AUTOMATIC QUALITY OF SERVICE (QOS) EVALUATION FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRA ASKAROGLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2011

Approval of the thesis:

AUTOMATIC QUALITY OF SERVICE (QOS) EVALUATION FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

submitted by **EMRA ASKAROGLU** in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE in Computer Engineering Department, Middle East Technical University by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ________________

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** ________________

Assist. Prof. Dr. Pınar Şenkul
Supervisor, **Computer Engineering Department, METU** ________________

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Dept., METU ________________

Asst. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU ________________

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU ________________

Prof Dr. Nihan Kesim Çiçekli
Computer Engineering Dept., METU ________________

Assoc. Prof. Dr. Erdoğan Doğdu
Computer Engineering Dept., TOBB Uni. ________________

Date: ________________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: EMRA ASKAROGLU

Signature :

ABSTRACT

AUTOMATIC QUALITY OF SERVICE (QOS) EVALUATION FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY FRAMEWORK

Askaroglu, Emra
M.S., Department of Computer Engineering
Supervisor : Assist. Prof. Dr. Pınar Şenkul

June 2011, 61 pages

Web Service technology is one of the most rapidly developing contemporary technologies. Nowadays, Web Services are being used by a large number of projects and academic studies all over the world. As the use of Web service technology is increasing, it becomes harder to find the most suitable web service which meets the Quality of Service (QoS) as well as functional requirements of the user. In addition, quality of the web services (QoS) that take part in the software system becomes very important. In this thesis, we develop a method to track the QoS primitives of Web Services and an algorithm to automatically calculate QoS values for Web Services. The proposed method is realized within a domain specific web service discovery system, namely DSWSD-S, Domain Specific Web Service Discovery with Semantics. This system searches the Internet and finds web services that are related to a domain and calculates QoS values through some parameters. When a web service is queried, our system returns suitable web services with their QoS values. How to calculate, keep track of and store QoS values constitute the main part of this study.

Keywords: Web Service, Quality of Service (QoS), QoS Primitive, Automatic QoS Calcula-

tion, Web Service Discovery

ÖZ

ALANA ÖZGÜ WEB SERVİS KEŞİF SİSTEMLERİNDE OTOMATİK SERVİS KALİTESİ HESAPLANMA YÖNTEMİ

Askaroglu, Emra
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yar. Doç. Dr. Pınar Şenkul

Haziran 2011, 61 sayfa

Günümüzde en hızlı gelişen teknolojilerden biri Web Servislerdir. Web Servisler dünyada birçok proje ve araştırmada kullanılmaktadır. Web Servis teknolojileri kullanımı artarken, kullanıcıların fonksiyonel isteklerini en iyi şekilde karşılayacak kaliteli servisleri bulmaları giderek zorlaşmaktadır. Ayrıca istekleri karşılayan servislerin yanında bulunan servislerin kaliteleri özellikle büyük sistemler için daha da önemli hale gelmektedir. Bu tezde, web servis kalite parametrelerini takip edip inceleyecek ve bu parametrelerden servisin kalite puanını otomatik olarak hesaplayacak bir algoritma geliştirdik. Bu geliştirilen algoritma alana özel Web Servis keşif sistemi (DSWSD-S - Domain Specific Web Service Discovery-Semantic) içerisinde çalışacak şekilde geliştirildi. Bu sistem internet üzerindeki bir alan ile ilişkili web servisleri bularak belli parametrelere göre bu web servislerin kalitelerini hesaplamaktadır. Kullanılmak üzere bir web servis bulunmak istendiğinde, sistemimiz kullanıcı tarafından girilen anahtar kelimeye uygun web servisleri kalite puanları ile birlikte kullanıcıya yansıtır. Servis ile ilgili bilgileri sistemde tutma yöntemimiz ve kalite hesabını nasıl ele aldığımız bu çalışmanın en önemli bölümlerini oluşturmaktadır.

Anahtar Kelimeler: Web Servis, Web Servis Kalitesi, Servis Kalite Parametreleri, Otomatik Servis Kalitesi Hesaplama, Web Servis Keşfi

To my family and my friends

ACKNOWLEDGMENTS

I would like to thank to Pınar Şenkul for her supervision and guidance through the development of this thesis.

I would like to thank to my family and my supportive friends for their belief in me.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

For building large software systems that include distributed parts, the use of Web Services is one of the most preferable techniques [1]. For the potential users, there are large number of public services on the web. These services provide a variety of functionality, while some of them may provide the same result.

With the increasing number of published web services, searching and selecting a web service is becoming a complex problem. First of all, user needs to spend much time to be able to find a web service that functionally matches the expectations without using an auxiliary tool. While searching a service, beside the difficulty of keyword selection, it is not easy to select the best suitable service from the result set. Random selection from the returned set is a high risk to take for large software systems. The other problem is that, even if the user finds a service that meets the expectations, its not easy to decide that the chosen web service has high enough Quality of Service (QoS) value to use in a software system. QoS for a web service can be calculated by considering the items below;

- how rapid the web service will response after invocation,
- how many transactions the service can handle cuncurrently,
- whether the service will be available and reliable during the lifetime of the project.

Besides, if there are two web services that provide same functionality, a user should have the chance to prefer a free service over another one with price.

In the literature, there are many studies on QoS evaluation for Web Services [1, 2, 3, 4, 5, 6, 7, 8]. Most of these studies rely on user evaluation for calculating QoS values of Web services. The others that perform QoS calculation automatically or semi-automatically have several drawbacks. One of them is the lack of considering aging factor for older parameter values. Another one is the indefinite QoS value ranges, which makes it harder to make QoS comparison.

In this thesis, we present a method to automatically calculate, keep track of and store QoS values for Web services. QoS value is calculated through tracking the values for ”Response Time”, ”Availability”, ”Reliability”, ”Throughput” and ”Price” parameters of services over time. These attributes are the most frequently used parameters in the literature that constitute QoS values [2, 3, 4, 5, 9]. Since these values are checked and recorded periodically, it is possible to calculate QoS values automatically, without any need for user rating. Another important feature incorporated in QoS calculation is that the old values contribute to the overall QoS value less than the newer ones. Hence, a service with increasing quality becomes more favorable.

Calculation of the QoS values can be done in different ways. Usually the calculation is based on the assumption that all of these parameters are of the same weight. However, users may want to use these parameters with different weights. For example, a user may request a service whose response time is very short, on the other hand, whose price is not important. In this situation, the weights of the parameters should change and QoS value calculation should be done accordingly. The proposed method provides this functionality.

The value calculation algorithm normalizes the overall value in [0-1] range. This normalization facilitates the comparison of the services considerably. On the contrary, the indefinite value range for a web service would not be much helpful for comparing different web services. These indefinite numbers can be used only in web service comparison.

1.1 Motivation

Web services becomes one of the most preferable techniques in software society. Web services help developers build large software systems so quickly and easily. Therefore, the main motivation of this thesis is to help developers find web services that meets their expectations with high quality. While doing this, we found out that the QoS calculation methods proposed in the literature have certain drawbacks.

The work that is proposed in this thesis is actually a part of a larger web service discovery system. The system searches the web, categorizes web services according to ontologies and presents users an easy way to find required web services. There are several previous studies on QoS calcuation. In this work, we aim to overcome the shortcomings of the previous studies for automatic QoS calculation.

Quality of web service may be affected from several parameters. Within the scope of this work, five criteria, which are response time, availability, reliability, throughput and price are considered but the proposed model can be extended with other criteria easily.

1.2 Contributions

The main contributions of this thesis are as follows:

- Tracking method for Quality of Web Service parameters is presented. While tracking the QoS parameters, how to get the parameter values for each web service invocation is described in detailed.
- A novel QoS calculation algorithm is presented. The algorithm is described step by step.
- Comparison between QoS calculation approaches is described. The advantages and disadvantages of the proposed QoS calculation algorithm is discussed.
- The proposed work is realized within a Web Service discovery framework. This framework and interaction of the proposed work with the framework are presented.

1.3 Thesis Organization

This thesis is organized as follows:

Chapter 2 presents the related work on QoS calculation and web service selections.

In Chapter 3, the overall design of domain specific web service discovery system is presented. The graphical user interface, database operations, calculation of quality and relations between these parts are described in detail. Also, Quality of Service and its influences on the main system are described in the rest of Chapter 3.

In Chapter 4, tracking quality of web service parameters and getting the parameter values for each invocation are described in detail. The chapter starts with the information on invocation of a web service and it continues with the data collection algorithms of QoS parameters (Response Time, Availability, Reliability and Throughput) for web services.

In Chapter 5, the proposed QoS calculation algorithm is presented step by step. The approach is discussed with different point of views.

In Chapter 6, case study and evaluations of the system and proposed QoS calculation algorithm are displayed. In addition, a comparison between the proposed approach and previous approaches is presented.

Conclusion and future work are discussed in Chapter 7.

CHAPTER 2

RELATED WORK

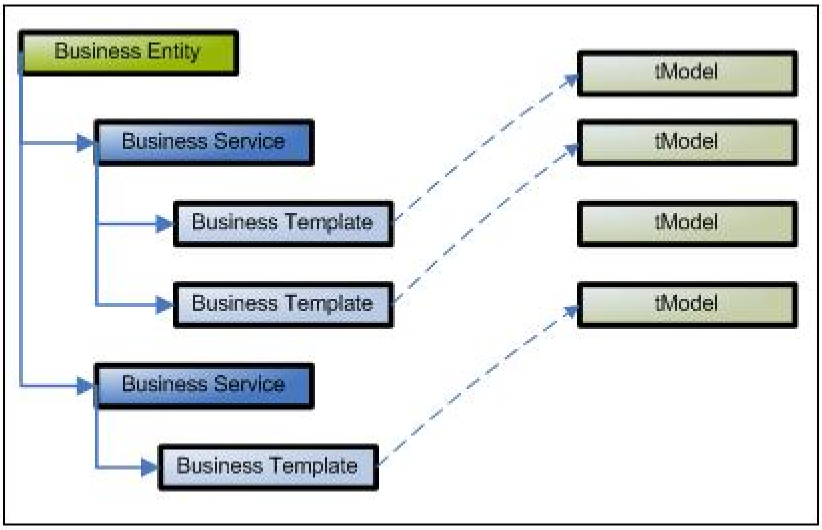
QoS for Web Services topic is studied by several researchers in the literature. The studies focus on different aspects and provide various solutions. In Section 2.1, UDDI extended models involving user feedbacks for QoS are presented. In Section 2.2, related Quality of Service research and use of QoS parameters are presented. In Section 2.3, QoS calculation algorithms of previous studies are given.

2.1 UDDI Extended Web Service Selection Models

UDDI (Universal Description, Discovery and Integration) is a directory that includes web service information. UDDI is an XML-based and platform-dependent structure. Web services publish their detailed information in the UDDI registry for public users [2]. For example, Ministry of Health or pharmacies can publish costs of medicines or detailed information about medicines in a UDDI registry. Therefore, public users can get information about medicines that meets their requirements.

UDDI registry includes four primary datatypes, which are businessEntity, businessService, bindingTemplate, and tModel. The businessEntity structure contains contact information, industry categories, business identifiers and a list of published services of the company. The businessService contains information about an individual web service. Type of the web service, taxonomical categories it belongs to and the way of binding to the web service can be given with the businessService data structure. The bindingTemplate structure contains technical description about a web service. The last UDDI registry datatype is tModel which represents various other information about web services [3][4]. The relation between these datatypes can be given as in Figure 2.1.

Figure 2.1: UDDI Registry Primary Datatypes [3]



As QoS becomes more important for web service to meet the users expectations, many studies focus on extending the current UDDI for including QoS values. The conventional UDDI does not support non-functional requirements for web services [5]. In Figure 2.2, current UDDI model is represented. In [6], the current UDDI model is extended with a new model including a new role called Certifier and the registry structure differs from conventional UDDI registry by having associated QoS registered in the repository. In Figure 2.3, the extended UDDI model is represented. Before registration of a web service, the Certifier verifies the claims of QoS for a web service. Therefore users can verify the QoS claims of web service before invocation and can select web services according to QoS information [6].

