation) can be retrieved and merged into the overall utility function by considering the appropriate weight.

Moreover, a series of experiments were designed to evaluate this proposed architecture. The experiments were conducted based on the research on dynamic service selection in the semantic environment. The QoS information was successfully collected in the experiments, and the utility value of the QoS attributes was measured by applying the proposed functions properly.

Finally, the proposed model of dynamic service selection based on the QoS constraints was verified successfully in the WSMO semantic architecture. Two algorithms for semantic service selection were implemented and tested in the evaluation experiments. In summary, the experiments demonstrate the architecture can achieve the predefined tasks as expected. Also, the method proposed for modeling QoS parameters can be simply extended to capture more static and dynamic QoS parameters and use them for dynamic web service selection in the semantic context.

5.2. FUTURE WORK

There are still many challenges in SWS. In our future work, we would like to conduct our research in two main directions. One is the service composition in the semantic environment. Another is addressing the interoperability problem in the heterogeneous environment of SWS. We'll investigate how the SWS can solve these problems.

Web service composition in SOA has gained much interest in recent years. A single web service sometimes is incapable of performing some complex task. There should be a possibility to find a combination of existing web services that could fulfill such tasks. Web service composition is such a mechanism for creating this service combination to achieve the requester's goal [47]. It requires identifying a set of suitable services that constitute a composite service.

There are some approaches to achieve the service composition in the traditional SOA framework. These approaches are designed static service resources. The services are already known to the consumers and the service execution flow is predefined. However, the problem is complicated by the fact that the services may be uncertain and dynamic. When compositing services for the purpose of achieving some goal, services could be discovered and bound during the running time. These current approaches can't achieve such tasks due to the lack of automation.

The SWS can solve such problems, since it aims to achieve automatic tasks by providing semantic information. Therefore, another one of our future works is to investigate how the SWS can achieve dynamic service composition.

Furthermore, the distributed software systems are becoming increasingly complicated because of more complicated business environments. These systems present the characteristic of heterogeneity, which means incompatibilities in various different categories (platform, software, message format, etc.). One of the challenges in the heterogeneous systems is the interoperability.

Interoperability is a process to make heterogeneous systems be able to understand and communicate each other. The purpose of interoperability is to solve the incompatibilities in the heterogeneous systems. For example, to achieve the efficient communication between two different applications, it requires the message delivery. Nevertheless, the message format could not be interpreted by any other applications. Thus, the developers need to compromise a message format mapping between two applications, which only convey syntactical information description. The current web service technology can easily achieve this task. However, if there are more than two heterogeneous systems intertwined, the complexity raises sharply, which is much more than the notion of one plus one equals two. Obviously, the technologies only with syntactical support are reluctant to fulfill this task.

Thus SWS aims to solve such interoperability issues. This is another topic of our future work.

APPENDIX A

SERVICE VENDERS

**Note that the GTIN provides a (fictious) global unique product identifier!*

Table A.1 Product Information Provided by All Vendors

Rummage							
Product name	Mac Book 13"	Mac Book 13"	HP NX6325	IBM X41	iSight	Creative NX Ultra Web cam	HP-Docking Station
Category	Notebook	Notebook	Notebook	Notebook	Web_cam	Web_cam	Docking_station
Product specs	13" flat screen 1.83 GHz Intel Core Duo Memory : 512 MB DDR2 – SO-DIMM HDD : 60 GB color : white	13" flat screen 2.0 GHz Intel Core Duo Memory : 1 GB DDR2– SO-DIMM HDD: 100GB color : black	15,0" / XGA (1024 x 768) Pixel AMD Turion 64 X2, 1600 MHz 512 MB DDR2-RAM 80 GB HDD Double Layer DVD+/- RW/DVD-ROM ATI Radeon Xpress 1150 Ethernet 10/100/1000B-TX 56K Modem WLAN 802.11a WLAN 802.11b	12,1" XGA TFT Intel Pentium M 758 1,5 GHz 1 GB DDR2-SDRAM 40 GB HDD DVD/CD-RW 56K V.92 Modem Integrated Intel PRO 1000 Gigabit Ethernet IBM 11b/g Wireless	VGA 640 x 480	No specification	for NX9XXXX series (00000009, 00000007)