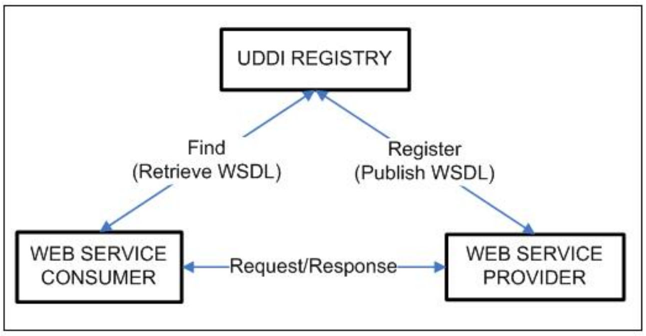


Figure 2.2: Current UDDI Model [6]

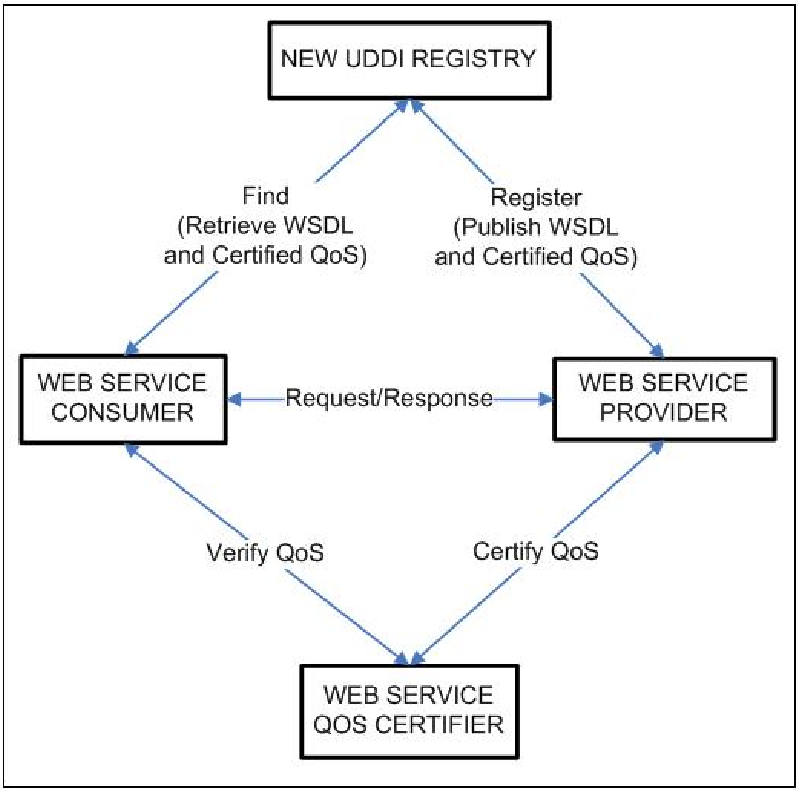


Figure 2.3: A new Web Services Registration and Discovery Model [6]

In [5], E. Michael Maximilien and Munindar P. Singh propose to extend web service selection architecture by adding an agent framework. The main purpose of the framework called Web Services Agent Framework (WSAF), is to keep the QoS parameter values and user feedbacks. WSAF creates an agent for each web service and each agent does the same operations for individual web services. In Figure 2.4, the proposed architecture can be seen.

The detailed usage of the agents and QoS details are proposed in their paper. In addition, they built a simulation for WSAF and evaluated a scenario involving agent usage and various QoS variables.

In [7], a QoS-based web service discovery with reputation-enhanced model is proposed. In Figure 2.5, the architecture of the proposed model can be seen. In this model, UDDI is extended by a reputation manager, discovery agent, rating database and QoS in the UDDI registry. Reputation manager gets the user feedbacks, calculates reputation scores and records the scores into the rating database. Discovery agent receives requests by a SOAP message from the service consumer, finds web services that meet the consumer's requirements and

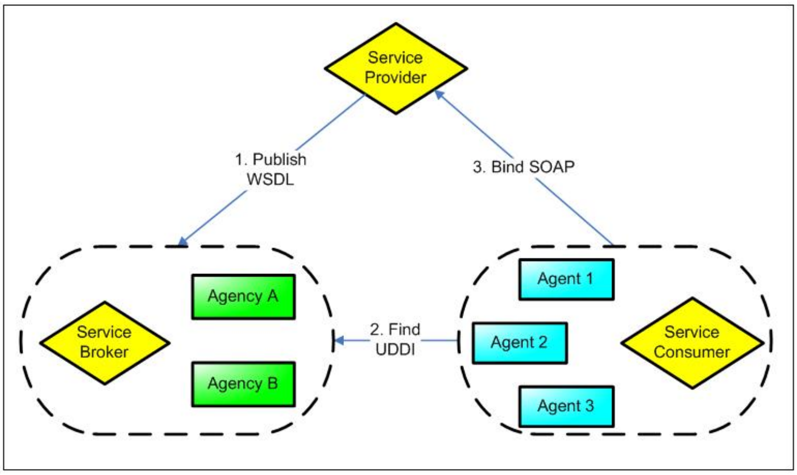


Figure 2.4: Agents and Agencies in a Service-Oriented Architecture [5]

returns the found web service list. Discovery agent applies service matching, ranking and selection algorithm while finding the web services that meet the users expectations.

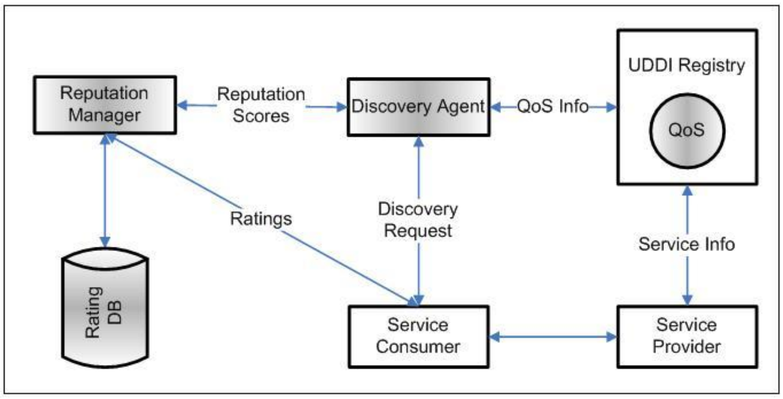


Figure 2.5: Reputation-Enhanced Web Service Discovery Model [7]

In this model, reputation manager calculates the reputation scores by applying the equation [7] given as 2.1;

$$U = \sum_{i=1}^{N} S_i \lambda^{d_i} \tag{2.1}$$

where N is the number of ratings for each service, S_i the rating of the ith service, λ is the inclusion factor and d_i is the age of the ith service in days [7].

Anna Averbak et al [8], propose a model to improve the web service selection process by using user feedbacks. Within their model, there is a matchmaker, which searches web services as response for users requests. The model, also allows users to give a rating point for each service by considering how relevant or appropriate the returned web service is for their requests. The matchmaker uses the user ratings when a user calls the proposed system with similar requests.

In [9], an architecture for agent-based web service selection is proposed. In this work, the architecture in [5] is extended by involving Verifier/Certifier given as in Figure 2.6.

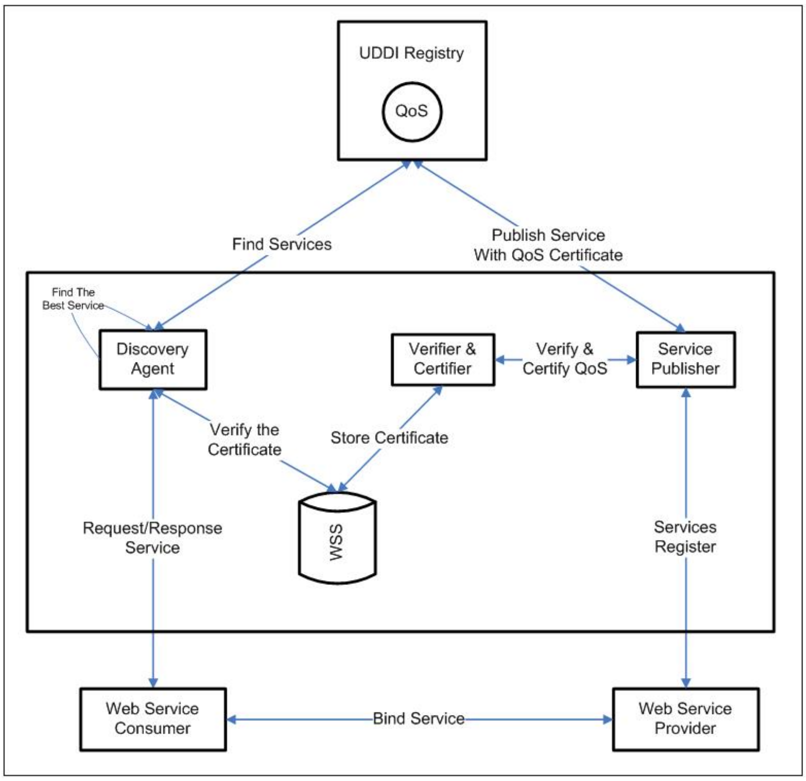


Figure 2.6: Architecture for WSB [9]

In this model, the service matching, ranking and selection algorithm in [5] is used with small changes. In addition, the UDDI tModel is used for QoS parameters as in [5].

2.2 Quality of Service Studies

QoS value calculation without extending UDDI architecture is the other important research area in the literature. QoS value calculation parameters ”Response Time”, ”Availability”,

”Reliability”, ”Throughput” and ”Price” are the mostly used parameters in several studies [6, 9, 10, 11, 12, 13]. The main idea for QoS calculation is tracking QoS parameters with web service invocations and selecting web services depending on the QoS parameters. The studies use different approaches for web service selection after tracking QoS parameters.

Liangzhao Zeng et al in [10], use QoS parameters for web service composition. In the paper, they define *execution paths* and *execution plans* for their goal. *”An execution path of a statechart is a sequence of states [t_1,t_2, ..t_n], such that t_1 is the initial state, t_n is the final state and for every t_i (1 < i < n) ...”*. And ”A set of pairs $p = < t_i, s_{i1} >, < t_2, s_{i2} >, ..., < t_N, s_{iN} >$ is an execution plan of an execution path...”

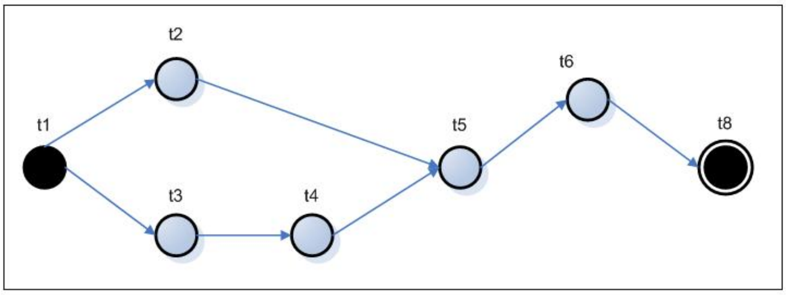


Figure 2.7: Execution Path 1 [10]

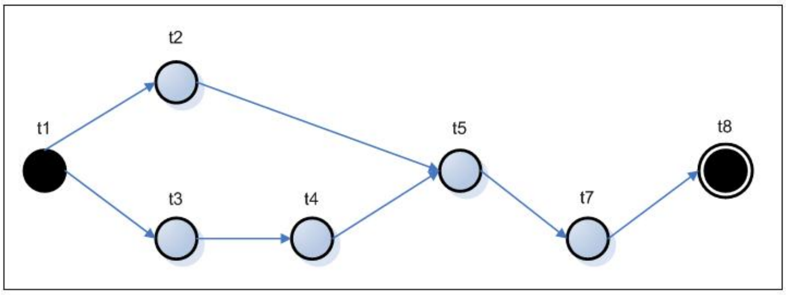


Figure 2.8: Execution Path 2 [10]

In the paper, execution price, execution duration, reputation, reliability and availability criterias are used as QoS parameters. Figures 2.7 and 2.8 can be used as examples for execution paths. In Table 2.1, the aggregation function of the used QoS parameters are given. The details about the parameters and the functions are given in the paper. In the experiments part of the paper, they aim to select an optimal execution plan depending on the given aggregation functions.

Table 2.1: Aggregation Functions for Computing the QoS of Execution Plans [10]

Criteria	Aggreation Function
Price	$Q_{price}(p) = \sum_{i=1}^{N} q_{price}(s_i, op_i)$
Duration	$Q_{du}(p) = CPA(q_{du}(s_1, op_1), ..., q_{du}(s_N, op_N))$
Reputation	$Q_{rep}(p) = \frac{1}{N} \sum_{i=1}^{N} q_{rep}(s_i)$
Reliability	$Q_{rep}(p) = \prod_{i=1}^{N} e^{q_{rel}(s_i)*z_i}$
Availability	$Q_{rep}(p) = \prod_{i=1}^{N} e^{q_{av}(s_i)*z_i}$

In the literature, there are several studies on QoS ontologies and query languages that aim to be helpful for non-expert users and select web services that meet users requirements properly. QoSOnt [14], WS-QoS [15] and OWL-Q [16] are proposed ontologies for semantic web service selection. In [25], a new QoS ontology is presented. In addition, there are also syntactic-based QoS query languages such as [17][18][19].

In [11], a QoS query language proposed. In addition, a user-friendly user interface is proposed with query language. They claim that the proposed query language is easy to use.

The proposed query consists of six parts, which are query ID, user ID, submission time, time constraints, QoS constraints and data source. Query ID, user ID and submission time are used as logging information. In time constraints part, there are start date, end date, duration and frequency of the usage. QoS constraints include constraints related to QoS parameters. Data source is used to define data source to process the service selection. An example segment of the proposed language is given in Figure 2.9.

In the paper, they proposed a user interface to help user specify QoS constraints. The specification can be done with four steps. In first step, user can select QoS attributes to use in service seletion from provided QoS attributes as in Figure 2.10. In the second step, user defines the QoS requirements as in Figure 2.11. In this paper, in addition to user-defined digit inputs, fuzzy inputs can be used for QoS requirements. In third step, the order of QoS attributes can be defined as in Figure 2.12. When user selects QoS attributes, s/he can set a priority value for the selected QoS attribute. The priority of unselected QoS attributes are considered as zero. Finally, the last step is defining the time constraints as in Figure 2.13. User may want to use web service at the time of service selection. Besides, user may want to use web service in the future. Therefore in time constraints, s/he can set date/time value as one of future time.

```xml
<?xml version="1.0" encoding="utf-8"?>
<QoSQuery>
 <qID>1</qID>
 <uID>10</uID>
 <sbTime>01/10/2010 16:40:40 ET</sbTime>
 <timeConstraints>
   <startDate>02/17/2010</startDate>
   <endDate>07/17/2010</endDate>
   <frequencyOfUsage>
     <value>3</value>
     <unit>week</unit>
   </frequencyOfUsage>
   <durationOfUsage>
     <value>2</value>
     <unit>hours</unit>
   </durationOfUsage>
 </timeConstraints>
 <QoSConstraints>
  <QoSConstraint>
   <name>price</name>
   <type>numeric</type>
   <unit>US dollar</unit>
   <tendency>negative</tendency>
   <preference>2</preference>
   < relaxationOrder>0</relaxationOrder>
   <weight>0.3</weight>
   <values type="range" >
    <from>100</from>
    <to>150</to>
   </values>
  </QoSConstraint>
……
  <QoSConstraint>
   <name>reliability</name>
   <type>numeric</type>
   <unit>%</unit>
   <tendency>positive</tendency>
   <preference>3</preference>
   < relaxation>1</relaxation>
   <weight>0.1</weight>
   <values type="fuzzy" >
    <value>good</value>
   </values>
  </QoSConstraint>
 </QoSConstraints>
 <dataSource>provider</ dataSource >
</QoSQuery>
```

Figure 2.9: A Segment of a Sample QQL Query [11]

Figure 2.10: Step 1 of Query Formulation [11]

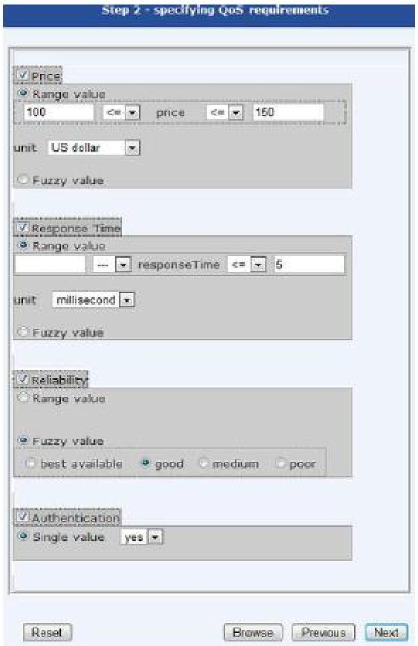


Figure 2.11: Step 2 of Query Formulation [11]

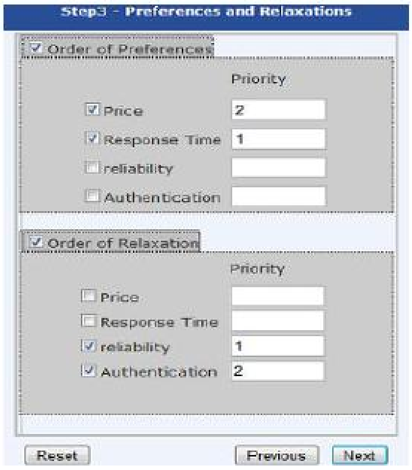


Figure 2.12: Step 3 of Query Formulation [11]

2.3 QoS Calculation Algorithms

There are also studies about QoS calculation considering non-functional properties of Web Services. The algorithm in [20] considers four QoS parameters: Availability, Reliability, Execution Time and Execution Price. In the paper, QoS parameters are categorized as negative QoS factors and positive QoS factors. If the higher value of QoS parameter lowers quality,

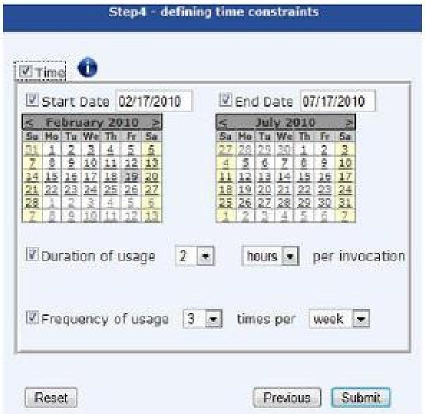


Figure 2.13: Step 4 of Query Formulation [11]

then the QoS parameter is a member of negative QoS factors. On the other hand, if the higher the value of QoS parameter leads to higher quality, then the QoS parameter is a member of positive QoS factors.

In the paper, values of each QoS paramaters are calculated by different equations and the overall QoS value has its own equation itself. The equations can be given step by step as;

For Availability($q_a(S)$):

$$q_a(S) = \frac{N_{AS}}{N_{AT}} \tag{2.2}$$

where N_{AS} is the number of times the service can be accessed and N_{AT} is the number of times that the service is invoked.

For Execution Time($q_{et}(S)$):

$$q_{et}(S) = T_{RT} - T_{ST} \tag{2.3}$$

where T_{RT} is the moment that the service returns result and T_{ST} is the moment when the service is invoked.

For Reliability($q_r(S)$):

$$q_r(S) = \frac{N_{RS}}{N_{RT}} \tag{2.4}$$

where N_{RS} is the number of times the service returns expected result in a time span and N_{RT} is the number of service invocation times in the same time span.

Execution Price value is used as itself. The value of price is used in the QoS value calculation algorithm with no change.

The QoS value calculation equation is given as;

$$\sum W_m * q_i + \sum W_n * \frac{1}{q_j} \tag{2.5}$$

where W_m and W_n are weights of QoS parameters; q_i is a negative QoS parameter and q_j is positive QoS parameter.

There are two important points that are not considered by this algorithm. The first important thing is the age of the QoS parameter values. The age of the parameter values are important to follow the evoluation of the service. Web services are mostly modified by their developers so a web service may become better or worse than earlier version. Therefore the latest QoS parameter values should have more weight than the oldest ones. Another important point is that QoS values are not generated in a definite range in this algorithm. Therefore, it is very hard to interpret such a value by itself. It can be only helpful for comparison when values for two web services are available. In this thesis, we considered these two important points while implementing our QoS calculation algorithm.

In [24], a framework for quality of web service evaluation is designed. In this framework, functional candidate services are chosen at first. Then, the chosen services are filtered with some constraints and the candidate services are found. After that, QoS values are calculated for each candidate service. The architecture can be given as in Figure 2.14.

The architecture is divided in two steps. The first step is finding services that meets the user's functional requirements. The other step is evaluating quality result of each service. In this work, used QoS parameters and calculation algorithm are not given in detail. The equation in 2.6 is used for the final quality value.

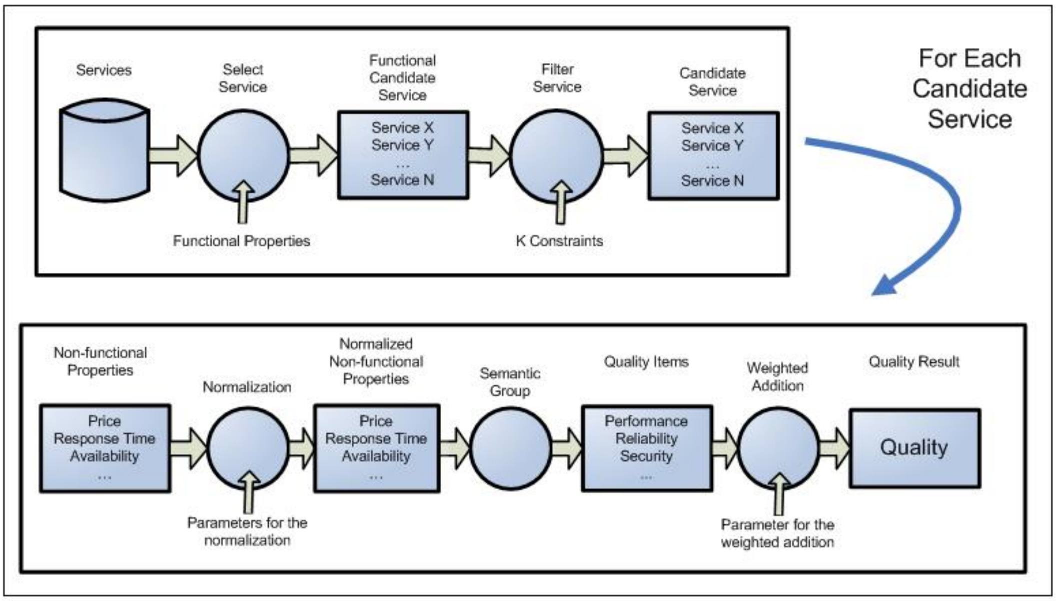


Figure 2.14: General View of All the Steps to Select Best Service[24]

$$Q_{ws} = \sum_{i=0}^{N} a_i q_i \tag{2.6}$$

where q_i is i_{th} quality item value, a_i is the weight of i_{th} quality item and $\sum a_i$ is equal to 1.

This algorithm normalizes the QoS result of each service in range [0-1]. However this algorithm does not consider the ages of QoS parameter values for each iteration.

CHAPTER 3

DOMAIN-SPECIFIC WEB SERVICE DISCOVERER WITH SEMANTICS AND QOS HANDLING IN THIS SYSTEM

The thesis is a part of project described in [21][23]. The project aims to develop a domain-specific web service discovery system with semantics (DSWSD-S). In this chapter, DSWSD-S system is described. In Chapter 3.1, overall design of the proposed system is presented. Chapter 3.2 is about graphical user interface of the system which plays an important role for the interaction between the main system and users. Finally, in Chapter 3.3, the place of Quality of Service in DSWSD-S is presented.

3.1 Overall Design of the System

The main idea of the DSWSD-S system is to search web services quickly, to keep their up-to-date status and evaluate their quality to respond users requests appropriately. While doing this, web services are annotated with the ontology of the domain specific discovery system. The system aims to provide following facilities, as well:

- Providing semantic queries for service search
- Ability to deal with high number of services, providing up-to-date information about web services
- Fast inclusion of recently published services

Finding the most appropriate web service that meets users requirements from the published web services, which are registered to different service registries becomes a difficult problem as the number of services increases. In addition, there are web services which are not registered to any of the registries. The aim of the DSWSD-S is searching both registries and the sites and presenting a a common graphical interface for requesters.

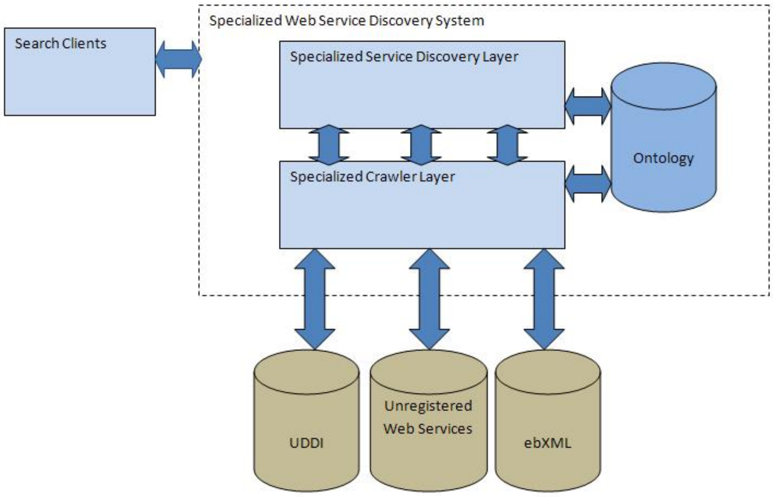


Figure 3.1: The architecture of DSWSD-S [21]

The proposed system consists of two layers as in Figure 3.1, which are domain-specific crawler layer and domain-specific service discovery layer.

In this system, each domain specific service discovery node has its own crawler. The process in domain specific crawler layer starts with web service address acqusition. After that, the context of a web service is downloaded and examined whether it is related to its own ontology. Next, crawler tries to validate web services by calling them with appropriate input parameters. Crawler passes validated web services to extraction module. Extraction module makes semantic service annotation. Finally, crawler adds verified web services into the service database as shown in Figure 3.2.

Domain-specific discovery layer has a graphical user interface to query the database. When it provides the user interface, user can select search ontology, enter keyword to search and set QoS constraints for required web services. After specifying inputs, discovery layer passes provided inputs to syntactic and semantic matching engine. The engine checks the inputs and decides whether the requirements are related to its own ontology or not. If it is related to its own ontology it uses its own database, otherwise it passes the request to DSWSD-S peers. Finally, the discovery layer finds the web services that meet users requirements from

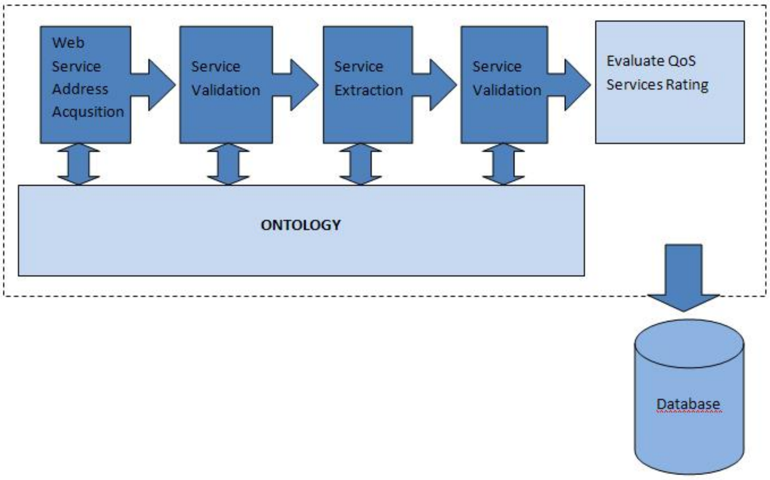


Figure 3.2: Achitecture of Crawler Layer [21]

the database, evaluates their QoS values and returns a list that is sorted by QoS values to user.