			WLAN 802.11g Bluetooth Windows XP Professional				
GTIN	00000001	00000002	00000005	00000008	00000011	00000014	00000017
Price	\$ 1,099.00	\$ 1,699.00	\$ 1,057.00	\$ 1,999.99	\$ 149,00	\$ 82,99	\$ 239,00

Hawker						
Product name	Mac Book 13"	HP NX6310	HP NX9000	Logitech QuickCam Express	Incase Neoprene Sleeve 13"	IBM-Dockingstation
Category	Notebook	Notebook	Notebook	Web_cam	Accessory	Docking_station
Product specs	13" flat screen 2.0 GHz Intel Core Duo Memory : 512 MB DDR2- SO-DIMM HDD : 80 GB color : white	15" TFT Intel Duo T2300+ 1.66GHz Centrino 512MB DDR-RAM 60GB HDD DVD Burner Wlan Windows XP	15" TFT XGA Pentium4-M 2.2 GHz Memory : 256 MB DDR-RAM HDD: 40 GB CD-RW / DVD-ROM Modem EtherNet (10MBit), Fast EtherNet (100MBit) WLAN 802.11g Windows XP Professional	VGA 352 x 288	Neoprene sleeve	for X41 (00000008)
GTIN	00000003	00000006	00000009	00000012	00000015	00000018
Price	\$ 1,349.00	\$ 905.99	\$ 857.00	\$ 23,90	\$ 29,95	\$ 199,00

Bargainer						
Product name	Mac Book 13"	HP NX9420	Mac Book 13"	Airport Extreme Base station	IBM-Dockingstation	Fjutsi Siemens Dockingstation
Category	Notebook	Notebook	Notebook	Networking	Docking_station	Docking_station
Product specs	13" flat screen	17" WXGA+	13" flat screen	WLAN-Router	docking station for X41	No specification

	2.0 GHz Intel Core Duo Memory : 1 GB DDR2– SO-DIMM HDD : 100 GB color : white	TFT (1440×900) Centrino Duo 1.83GHz 1GB DDR2- RAM 100GB HDD DVD+/-RW Windows XP Profesional Modem WLAN BlueTooth	2.0 GHz Intel Core Duo Memory : 2 GB DDR2– SO-DIMM HDD : 120 GB color: black	802.11b and g compatible transfer rate up to 54 Mbps	(00000008)	
GTIN	000000004	000000007	000000010	000000013	000000016	000000019
Price	\$ 1,449.00	\$ 1,249.99	\$ 2,049.00	\$ 199,00	\$ 269,95	\$ 234,00

APPENDIX B

QoS Collector for Response time Implemented in JAVA

The algorithm of retrieving response-time in QoS collector described in Chapter three is implemented in Java as following:

```
try{

    QoSResponseTimeMonitor QosRTM = null;

    for(int i=0; i<5; i++){

        QosRTM = new QoSResponseTimeMonitor();

        QosRTM.start();

        BargainerService bs = new BargainerService();

        BargainerPortType bpt = bs.getBargainerPort();

        ProductList pl = bpt.listProducts(ProductCategory.NOTEBOOK);

        QosRTM.stop();

        long timeElapsed = QosRTM.getElapsedTime();

        System.out.println("timeElapsed:"+timeElapsed);

    }

}

catch(Exception ex){}
```

Figure B.1 Implemented QoS response-time collector in Java

Experiment Result for Evaluating QoS Collector

In this experiment, we collect the response time for each service, and retrieved that five times for each. The data collected is presented in the following figures.

Table B.1 Experiment Result: the QoS information for Service Bargainer

# of Execution	Response Time(ms)	Throughput (kb/sec)
1	2.664	0.8314
2	0.651	3.4024
3	0.691	3.2054
4	2.323	0.9535
5	0.451	4.9113

Table B.2 Experiment Result: the QoS information for Service Hawker

# of Execution	Response Time(ms)	Throughput (kb/sec)
1	2.754	0.8830
2	0.491	4.9531
3	0.531	4.58
4	0.44	5.2727
5	0.391	6.2199