3.2 Graphical User Interface

As mentioned in Section 3.1, domain-specific discovery layer provides a user interface. The user interface has a place between users and the down-level of the proposed architecture. In general, the user interface takes the search inputs from user, pass the inputs to search engine and lists the web services compatible with the inputs to the user.

The user interface lets user set the ontology that s/he want to search in. After selecting the ontology user should enter keyword(s) to search in the selected ontology as in the Figure 3.3.

The user can also set QoS constraints to make an advanced search. While making advanced search user can use either default QoS constraints or custom QoS constraints. If user selects to use default QoS constraints the inputs of QoS constraints become disabled as in Figure 3.4. If user wants to set custom weights of QoS parameters inputs become enabled as in Figure 3.5

As in Figure 3.5, user can set response time, availability, reliability, throughput and price values of web service as QoS constraints. The first four QoS constraints (response time, availability, reliability and throughput) are entered as a number between 0 and 100. These

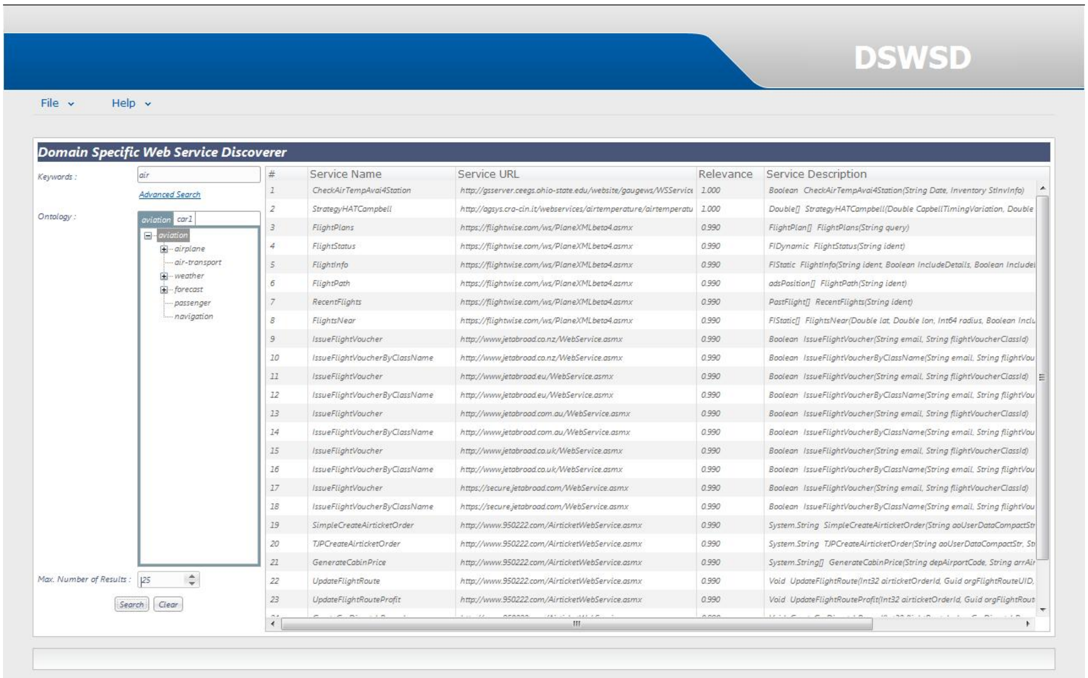


Figure 3.3: User Interface Without QoS Constraints

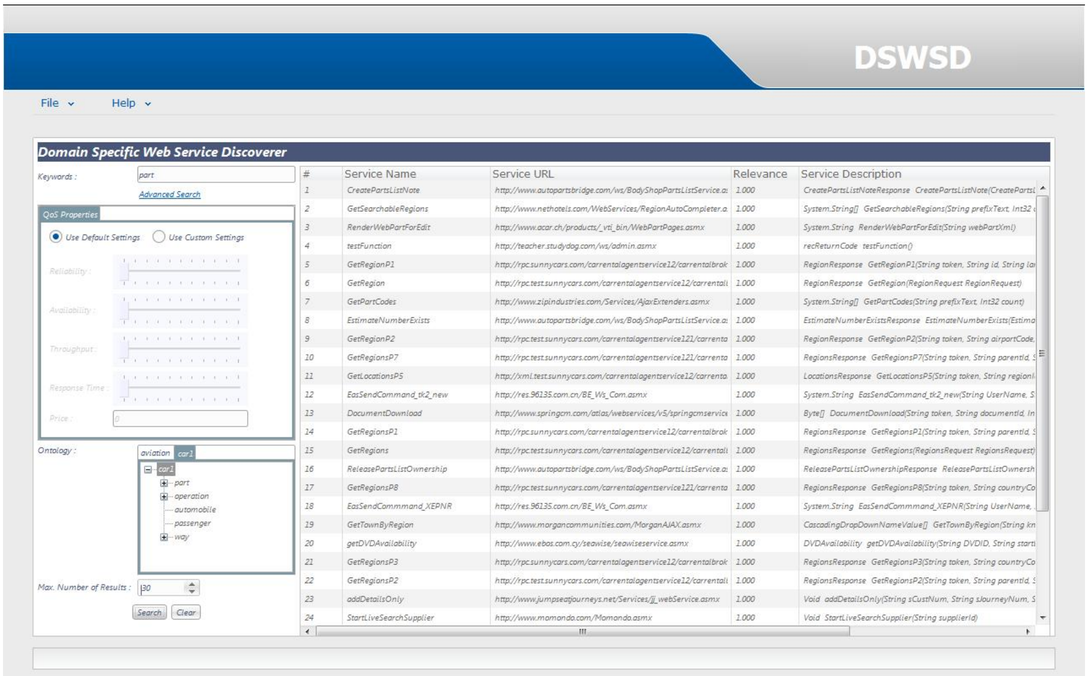


Figure 3.4: User Interface with Default QoS Constraints

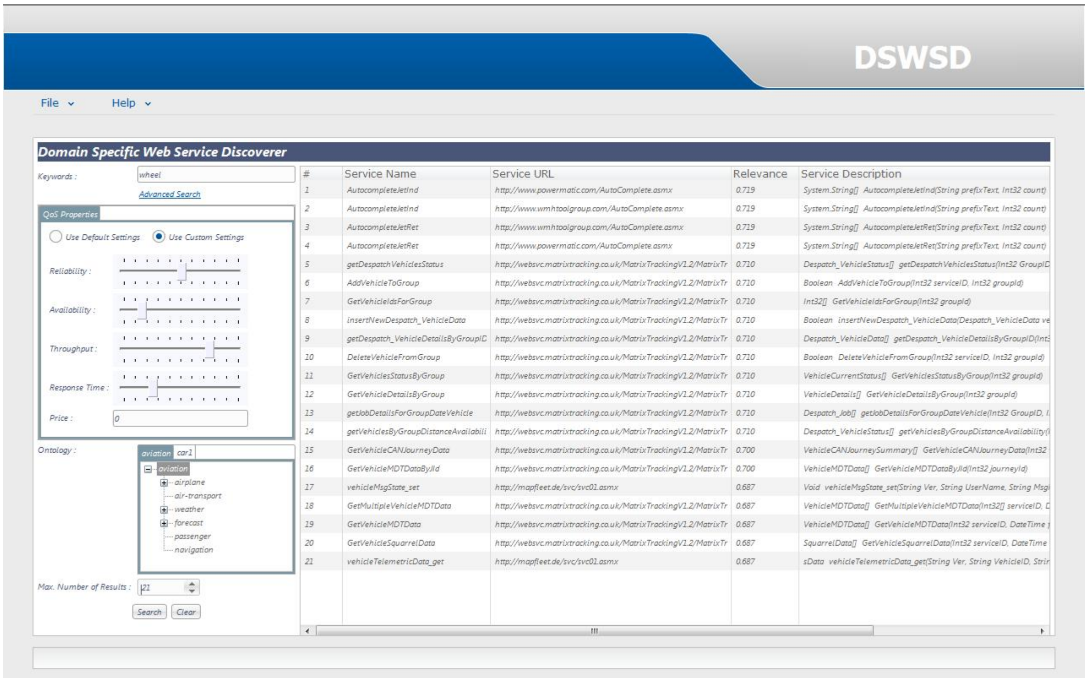


Figure 3.5: User Interface with Custom QoS Constraints

values are used as the weight of the related QoS parameters. The average of the inputs are used as threshold for result list. For example, if user sets 60 for response time, 80 for availability, 100 for reliability and 80 for throughput, then the entered values are taken as weight of the related QoS parameter and their average, which is 80, is taken as threshold which means result list would be eliminated from web services which have QoS value less than 0.8 (80 / 100).

There is also price value which is taken as a QoS constraint. User may want to set a maximum price value that s/he can afford for a web service. Price value is used to eliminate the result list from the web services whose prices are higher than the entered price value. If user sets price value as 0, then only free web services are listed on the user interface.

3.3 Quality of Service in DSWSD-S

After URL acquisition and recording web services into database, a new engine starts working in order to evaluate qualities of the discovered services. The engine that is proposed in this thesis gets the QoS parameter values of each web service recorded into service database. Quality of Service has an important role in DSWSD-S system. In general, DSWSD-S can search registries and unregistered web sites and finds web services semantically. In addition, with the usage of user interface DSWSD-S can give response to users requests. The functionality of the system is further extended with QoS supported service query and automatic QoS evaluation, so that higher quality services can be highlighted among the ones that all match syntactic and semantic queries.

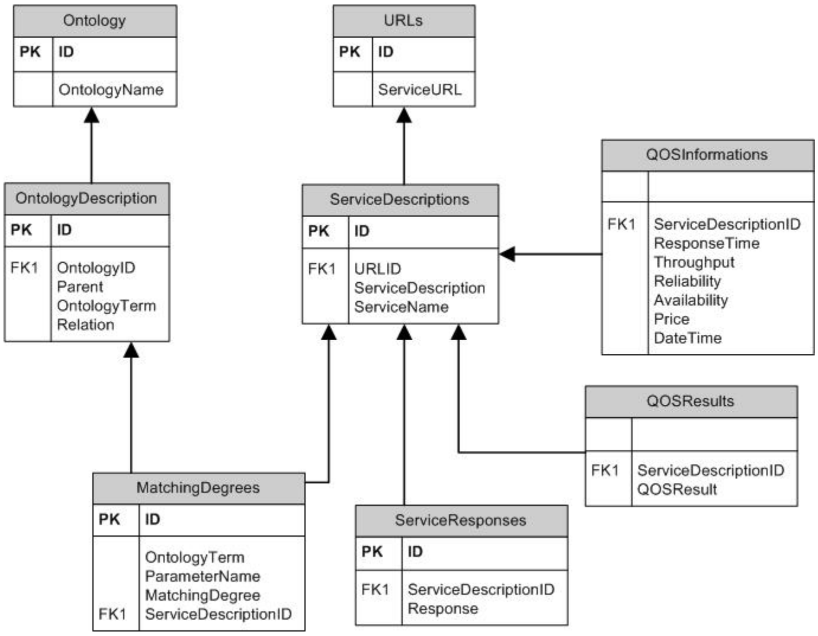


Figure 3.6: Database Architecture

In DSWSD-S system we used the database architecture as shown in Figure 3.6. We keep ontologies, service URLs, service descriptions, QoS parameter values and service responses in the database. In QoSResults table, calculated QoS values by the same weights of QoS parameters are recorded.

As a part of this thesis, a web service that handles connection with the database is implemented. In the web service, there exists operations such as inserting a data to a table

- insertURL(String URL)
- insertServiceDescription(ServiceDescriptionClass serviceDescription)
- insertOntology(Ontology ontology)
- insertQOSInformation(QoSInformations QoSParameters)

- ...

getting data from a table

- getURLs()
- getURL(int URLID)
- getQoSValue(int serviceDescriptionID)
- ...

calculating QoS of services and getting QoS values of the services

- calculateQoSResult(int serviceDescriptionID)
- getQoSResults(List ServiceDescription)

As shown in Figure 3.6, the table holds QoS parameters for each web service with their inserted date time (QoSInformations). QoS values are kept in a different table (QoSResults). During a certain period of time, all web services in this database (table ServiceDescriptions) are invoked by a program and their QoS parameter results are taken and saved in QoSInformations table. Meanwhile we can see the evolution of each web service according to these information.

The relation between user interface, QoS and database are presented in Figure 3.7.

As shown in Figure 3.7, GUI calculates matching degree of keyword with selected ontology. After that, it gets services with their matching degrees from database. After applying a custom threshold to the returned list, it uses QoS engine to get the list ordered by their QoS values depending on QoS constraints entered by user.

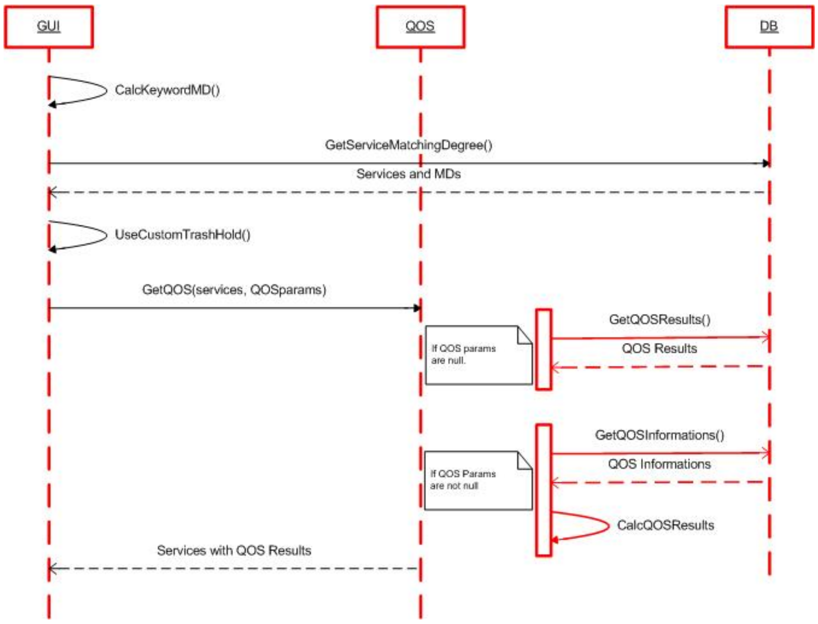


Figure 3.7: Relation between GUI, QoS and DB

CHAPTER 4

TRACKING QUALITY OF SERVICE PARAMETER VALUES

In DSWSD-S system, crawlers search the Web for URLs and service descriptions of web services. In each service description file, there may be several web services. All of the discovered web services are recorded into the database with their service names, parameters and URLs.

In this chapter, invocation of a web service and tracking QoS parameter values for each web service are described.

4.1 Invocation of a Web Service

In the task of tracking QoS parameter values of web services and calculating QoS values of each web service starts with getting web service descriptions from the database. After that, for each web service, URL of service is taken from the database in a loop. ”index.asmx” files of the specified URLs contain descriptions of web services. Therefore, in the next step the program downloads the ”index.asmx” files of URLs to local storage one by one.

After downloading whole content of a URL, the invocation process begins. There may be more than one service in each wsdl file. Each web service is recorded in the database separately. Therefore, the services in the downloaded wsdl file are taken at first. The next step is to generate parameters of each web service. Service descriptions of web services contain parameter information. Therefore, with the given information, information the program generates the values of the parameters as shown in Table 4.1.

The values for primitive types are generated by following some basic rules. As in Table 4.1,

Table 4.1: Default Values of Web Service Parameters

Type of Parameter	Value
Boolean	true
Int64 or Int32 or Int16 or Byte	1
Double or Decimal or Single	1.0
String	”1”
DateTime	Current date time
Enum	First value of the type of the parameter
Class	execute the process for each property of object
Array	execute the process for each object in the array

the program sets ”true” value to boolean parameter, 1 to integer parameter, 1.0 to double, decimal or single parameters. Since some of services get a parameters as ”String” type but convert the parameters to integer inside of the service. Therefore the program sets ”1” to ”String” type parameters. The program sets the current system time to ”DateTime” parameters and the first value of the parameter if the parameter is an enumaration. If a parameter is an object of a Class, then the program executes the process for each property of object. In addition, if a parameter is an Array, then the program executes the same process for each value of the array again.

After filling values of parameters, the program invokes the web service as last step of the process. The whole process is given in the Figure 4.1;

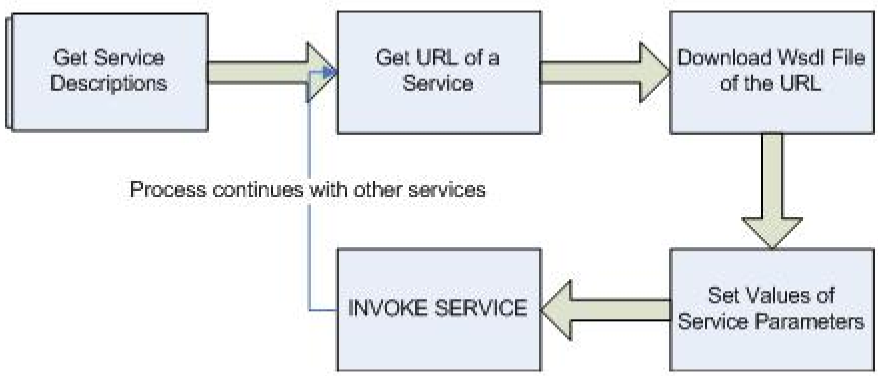


Figure 4.1: Tracking Algorithm

This process is illustrated with an example as follows;

1. Getting service descriptions from the database:

Table 4.2: Sample Web Service Descriptions

ID	URLID	Service Description	Service Name
249	34	ResultsGetBooking GetBooking(Int32 BookingNumber, String ConfirmationKey)	GetBooking
256	35	CascadingDropDownNameValue[] GetModel(String knownCategoryValues, String category)	GetModel
261	36	Int16 Validate_CreditCardType(String ps_InputNo)	Validate_CreditCardType
262	36	Int16 Validate_CreditCard(String ps_InputNo)	Validate_CreditCard
263	36	Int16 Validate_SporeCarRegNo(String ps_InputNo)	Validate_SporeCarRegNo

The five lines in the Table 4.2 are taken from the database as sample.

2. Getting URL description of the first line: URLID of the first line is 34. The record with ID 34 of the URL Table is given as 4.3;

Table 4.3: URL Descriptions

ID	ServiceURL
34	http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx

3. Download the wsdl file: The program generates the address as ServiceURL + ”?wsdl” and uses ”index.asmx” as file name.

4. Set values of parameters: The name of the example service is GetBooking and the service takes two parameters BookingNumber and ConfirmationKey. The types of the ”BookingNumber” is Int32 and ”ConfirmationKey” is String. Therefore, the program sets the value of ”BookingNumber” as 1 and the value of the ”ConfirmationKey” as ”1”.

5. Invoke the web service: The service is called by using the generated parameter values

6. The process continues from the second step for the next web service.

4.2 QoS Parameter Values

Since a service is invoked by the program, the QoS parameters values can be taken automatically. The processes of tracking web service for QoS parameters - Response Time, Availability, Reliability and Throughput - are different from each other. Therefore in this section getting response time, availability, reliability and throughput values of web services are described seperately.

4.2.1 Response Time

Response Time represents the delay time of taking response from the service after calling it. In other words, it tells the speed of the algorithm in the invoked service. For this parameter, the time just before calling service and the time just after receiving respond from the service are taken. The difference of the taken time values gives the response time of the service and is recorded into the database as a value in miliseconds. Generally, users prefer services that return quick response, therefore a high value for response time parameter reduces the QoS for the service. Hence, QoS is inversely proportional to the value of response time [22].

4.2.2 Availability

Availability represents the accessibility and effectivity of a service. After calling a service, if it returns any result, it means that the service is available at that time. If the service is inactive or returns an exception, it means the service is not available. When the service returns a result in other words if the service is available, then value 1 is written to the database as the availability value. When the service is inactive in other words if the service is not available, then the value 0 is written to the database. For the Availability parameter, the high number of the value 1 for a service will increase the QoS value of the service. It means that, the service is active and accessible in different times during a period of lifetime. Therefore it becomes preferable when availability is important for user [22].

4.2.3 Reliability

Reliability refers that a service returns the same result when invoked at different time instances, with the same parameter. Therefore to get reliability value of a service, the result value of each service invocation has to be recorded for result comparison. To do that, ServiceResponses table is created into the database. When program gets the result of a service, it serializes the result object to XML and records the content of the serialized object into the ServiceResponses table.

In order to obtain reliability value, we call service with the same parameters and get the result. The program serializes the result object to XML for the next invocation, gets the previous result for the service recorded in the ServiceResponses table and compares the contents of these results. If the same result is obtained, we record 1 to the database as reliability value and record 0 for different result. If the service is invoked for the first time, then there would be no result value for comparison. Therefore, for this situation -1 is recorded in the database and this value will not be included in the calculation algorithms. As in availability, for reliability parameter, the high number of the value 1 for a web service will make the QoS value of the service increase [22].

4.2.4 Throughput

Throughput value refers to whether the service can handle high number of concurrent calls. When calculating the maximum number of concurrent calls, if the maximum number is exceeded, the system gives an exception error. Therefore, in our program, we decided to use the effect of concurrent calls to response time for throughput value.

In order to obtain throughput value, the program starts 50 threads for each web service concurrently. All threads calculate response time of the service seperately. After all threads finish their jobs, the program records the average of the response times calculated by threads to the database as throughput value. The difference between response time of one service call (the value calculated in Response Time) and the average value of these response times of 50 threads gives us the behaviour of the service under 50 clients. Higher throughput will decrease QoS value of the service. QoS is inversely proportional to the value of throughput, as in response time [22].

CHAPTER 5

AUTOMATIC QOS CALCULATION ALGORITHM FOR WEB SERVICES

In this chapter, we describe the proposed algorithm that aims to evaluate values of the QoS parameters by considering their ages and set a QoS value for each Web services in range [0-1].

The proposed algorithm takes date/time value for each QoS parameter set as prerequisite. The date/time value keeps the date of the invocation of a web service. In addition, values of QoS parameters must be in ascending order by date/time. The date/time value helps us set the age of the values.

By ordering QoS parameter values, the newest QoS value set becomes the last, and the oldest value set becomes the first value set of the ordered list. The oldest QoS value set (the first value in the ordered list) is set to be 1. Each QoS value set takes the day difference between their date/time value and date/time value of the oldest set as age. Therefore, the newest set (the last value in the oredered list) has highest value as age.

Assume a web service invoked five times in different days and values of the QoS parameters are taken as in Table 5.1.

As we can see in Table 5.1 the values are sorted in ascending order by date/time. The age of the first value set taken in 12.03.2011 is set to be 1. The age of the second line becomes 3 and the age of the third line becomes 4 and the ages of rest of the lines are set in the same way, as shown in Table 5.2

There are five steps in the proposed algorithm. In the first four step, average QoS parameter values (response time, availability, reliability and throughput values) are calculated. These

Table 5.1: QoS Parameters for a Web Service

Response Time (ms)	Availability	Reliability (ms)	Throughput	Date Time
3200	1	1	29400	12.03.2011
2700	1	0	25760	14.03.2011
1800	1	0	19100	15.03.2011
1500	1	1	16590	17.03.2011
1000	1	1	12090	19.03.2011
1200	1	1	14530	23.03.2011
980	1	1	10920	28.03.2011

Table 5.2: QoS Parameters for a Web Service

Response Time (ms)	Availability	Reliability (ms)	Throughput	Date Time	Age
3200	1	1	29400	12.03.2011	1
2700	1	0	25760	14.03.2011	3
1800	1	0	19100	15.03.2011	4
1500	1	1	16590	17.03.2011	6
1000	1	1	12090	19.03.2011	8
1200	1	1	14530	23.03.2011	12
980	1	1	10920	28.03.2011	17

four steps give the results in range [0-1] and considers ages of the values while calculating values of QoS parameters. In the last step, overall QoS is calculated by using the results of the previous steps.

While describing steps of the algorithm, the values in Table 5.1 are used as a running examples.

5.1 Calculating Response Time

Response time for a web service is kept as a value in miliseconds in the database. Response time value in this form is not helpful to determine whether the web service is good enough to use in a system point of view for response time. Therefore, to evaluate response time of web service, instead of using response time value itself, the progress in the response time is considered in this algorithm.

To see the progress, we first calculate the average of the response time values with the equation

5.1.

$$\frac{\sum_{k=1}^{N} RT_k}{N} \tag{5.1}$$

where RT_k is response time value and N is the length of the QoS parameter values set.

As we mentioned in previous sections, the weight of the recent QoS value set should be higher than the older ones. So, when we calculate the response time average by considering ages of the value sets with the equation 5.2, so that the resulting value reflects the progress of the service.

$$\frac{\sum_{k=1}^{N} RT_k * a_k}{\sum_{k=1}^{N} a_k} \tag{5.2}$$

where RT_k is the response time value, a is the age of the value and N is the length of the QoS parameter values set.

The comparison between these two equations gives us an important information about progress of the web service. If the result of 5.2 is higher than 5.1, then the web service is slower than before. But if the result of 5.1 is higher than 5.2, then the web service is faster than before.

Since high response time value makes the quality low, if the result of 5.2 is higher than 5.1, then the quality of the web service must be low. But if the result of 5.1 is higher than 5.2, then the quality of the web service must be high.