Table B.3 Experiment Result: the QoS information for Service Rummage

# of Execution	Response Time(ms)	Throughput (kb/sec)
1	2.664	0.8314
2	0.651	3.4024
3	0.691	3.2054
4	2.323	0.9535
5	0.451	4.9113

APPENDIX C

Utility Analyzer for Response Time Implemented in Java

The algorithm of the utility analyzer is defined in the Chapter three and it is implemented in Java language in the following figure.

```
QosAttribute qosAttr = new QosAttribute();  
int Br = 1; //1,2,4  
double e = 2.71828;  
double Kr = 100*(1 + Math.pow(e,Br)) / Math.pow(e,Br);  
double qosUtility = 0.0;  
double r = qosCollector.getBargainerQoSResponseTime ()/1000;  
qosUtility = (Kr * Math.pow(e,(-r +Br))) / (1 + Math.pow(e,(-r +Br)));  
qosAttr.setResponseTime(r);  
qosAttr.setUtilityValue(qosUtility);
```

Figure C.1 Implementation of The utility calculator (service Bargainer) in Java

Experiment Result Evaluating Utility Analyzer

Table C.1 Utility for Service Bargainer

# of Execution	Utility for Response Time	Utility for Throughput	The Overall Utility
1	21.78	12.322	17.997
2	80.209	75.115	78.171
3	78.877	72.889	76.482
4	28.769	16.482	23.854
5	80.871	76.188	78.998

Table C.2 Utility for Service Hawker

# of Execution	Utility for Response Time	Utility for Throughput	The Overall Utility
1	20.182	14.078	17.740
2	85.433	83.890	84.816
3	84.144	82.756	83.589
4	87.059	85.004	86.236
5	88.599	85.730	87.451

Table C.3 Utility for Service Rummage

# of Execution	Utility for Response Time	Utility for Throughput	The Overall Utility
1	23.11	9.152	17.527
2	87.659	17.017	59.402
3	86.710	14.814	57.952
4	88.630	19.650	61.038
5	88.286	18.669	60.439

APPENDIX D

This simple test is designed for semantic web service solution executed in WSMO environment, which is followed the execution flow shown in Figure 4.7. The test has no any QoS constraints involved. The scenario and test case result is shown in Appendix D. In this case, the goal aims to find a notebook of Apple Mac. The requirement is described as following:

1. Processor: Intel Duo Core Processor 2.0 GHz
2. Memory: at least 512 MB RAM
3. Hard Disk: at least 120 GB HDD
4. Piece: at least 2000 \$

The capability in the WSMO goal is defined as in the following Figure

```

capability GoalA2Capability
  nfp

  _"http://www.wsmo.org/goal/discovery/instancebased/mainElements" hasValue {"?x"}

endnfp

postcondition
  definedBy
    ( ?x[
      po#processorType hasValue po#intelCoreDuo,
      po#processorGHz hasValue ?procGhz,
      po#price hasValue ?price, po#color hasValue ?color,
      po#hddGB hasValue ?hddGB,
      po#memoryMB hasValue ?memMB] memberOf po#Notebook

      and ?procGhz = 2.0
      and ?memMB >= 512
      and ?hddGB >= 120
      and ?price > 2000
    ).

```

Figure D.1 Capability Definition for Notebook Purchasing

Also, by analyzing all the products from three vendors, we can easily know the product 00000010 offered by Bargainer can meet this requirement. The figure 4.9 and 4.10 shows the service discovery executed in SoapUI and output message in WSMX monitor respectively. And the result is same as our expectation.