The last step for calculation Response Time value is to normalize the value in range [0-1]. For the equation assume that;

- The result of the equation 5.1 is M
- The result of the equation 5.2 is N

$$ResponseTime = \begin{cases} 0 & \text{if } N > 2M \\ 0.5 - \frac{N-M}{2M} & \text{if } N > M \\ 0.5 & \text{if } N = M \\ 0.5 + \frac{M-N}{2M} & \text{if } N < M \end{cases} \tag{5.3}$$

In this equation, we considered that if the results of the equations 5.1 and 5.2 are the same, then the web service has no progress in positive or negative way, so we give 0.5 point for response time value. If the results of 5.1 is higher than the result of 5.2, which means there is a progress in positive way, we add the ratio of the progress to 0.5. But if the result of 5.1 is smaller than the result of 5.2, which means there is a progress in negative way, we substract the ratio of the progress from 0.5. If the result of the substraction becomes a negative value then the result is set to 0.

If we apply the response time calculation algorithm to the values in Table 5.1;

- At first, the average of the response times is calculated

$$\frac{3200 + 2700 + 1800 + 1500 + 1000 + 1200 + 980}{7} = 1769 \tag{5.4}$$

- Secondly, the average of the response times is calculated by considering the age values

$$\frac{(3200 * 1) + (2700 * 3) + (1800 * 4) + (1500 * 6) + (1000 * 8) + (1200 * 12) + (980 * 17)}{(1 + 3 + 4 + 6 + 8 + 12 + 17)} = 1305 \tag{5.5}$$

- Finally, equation 5.3 is applied for the values 1769 and 1305. Since M is 1769 and N is 1305, fourth rule of the equation 5.3 gives us the result of the response time value for this example. The calculation of the final result is given in 5.6

$$0.5 + \frac{M - N}{2M} = 0.5 + \frac{1769 - 1305}{2 * 1305} = 0.68 \tag{5.6}$$

5.2 Calculating Availability

Availability represents the accessibility of a web service in a certain period of lifetime. As mentioned before, while invocation of a web service, if the service gives a response, then ”1” is set as the availability value in the database. If there is no response, "0" is recorded into the database.

At the first sight, one may think that the average of the values in the database may give us the availability of a web service. Assume that, a web service is not accessible at the beginning of the period of value collection, but it is always accessible other times including use time of the service. Assume that another web service is accessible at the beginning but it is not accessible other times. In these two cases, averages of the values may be nearly same but it is not fair to give the same availability value for both web service. Therefore, instead of taking averages of the values without ages of the values, an equation considering ages must be used for calculating availability. By considering ages, availability value in the first case differs from the second case. Availability value in the first case becomes higher than the availability value in the second case.

$$\frac{\sum_{k=1}^{N} A_k * a_k}{\sum_{k=1}^{N} a_k} \tag{5.7}$$

where A_k is the availability value at time k, a is the age of the value and N is the number of the QoS parameter values recorded.

The equation 5.7 calculates the average of the availability by considering ages of the recorded values. In Table 5.1, all availability values recorded in the database are "1", therefore the availability value must be "1". If we apply the equation 5.7, the same result is obtained as shown in 5.8.

$$\frac{(1*1)+(1*3)+(1*4)+(1*6)+(1*8)+(1*12)+(1*17)}{1+3+4+6+8+12+17} = 1 \tag{5.8}$$

5.3 Calculating Reliability

Reliability represents the consistency of a web service. In other words, it refers to whether a service returns the same result for the same parameters when invoked at different time. As in Availability, while invoking a web service, if the service gives the same response as the previous response with the same paramaters, then "1" is recorded as the reliability value into the database. If the response is different from the response of the previous invocation, "0" is set into the database. For the first invocation for a web service ”-1” is set into the database. But the value ”-1” is not used in any equation including calculation reliability.

The example cases mentioned in calculating availability part occurs for reliability, as well. Assume that there are two web services. The first web service gives the same response at the beginning of the invocations, but in recent invocations, the responses differ from each other. The second web service gives different responses at the beginning of the invocations. However, in recent invocations, it always gives the same responses. The reliability values of these two web services must be different. The reliability value of the first web service must be less than reliability value of the second web service. Since using averages without considering ages would not reflect this difference, the equation for calculating reliability value must consider ages of the recorded values.

$$\frac{\sum_{k=1}^{N} R_k * a_k}{\sum_{k=1}^{N} a_k} \tag{5.9}$$

where R_k is the reliability value at time k, a is the age of the value and N is the number of the QoS parameter values recorded.

The equation 5.9 calculates the average of the reliability by considering ages of the values.

For the example given in Table 5.1, the calculated reliability value s shown in 5.10.

$$\frac{(1*1) + (0*3) + (0*4) + (1*6) + (1*8) + (1*12) + (1*17)}{1 + 3 + 4 + 6 + 8 + 12 + 17} = 0.86 \tag{5.10}$$

5.4 Calculating Throughput

Throughput represents how many concurrent invocations a web service can handle. For throughput value of a web service, 50 concurrent transactions are used to invoke the service and their average response times is kept in the database as throughput value. Therefore, in the database, values in milliseconds are recorded for throughput as in response time. Here the important thing is to be able to follow the progress of the service for concurrent calls. If the service can handle more concurrent calls recently, than throughput value must be high, but if the service can handle less concurrent calls, then the throughput value must be low. In order to observe the progress, firstly, the average of the throughput values is calculated as given in equation 5.11.

$$\frac{\sum_{k=1}^{N} T_k}{N} \tag{5.11}$$

where T_k is throughput value at time k and N is the number of the QoS parameter values recorded.

After that, the average of throughput values is calculated again. However, this time the equation includes the age of the recorded values, as shown in equation 5.12.

$$\frac{\sum_{k=1}^{N} T_k * a_k}{\sum_{k=1}^{N} a_k} \tag{5.12}$$

where T_k is the throughput value at time k, a is the age of the value and N is the number of the QoS parameter values recorded.

The comparison between equations 5.11 and 5.12 gives the progress of the web service. If the result of 5.11 is higher than the result of 5.12, then the web service can handle more or the same number of concurrent calls in less time. However, if the result of 5.12 is higher than the value of 5.11, then the web service can handle less or the same number of concurrent calls in more time.

If the results of the equations 5.11 and 5.12 are the same, then the web service has no progress in positive or negative way, so we give 0.5 point for throughput value. If the results of 5.11 is higher than the result of 5.12, which means there is a progress in positive way, we add the ratio of the progress to 0.5. But If the results of 5.11 is smaller than the result of 5.12, which means there is a progress in negative way, we substract the ratio of the progress from 0.5. If the result of the substraction becomes a negative value then the result is set to 0. The corresponding equation is given in 5.13. In this equation, assume that

- The result of the equation 5.11 is M
- The result of the equation 5.12 is N

$$Throughput = \begin{cases} 0 & \text{if } N > 2M \\ 0.5 - \frac{N-M}{2M} & \text{if } N > M \\ 0.5 & \text{if } N = M \\ 0.5 + \frac{M-N}{2M} & \text{if } N < M \end{cases} \tag{5.13}$$

When the algorithm is applied to the example values in Table 5.1;

- At first, the average of throughput is calculated

$$\frac{29400 + 25760 + 19100 + 16590 + 12090 + 14530 + 10920}{7} = 18341 \tag{5.14}$$

- Secondly, the average of the throughput is calculated by considering the age values

$$\frac{(29400 * 1) + (25760 * 3) + (19100 * 4) + (16590 * 6) + (12090 * 8) + (14530 * 12) + (10920 * 17)}{(1 + 3 + 4 + 6 + 8 + 12 + 17)} = 14497 \tag{5.15}$$

- Finally, the comparison in 5.13 must be done for the values 1769 and 1305. Since M is 18341 and N is 14497, fourth rule of the equation 5.13 gives us the result of the throughput value for this example, as shown in 5.16.

$$0.5 + \frac{M - N}{2M} = 0.5 + \frac{18341 - 14497}{2 * 14497} = 0.63 \tag{5.16}$$

5.5 Calculating Overall QoS Value

The final step of the proposed algorithm is calculating the overall QoS value by considering the calculated response time, availability, reliability and throughput values. The user may want to use each QoS parameters with different weights. For example the user may want a service which is available as long as possible but response time is not important. In this situation the user decreases the weight of response time but increases weight of availability on the screen provided by the GUI. Therefore, in this calculation, weights of each QoS parameters are included, given as 5.17. In this equation assume that;

- The result of the response time calculation is RT
- The result of the availability calculation is A
- The result of the reliability calculation is R
- The result of the throughput calculation is T

$$\frac{(w_{RT} * RT) + (w_A * A) + (w_R * R) + (w_T * T)}{w_{RT} + w_A + w_R + w_T} \tag{5.17}$$

where w_{RT} is the weight of response time, w_A is the weight of availability, w_R is the weight of reliability and w_T is the weight of throughput.

In this study, the weights are values between 0 and 100. In addition, the average of the weights is used as the threshold value on the result list. The weights are taken from the user through GUI.

When the equation is applied to the calculation results of the example values in Table 5.1, the result of the response time is 0.68, the result of the availability is 1, the result of the reliability is 0.86 and the result of the throughput is 0.63. Assume that, weight of the response time is set to be 80, weight of the availability is set to be 60, weight of the reliability is set to be 80 and weight of the throughput is set to be 100;

$$\frac{(80 * 0.68) + (60 * 1) + (80 * 0.86) + (100 * 0.63)}{80 + 60 + 80 + 100} = 0.77 \tag{5.18}$$

The QoS value of the example web service is calculated as 0.77.

5.6 Evaluating Price Value and Sort Algorithm

In the Web, there are a lot of services which are cost-free but there are also service that must be paid to use. The price value does not effect the quality of the service. But the user may want to express his/her preference for the price of the web service. S/He may want to use a free web service or a web service up to a certain cost. The price value can be set by user. The given value is used as the maximum cost that user can afford. Therefore, when user set a value from the GUI, web services with higher prices are eliminated from the result list.

In this work, a service is implemented such that it takes the list of web services and calculates QoS values of the web services by including weights of QoS parameters that are taken from user and returns a list ordered by QoS values. The service also eliminates web services whose costs are more than user can afford. After this filtering, if there are web services that have the same QoS value, then web services are ordered by the price values of the services.

While sorting the web services with respect to their QoS values, some tuning is necessary to break the ties. Assume that there are two services having the same overall QoS values in our list. If both services have the same progress in response time and throughput values but the averages of response times and throughput are different so these service should have different QoS values. To this aim, while sorting web services, average value of response time by considering the age of the recorded values is used as the weight of the services in sorting. The same rule is used for throughput value, as well.

To give an example, assume that there are two web services which have the same response time and throughput values for each invocation and they are both availabile and reliable. But the response time of the first service is 800 ms, and the response time of the second service is 1000 ms. In this situation, each QoS parameter value for these two web services becomes the same. Assume that response time value of each service is 0.5. The response time value of the second service is multiplied by 800/(800+1000) and the response time value of the first service multiplies by 1000/(800+1000). The resulting values of these multiplications are added to response time values. By this way, the response time value of first service becomes (0.5 + 0.28) = 0.78 and the response time value of the second service becomes (0.5 + 0.22) = 0.72. As a result, when the overall QoS values are the same, the Web service with smaller response time is favoured.

CHAPTER 6

CASE STUDIES AND EVALUATION

In this chapter, the experimental results of the proposed algorithm are presented. This chapter consists of two parts. The first part presents the description of the real Web services obtained from the Web through DSWSD and tracked by using the proposed method. This part also contains QoS value calculation with real values for two services. The second part presents comparison of the QoS results obtained by the proposed algorithm and by the algorithm given in [20].

6.1 Web Services used in the Experiments

In this project 15 sample Web services discovered from the Web by the service discovery system DSWSD-S are selected and tracked for about one month. In Table 6.1, information about these web services can be seen. The services are invoked once a day and mostly at the evening hours. In Table 6.2, the URL information about example web services are given.

In this thesis, the services in Table 6.1 are used as test services. Column ID represents primary key of each row. URLID is the database primary key of the URL of the service. ServiceDescription column gives the description of the service which includes name, parameters and result type of the service. The last column, which is ServiceName, keeps the name of the service. URL descriptions of each service in Table 6.1 are shown in Table 6.2.

As the URLs and service names imply, test services are taken from car domain. The Web service tracking procedure given in Chapter 4 is applied to the web services given in Table 6.1. The developed program is used in different times and the values for each web service are taken and recorded into the database. Recorded values of services 257, 258, 260 and 266

Table 6.1: Web Service Descriptions

ID	URLID	Service Description	Service Name
250	34	ResultsGetCountries GetCountries()	GetCountries
251	34	ResultsGetLocations GetLocations(Int32 CountryID)	GetLocations
252	34	ResultsCarAvailabilityByAirport CarAvailabilityByAirport(String PickupAirport, String PickupDate, String DropoffDate, String PickupTime, String DropoffTime, String Currency)	CarAvailability ByAirport
253	34	ResultsCarAvailabilityByTownCity CarAvailabilityByTownCity(String PickupTownCity, String PickupDate, String DropoffDate, String PickupTime, String DropoffTime, String Currency)	CarAvailability ByTownCity
254	34	ResultsCarAvailabilityByLocationID CarAvailabilityByLocationID(Int32 PickupLocationID, Int32 DropoffLocationID, String PickupDate, String DropoffDate, String PickupTime, String DropoffTime, String Currency)	CarAvailability ByLocationID
255	34	ResultsGetBooking GetBooking(Int32 BookingNumber, String ConfirmationKey)	GetBooking
256	34	ResultsGetVoucher GetVoucher(Int32 BookingNumber, String ConfirmationKey)	GetVoucher
257	35	CascadingDropDownNameValue[] GetMake()	GetMake
258	35	CascadingDropDownNameValue[] GetModel(String knownCategoryValues, String category)	GetModel
260	43	System.String[] GetSuburbSuggestions(String prefixText, Int32 count)	GetSuburb Suggestions
263	48	CascadingDropDownNameValue[] GetMakesByYear(String knownCategoryValues, String category, String contextKey)	GetMakesByYear
264	48	CascadingDropDownNameValue[] GetModelsByMake(String knownCategoryValues, String category, String contextKey)	GetModelsByMake
265	55	CascadingDropDownNameValue[] GetMake(String knownCategoryValues, String category)	GetMake
266	55	CascadingDropDownNameValue[] GetModel(String knownCategoryValues, String category)	GetModel
268	55	CascadingDropDownNameValue[] GetMakeCount(String knownCategoryValues, String category)	GetMakeCount

Table 6.2: Web Service URL Informations

ID	Service URL
34	http://www.reservations.wwcars.co.uk/WWCarsOnlineRes/XML/wwcarsXMLInterface.asmx
35	http://cardealer.com.pk/Services/cars.asmx
43	http://www.fixedpricecarservice.com.au/Services/WebService.asmx
48	http://www.carquotes.com/Tools/DynamicPopulateDictionary.asmx
55	http://carseller.co.nz/controls/searchService.asmx

Table 6.3: QoS Parameter Values of Service 257

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
257	984,375	12400	1	1	07.03.2011 22:04
257	875	11546,5625	1	1	09.03.2011 22:45
257	937,5	11667,5	1	1	10.03.2011 23:20
257	875	11430,3125	1	1	12.03.2011 10:56
257	1078,125	11637,1875	1	1	14.03.2011 15:53
257	968,75	12637,5	1	1	15.03.2011 16:03
257	1156,25	13158,4375	1	1	19.03.2011 13:05
257	1140,625	11368,125	1	1	20.03.2011 12:51
257	1140,625	13866,875	1	1	23.03.2011 23:18
257	1031,25	13396,25	1	1	24.03.2011 22:29
257	1390,625	14382,5	1	1	29.03.2011 23:23
257	1321,875	12805	1	1	01.04.2011 23:41
257	1475	13627,5	1	1	03.04.2011 12:56
257	1475,5	14011,875	1	1	04.04.2011 18:15
257	1393,75	14115,625	1	1	05.04.2011 16:03

are given in Tables 6.3-6.7 and progress charts of response time and throughput paramaters of these services are given in Figures 6.1-6.5. These services are used in the next section for evalution, therefore we present the detailed list of recorded values here just for these services. QoS values of other services are given in Appendix A.

As shown in the tables, each service has its own characteristics. QoS parameters of each service changes day by day. Response times of services generally do not have a specific path. However, when the values of Service 257 is examined, it is observed that the response time values are increasing day by day. So its response time quality becomes less than other services. Service 268 and Service 266 are not reliable services because they return different result for each invocation. Service 260 gives a different result once, the reliability point is lower than other services.

For reliability and availability, there should be more data to make a good comparison. In this calculation whole data in the database is used. After the proposed system runs and collects data for a long period, subset of data may be used for calculation.

Table 6.4: QoS Parameter Values of Service 258

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
258	968,75	12426,25	1	1	07.03.2011 22:04
258	1156,25	14342,8125	1	1	09.03.2011 22:45
258	875	12154,6875	1	1	10.03.2011 23:20
258	890,625	11210	1	1	12.03.2011 10:56
258	1853,25	13751,875	1	1	14.03.2011 15:53
258	953,125	11549,375	1	1	15.03.2011 16:03
258	1156,25	14310,9375	1	1	19.03.2011 13:05
258	1109,375	13672,8125	1	1	20.03.2011 12:51
258	1125	24643,75	1	1	23.03.2011 23:18
258	1015,625	11344,375	1	1	24.03.2011 22:29
258	937,5	11586,25	1	1	29.03.2011 23:23
258	937,5	11383,4375	1	1	01.04.2011 23:41
258	875	10675	1	1	03.04.2011 12:56
258	921,875	11261,5625	1	1	04.04.2011 18:15
258	1031,25	12209,0625	1	1	05.04.2011 16:03

Table 6.5: QoS Parameter Values of Service 260

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
260	3625	12867,8125	1	1	07.03.2011 22:04
260	1218,75	11910,3125	1	1	09.03.2011 22:45
260	1203,125	11010,625	1	1	10.03.2011 23:20
260	1515,625	9688,125	1	1	12.03.2011 10:56
260	2265,625	25399,0625	0	1	14.03.2011 15:53
260	2218,75	11918,75	1	1	15.03.2011 16:03
260	2078,125	11742,8125	1	1	19.03.2011 13:05
260	1531,25	8984,375	1	1	20.03.2011 12:51
260	1406,25	11413,125	1	1	23.03.2011 23:18
260	1343,75	12781,25	1	1	24.03.2011 22:29
260	1265,625	14801,25	1	1	29.03.2011 23:23
260	1234,375	14132,1875	1	1	01.04.2011 23:41
260	1171,875	13567,1875	1	1	03.04.2011 12:56
260	1468,75	13261,5625	1	1	04.04.2011 18:15
260	1390,625	9299,6875	1	1	05.04.2011 16:03

Table 6.6: QoS Parameter Values of Service 266

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
266	218,75	3117,1875	1	1	07.03.2011 22:04
266	546,875	4248,125	1	1	09.03.2011 22:45
266	328,125	4615,3125	0	1	10.03.2011 23:20
266	171,875	2188,125	0	1	12.03.2011 10:56
266	234,375	7413,75	0	1	14.03.2011 15:53
266	218,75	3744,375	0	1	15.03.2011 16:03
266	312,5	2064,6875	0	1	19.03.2011 13:05
266	250	3383,125	0	1	20.03.2011 12:51
266	234,375	3305,625	0	1	23.03.2011 23:18
266	265,625	3418,125	0	1	24.03.2011 22:29
266	375	4635,3125	0	1	29.03.2011 23:23
266	265,625	6096,875	0	1	01.04.2011 23:41
266	140,625	2571,875	0	1	03.04.2011 12:56
266	156,25	1975	0	1	04.04.2011 18:15
266	359,375	6644,375	0	1	05.04.2011 16:03

Table 6.7: QoS Parameter Values of Service 268

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
268	218,75	3044,6875	1	1	07.03.2011 22:04
268	343,75	4701,5625	0	1	09.03.2011 22:45
268	328,125	4237,8125	0	1	10.03.2011 23:20
268	171,875	1970	0	1	12.03.2011 10:56
268	265,625	4728,4375	0	1	14.03.2011 15:53
268	359,375	5485,3125	0	1	15.03.2011 16:03
268	140,625	1984,0625	0	1	19.03.2011 13:05
268	281,25	4500,3125	0	1	20.03.2011 12:51
268	234,375	3243,75	0	1	23.03.2011 23:18
268	265,625	3610	0	1	24.03.2011 22:29
268	375	4456,5625	0	1	29.03.2011 23:23
268	250	3290,9375	0	1	01.04.2011 23:41
268	140,625	1795,625	0	1	03.04.2011 12:56
268	203,125	1870,3125	0	1	04.04.2011 18:15
268	828,125	6181,5625	0	1	05.04.2011 16:03

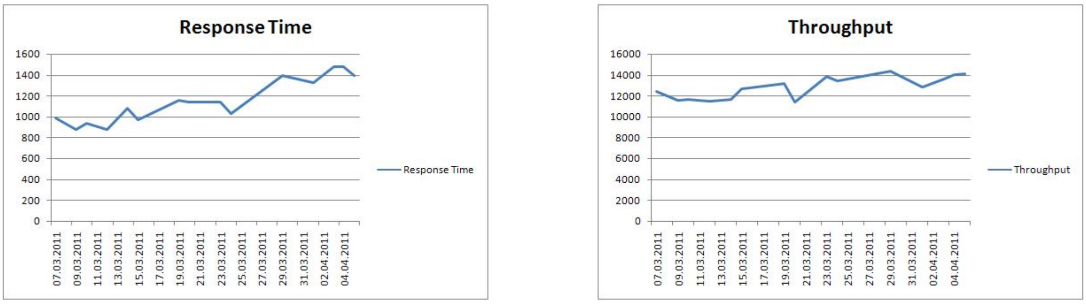


Figure 6.1: Response Time and Throughput of Service 257

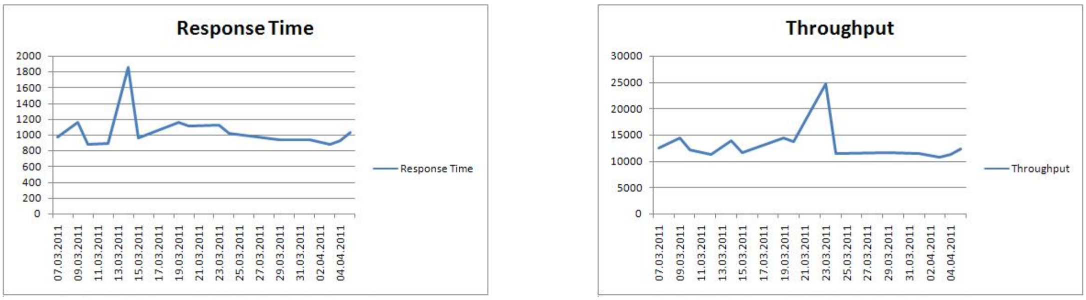


Figure 6.2: Response Time and Throughput of Service 258

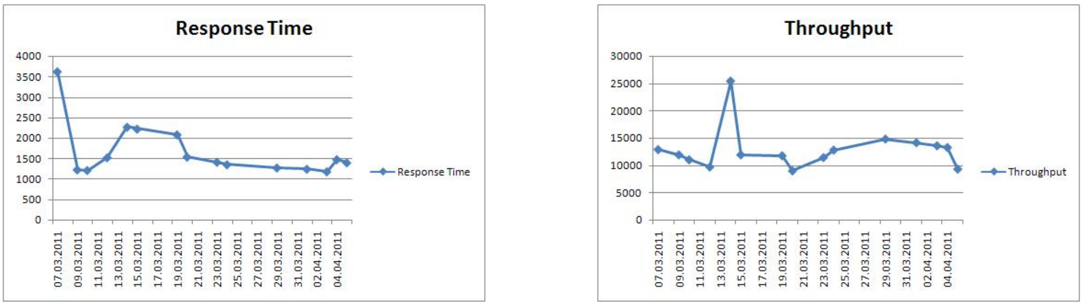


Figure 6.3: Response Time and Throughput of Service 260

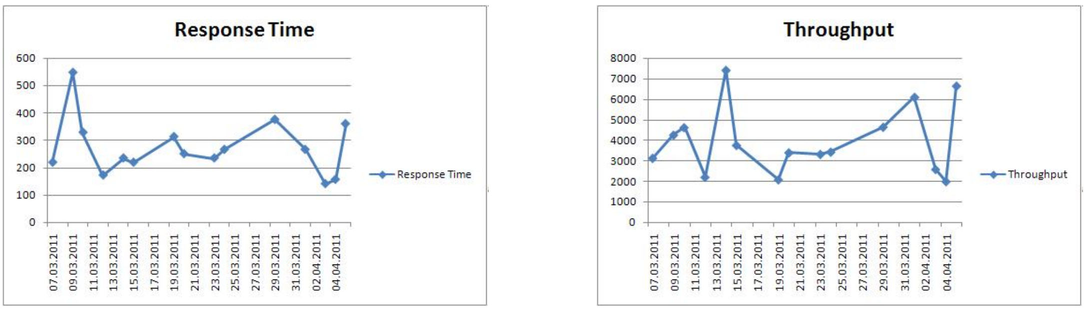


Figure 6.4: Response Time and Throughput of Service 266

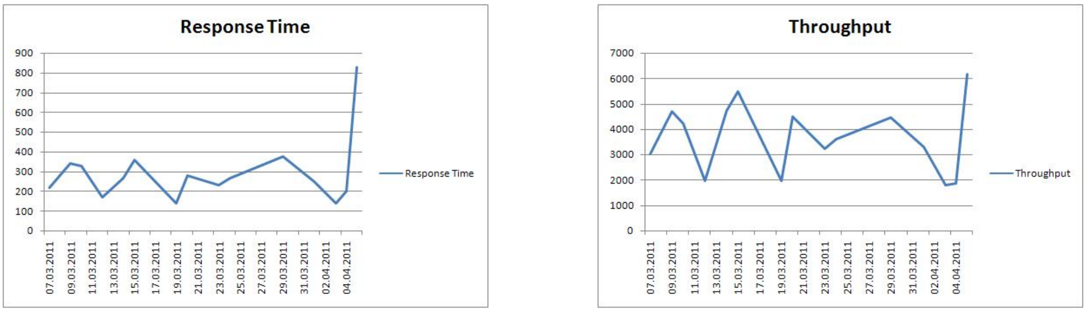


Figure 6.5: Response Time and Throughput of Service 268

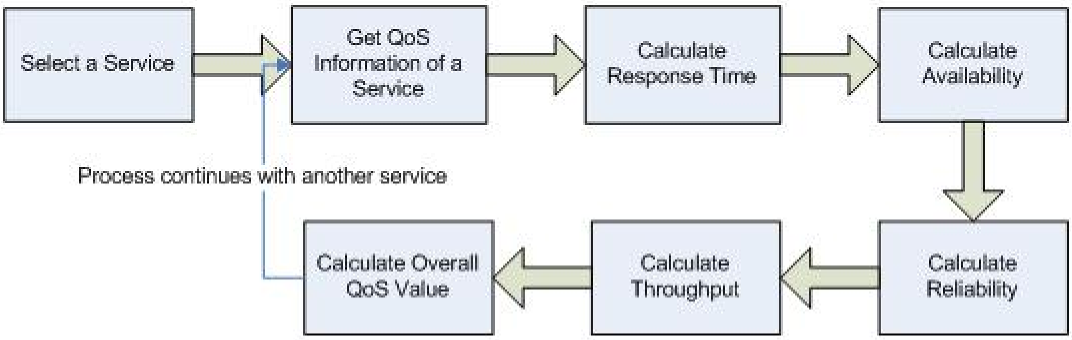


Figure 6.6: Proposed Algorithm

6.2 Comparison of Algorithms

In this part of this chapter, a comparison between proposed algorithm in our work and algorithm given in [20] is given. For comparison, the real values of services 257, 258, 260 and 266, which are given in previous section are used.