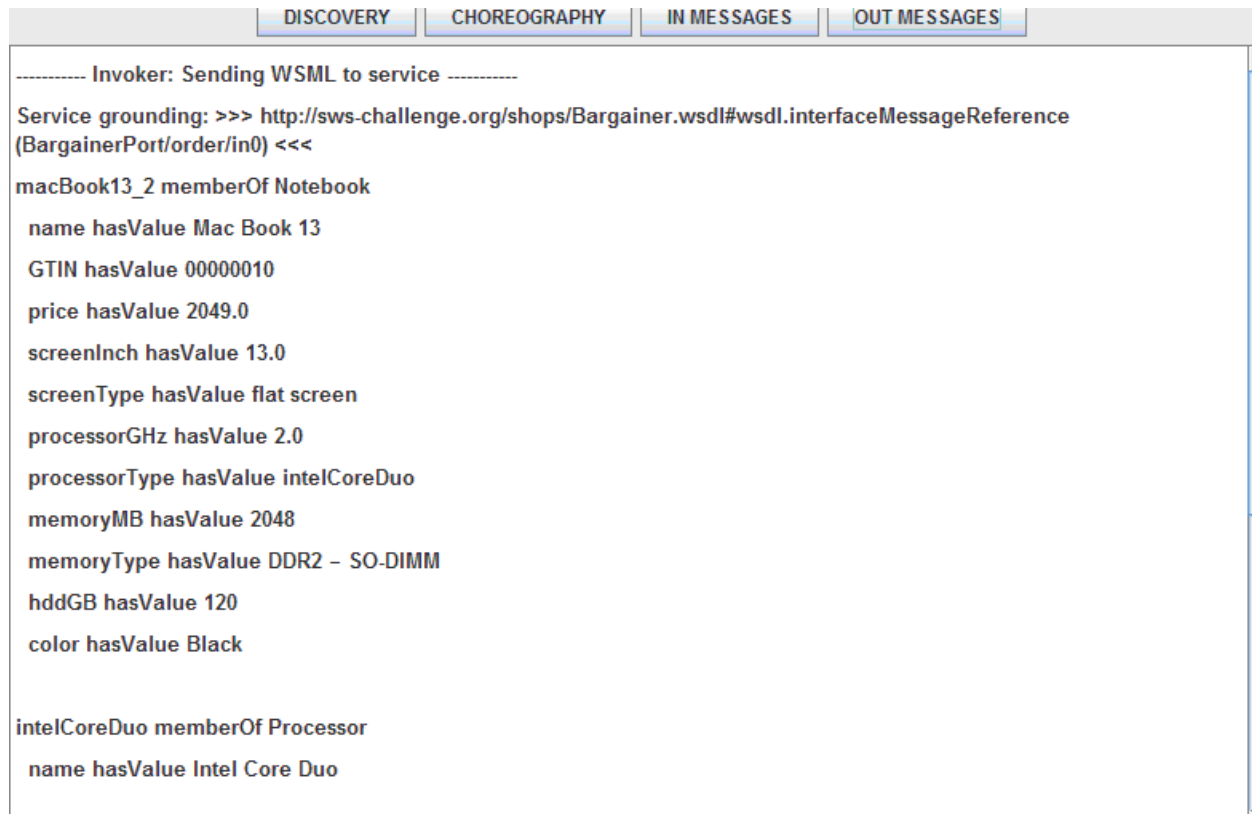


Figure D.2 Service Selection result Shown in WSMX Message Window.

REFERENCES

- [1] J.A.F. da Silva, N. das Chagas Mendonca, “Dynamic invocation of replicated Web services”, *In Proceedings of the WebMedia and LA-Web*, pp. 22 – 29, Brazil, 2004
- [2] Yijun Chen, Abdolreza Abhari, “An agent-based framework for dynamic web service selection”, *In Proceedings of the 2008 Spring simulation multiconference*, no. 6, Ottawa, Canada, 2008
- [3] E.M Maximilien and M.P. Singh, “A framework and ontology for dynamic Web services selection”, *International Journal of IEEE Internet Computing*, vol. 8, pp. 84 – 93, IBM Corp., USA, Sep. 2004
- [4] J. Kopecký and E. Simperl, “Semantic web service offer discovery for e-commerce”, *In Proceedings of the 10th international conference on Electronic commerce*, vol. 342, no. 29, Innsbruck, Austria, 2008
- [5] D. Bachlechner, “Toward a Semantic Web service technology roadmap”, *In Proceedings of Research Challenges in Information Science(RCIS 2008)*, pp. 17 – 28, Marrakech, Sep 26, 2008
- [6] X. Wang, M. Hauswirth, T. Vitvar, and M. Zaremba, “Semantic web services selection improved by application ontology with multiple concept relations”, *In Proceedings of the 2008 ACM symposium on Applied computing*, pp. 2237-2242, Ceara, Brazil, 2008

- [7] J. Cardoso, “Discovering Semantic Web Services with and without a Common Ontology Commitment”, *In Proceedings of the IEEE Services Computing Workshops*, pp. 183 – 190, Washington, DC, USA, 2006
- [8] G. Yeom and D. Min, ” Design and implementation of Web services QoS broker”, *In Proceedings of Next Generation Web Services Practices(NWeSP 2005)*, p.2, Seoul, South Korea, Aug 2005.
- [9] D.A. Menasce and V. Dubey, “Utility-based QoS Brokering in Service Oriented Architectures”, *In Proceedings of IEEE International Conference on Web Service (ICWS 2007)*, pp. 422 - 430, Salt Lake City, UT, 2007
- [10] M.A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, “A QoS broker based architecture for efficient Web services selection”, *In Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, vol.1, pp. 113 – 120, Que., Canada, 2005
- [11] C. J. Acuña and E. Marcos, “Modeling semantic web services: a case study”, *In Proceedings of the 6th international conference on Web engineering*, vol. 263, pp. 32 - 39, California, USA 2006
- [12] RDF Semantics, <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>, last visited Aug 29, 2010

- [13] Resource Description Framework (RDF): Concepts and Abstract Syntax, <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, last visited Aug 29, 2010
- [14] K. Kritikos and Dimitris Plexousakis, “OWL-Q for Semantic QoS-based Web Service”, *In Proceedings of Fifth European Conference on Web Services*, pp. 181 – 190, Halle, Germany , 2007
- [15] OWL Web Ontology Language Guide, <http://www.w3.org/TR/owl-guide/>, last visited Aug 29, 2010
- [16] I. Toma, D. Foxvog, and M.C. Jaeger, “Modeling QoS characteristics in WSMO”, *In Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, vol. 184, pp. 42 – 47, Melbourne, Australia, 2006
- [17] M. Hanafy and M. Fakhry “A Proposed Architecture for Learning Object Selection and Discovery Based on WSMO”, *In Proceedings of 4th International Conference on Next Generation Web Services Practices*, pp. 10 – 14, Seoul , Dec 2008
- [18] D. Bachlechner, “Toward a Semantic Web service technology roadmap”, *In Proceedings of Second International Conference on Research Challenges in Information Science*, pp. 17 – 28, Marrakech, 2008.

- [19] Web Services Description Language (WSDL) 1.1, <http://www.w3.org/TR/wsdl>, version 1.1, last visited Aug 30, 2010
- [20] The SWS Challenge, <http://sws-challenge.org>, last visited Aug 30, 2010
- [21] Simple Object Access Protocol (SOAP) 1.1, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>, version 1.1, last visited Aug 30, 2010
- [22] UDDI Version 3.0.2, http://www.uddi.org/pubs/uddi_v3.htm version 3.0.2, last visited Aug 30, 2010
- [23] D. Cooney and P. Roe. “Mobile Agents Make for Flexible Web Services”, *In Proceedings of the Ninth Australian World Wide Web Conference*, Australia, July 2003
- [24] C. Petrie, T. Margaria, H. Lausen, and Mi. Zaremba, “Semantic Web Services Challenge”, pp.13-27, Springer US, ISBN: 978-0-387-72495-9, 2009
- [25] T. Margaria, “The Semantic Web Services Challenge: Tackling Complexity at the Orchestration Level”, *In Proceedings of 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 183 – 189, Belfast, 2008

- [26] D. Fensel, H. Lausen, A. Polleres, J.d. Bruijn, M. Stollberg, D. Roman , and J. Domingue. “Enabling Semantic Web Services: The Web Service Modeling Ontology”, pp.63-81, Springer-Verlag New York, Secaucus, NJ, USA, ISBN: 3540345191, Nov, 2006
- [27] Web Service Modeling Ontology (WSMO), D2v1.3., <http://www.wsmo.org/TR/d2/v1.3/> , last visited Aug 30, 2010
- [28] S. Li and J. Zhou, "The WSMO-QoS Semantic Web Service Discovery Framework", *In Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE 2009)* , pp. 1-5, Wuhan, China. Dec, 2009.
- [29] J. Kopeck, D. Roman¹, M. Moran and D. Fensel, “Semantic Web Services Grounding”, *In Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, pp.127-127, Washington, DC, USA. 2006.
- [30] WSML, <http://www.wsmo.org/TR/d16/d16.1/v0.21/#sec:wsml-xml>, version 0.21, last visited Aug 30, 2010
- [31] W. Hodges, “The Blackwell Guide to Philosophical Logic”, pp. 10-31, Blackwell press, ISBN: 0631206930, 2001