Figure 6.6 represents the algorithm given in Chapter 5. QoS calculation starts with selecting a service to evaluate. When a service is selected QoS parameter values are retrieved from the database. Following this, for each QoS parameter, evaluation calculation is done and lastly, overall QoS value is calculated. When the proposed algorithm in Figure 6.6 is applied to the values of the tables given, the results can be seen as the second column of Table 6.8.

The equation in the algorithm given in [20] can be given as Formula 6.1.

Table 6.8: QoS Results

Service ID	Overall QoS value by the proposed algorithm	Overall QoS value by [20]
250	0,75	3701,20
251	0,75	3726,75
252	0,76	3655,37
253	0,78	4169,91
254	0,77	15740,35
255	0,75	18148,91
256	0,76	13737,54
257	0,71	13955,03
258	0,76	14157,23
260	0,78	14516,37
263	0,78	13709,64
264	0,78	15622,47
265	0,80	11931,33
266	0,51	4334,33
268	0,48	3968,14

$$\sum W_m * q_i + \sum W_n * \frac{1}{q_j} \tag{6.1}$$

where W_m and W_n are weights of QoS parameters; q_i is a negative QoS parameter and q_j is positive QoS parameter.

When QoS values are calculated with the proposed algorithm in [20], then the results are as the third column of Table 6.8.

According to the QoS calculation algorithm given in [20], the service with less QoS value is more preferrable. In Figure 6.7, first chart gives QoS values of services calculated by our algorithm and second chart gives QoS values of services calculated by proposed algorithm in [20].

As seen in the figure, the QoS values calculated by algorithm in [20] are hard to interpret by itself since the range of the value is not known themselves. These values are useful only for comparison of web services.

When we check the descriptions of the services, it is seen that service 258 and service 266 are functionally the same. They both return models of cars. Assume that both services are returned as the result of some service query. When we compare them in terms of QoS, our algorithm gives that Service 258 is more preferable than Service 266. Even if response time

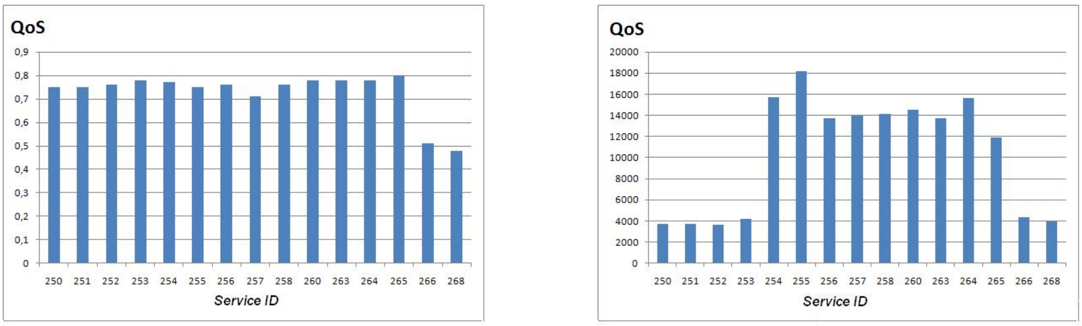


Figure 6.7: QoS values calculated with the Proposed Algorithm of This Work and the Algorithm in [20]

and throughput of Service 266 are better than Service 258, Service 266 is not reliable and response time quality of Service 266 decreases day by day. On the other hand the algorithm in [20] gives that Service 266 is more preferable. The progress of QoS values of Service 258 and Service 266 with the proposed algorithm are given in Figure 6.8.

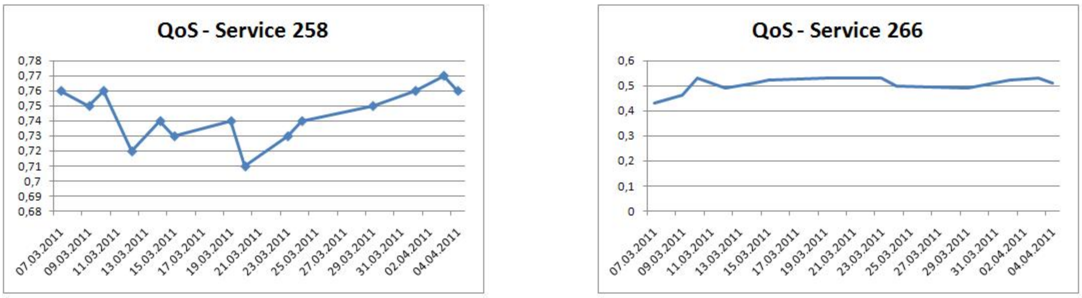
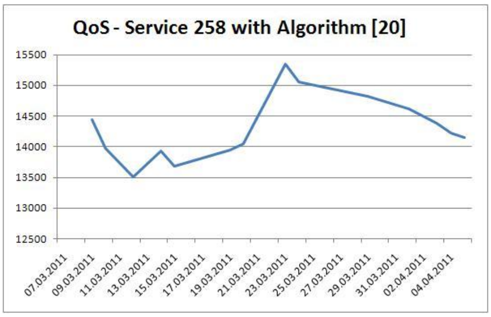


Figure 6.8: The progress of QoS values of Service 258 and Service 266 with Proposed Algorithm

The difference between QoS evaluations for Service 258 and Service 266 can be seen clearly in Figure 6.8. Since Service 266 is not reliable and the other QoS parameters are not effective enough, QoS value is less than Service 258 at all the time. Figure 6.9 shows the progress of QoS values of Service 258 and Service 266 with Algorithm [20]. In this figure it is hard to interpret the QoS evaluation result since calculation is accumulative and recent values are not emphasized. However it is clear that in contrast to our method it gives higher QoS value to Service 266 than Service 258. There are many reasons for the difference between two algorithms. The first one is that our algorithm considers the ages of recorded values but the algorithm in [20] does not. The second one is that in algorithm in [20], all parameters are used in the same formula. Since each parameter is in different ranges and has different features, each parameter should be examined differently. When a service is reliable and available, the reliability and availability values of the service becomes 1. Since reliability and availability are positive QoS factors, in algorithm in [20] opposites of these values are used. Since opposites of these values are 1 and overall QoS values has response time and throughput values in it, overall QoS value could not be affected by reliability and availability values.



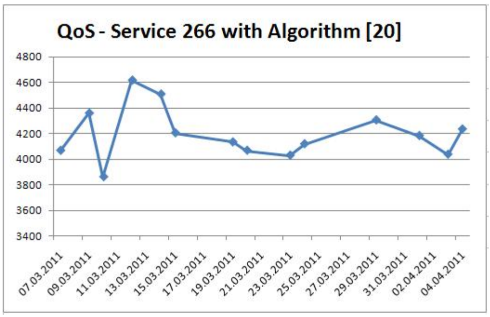


Figure 6.9: The progress of QoS values of Service 258 and Service 266 with Algorithm [20]

As another comparison, Service 257 and Service 265 are functionally equivalent as their names imply. Both algorithms gives the same result as Service 265 is more preferable than Service 257. However, the comparison is more clear for the results of the proposed algorithm since a definite range is used. The distance between the QoS values is clearly seen, as well.

Table 6.9: QoS Results

Service ID	Overall QoS value by the proposed algorithm	Overall QoS value by [20]
257	0,71	13955,03
265	0,80	11931,33

.

CHAPTER 7

CONCLUSION

In this thesis, the main idea of the work is to provide a method to find web services that meet their functional requirements with high enough quality. The proposed technique searches the Internet to find web services, records information about the discovered web services into the database and checks the services periodically to keep QoS information updated. The system includes a graphical user interface through which the user can enter keywords to search and set weights of QoS parameters. When a keyword is queried, the system returns a web service list that includes best matched services with their QoS values and it shows the list sorted by calculated QoS values in decreasing order.

The other aim of this thesis is to find more feasible way to handle QoS parameters and more efficient algorithm to calculate QoS values. In this thesis, we used five QoS parameters which are Response Time, Availability, Reliability, Throughput and Price. The first four parameters (Response Time, Availability, Reliability and Throughput) are used to calculate QoS values and the last parameter (Price) is used for refining the result list. As a future work, other QoS parameters can be tracked and added to the calculation algorithm. Therefore web services can be handled with different aspects and QoS values can be more efficient.

While calculating QoS values, we normalized the results in range [0-1] to make them comparable and meaningful among themselves. The calculation algorithm also considers ages of the recorded QoS parameter values which makes QoS values more comprehensible.

In the proposed system, users can set weights of QoS parameters as an integer. As a future work, the GUI may be extended to take fuzzy inputs for weights of QoS parameters. For example for response time parameter, in spite of using integer numbers between 0 and 100, using ”fast”, ”average” and ”slow” is more preferable for the users.

User may set specific values for parameters of web services. In addition, for some web services using default values for parameter may not give reliable results to calculate overall QoS values. Therefore, the system should verify the parameter values. Since the verification of parameters is out of scope of this thesis, the verification of parameter values is not considered. The system may be improved after a study on verification of web service parameters.

Some of web services may produce different results for each invocation. For instance, assume that there is a web service which returns the invocation date. This service returns different results for each invocation. Therefore, in our algorithm, reliability for this service becomes 0. After a study on verification of web service, semantics of such services can be discovered more clearly and input and expected output values can be determined more accurately. After such a study, the reliability parameter value for such web services can be calculated more reliably.

REFERENCES

[1] Zibin Zheng, Yilei Zhang, and Michael R. Lyu. *Distributed QoS evaluation for real-world web services*. IEEE International Conference on Web Services, 1:1 - 8, 2010.

[2] uddi.org, ”Introduction to UDDI:Important Features and Functional Concepts”. Retrieved at April 2, 2011 from http://uddi.org/pubs/uddi-tech-wp.pdf

[3] ibm.com, ”Understanding WSDL in a UDDI registry”. Retrieved at April 2, 2011 from http://www.ibm.com/developerworks/webservices/library/ws-wsdl/

[4] tutorialspoint.com, ”UDDI Data Model”. Retrieved at April 2,2011 from http://www.tutorialspoint.com/uddi/uddi_data_model.htm

[5] E. Michael Maximilien, Munindar P. Sing. *A Framework and Ontology for Dynamic Web Services Selection*. IEEE Internet Computing, 8(5):84-93, 2004.

[6] Shuping Ran. *A Model for Web Services Discovery with QoS*. ACM SIGecom Exchanges, vol. 4, no. 1, Spring 2003.

[7] Ziqiang Xu, Patrick Martin, Wendy Powley and Farhana Zulkernine. *Reputation-Enhanced QoS-based Web Services Discovery*. IEEE International Conference on Web Services (ICWS), pp. 249-256, 2007.

[8] Anna Averbakh, Daniel Krause, and Dimitrios Skoutas. *Exploiting User Feedback to Improve Semantic Web Service Discovery*. International Semantic Web Conferenc (ISWC) pp. 33-48, 2009.

[9] T. Rajendran and Dr.P. Balasubramanie. *An Optimal Agent-Based Architecture for Dynamic Web Service Discovery with QoS*. IEEE International Conference on Web Services (ICWS), 1-7, 2010.

[10] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam and Henry Chang. *QoS-Aware Middleware for Web Services Composition*. Proceedings of the 12th international conference on World Wide Web (WWW), 9-24, 2003.

[11] Delnavaz Mobedpour, Chen Ding and Chi-Hung Chi. *A QoS Query Language for User-Centric Web Service Selection*. IEEE International Conference on Services Computing, 1-8, 2010.

[12] Qian MA, Hao WANG, Ying LI, Guotong XIE and Feng LIU. *A Semantic QoS-Aware Discovery Framework for Web Services*. IEEE International Conference on Web Services, 1-8, 2008.

[13] Li xiaotang, Zhan feng and Zhan shaobin. *QoS-based Web Service Composition Technology Research*. IEEE International Conference on Services Computing, 1-4, 2010.

[14] G. Dobson, R. Lock, and I. Sommerville. *QoSOnt: a QoS Ontology for Service-Centric Systems*. 31st ERUOMICRO Conference on