- [32] L. S. Cauman, “First-Order Logic: an introduction”, pp.1-10, Berlin, Germany, ISBN: 0486683702, 1998
- [33] O. Shafiq, M. Moran, E. Cimpian, A. Mocan, Mi. Zaremba and D. Fense, “Investigating Semantic Web Service Execution Environments: A Comparison between WSMX and OWL-S Tools”, *In Proceedings of the Second International Conference on Internet and Web Applications and Services (ICIW '07)*, p. 31, Morne, June 2007.
- [34] M. Zaremba, T. Vitvar, M. Moran and T. Hasselwanter, “WSMX Discovery for SWS Challenge”, Digital Enterprise Research Institute, National University of Ireland, Galway, University of Innsbruck, Austria, Sep 01, 2008, available at <http://sws-challenge.org/workshops/2006-Athens/papers/DERI-sws-challenge-3.pdf>
- [35] G. Dai, X. Bai, and C. Zhao, “A Framework for Model Checking Web Service Compositions Based on BPEL4WS”, *In Proceedings of the IEEE International Conference on e-Business Engineering*, pp. 165 – 172, Hong Kong, China, 2007
- [36] M. Wang, Z. Du, Y. Chen, S. Zhu, and W. Zhu, “Dynamic Dataflow Driven Service Composition Mechanism for Astronomy Data Processing”, *In Proceedings of the IEEE International Conference on e-Business Engineering*, pp. 596 – 599, Hong Kong, China. 2007

- [37] Z. Chen, J. Ma, L. Song, and L. Lian. “An Efficient Approach to Web Services Discovery and Composition when Large Scale Services are Available”, *In Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages: 34 – 41, Guangzhou, Guangdong, China, 2006.
- [38] W.T. Tsai, C. Fan, Y. Chen, R. Paul, and J. Chung, “Architecture classification for SOA-based applications”, *In Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp.295-302, Tempe, AZ, USA. 2006
- [39] T. Yu, Y.Zhang, and K. Lin, “Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints”, *International Journal of ACM Transactions on the Web (TWEB)*, vol. 1, no. 6, New York, NY, USA, May 2007
- [40] Resource Description Framework (RDF), <http://www.w3.org/RDF/>, last visited Aug 30, 2010
- [41] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, “The Description Logic Handbook”, pp.1-5, Cambridge University Press, ISBN-10: 0521781760, 2003
- [42] M. Kerrigan, “Web service selection mechanisms in the Web Service Execution Environment (WSMX)”, *In Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1664 - 1668, Dijon, France 2006.

- [43] W.E. Walsh, G. Tesauro, J.O. Kephart, R. Das, “Utility functions in autonomic systems”, *In Proceedings of the First International Conference on Autonomic Computing*, pp. 70 - 77, NY, USA, 2004
- [44] R.D. van der Mei and H.B. Meeuwissen “Modelling End-to-end Quality-of-Service for Transaction-Based Services in Multi-Domain Environments”, *In Proceedings of IEEE International Conference on Web Services*, pp.3 – 462, Washington, DC, USA, 2006
- [45] D. Zuquim G. Garciz and M. Toledo, “Semantics-enriched QoS policies for web service interactions“, *In Proceedings of the 12th Brazilian Symposium on Multimedia and the web*, vol. 192, pp. 35 – 44, Natal, Rio Grande do Norte, Brazil, 2006
- [46] WSMO Studio Users Guide V.1.27, <http://www.wsmostudio.org/doc/wsmo-studio-ug.pdf>, last visited Aug 30, 2010
- [47] R. Karunamurthy, F. Khendek and R.H. Glitho, “A Novel Business Model for Web Service Composition”, *In Proceedings of the IEEE International Conference on Services Computing* , pp. 431 – 437, Montreal, Que, CA, 2006

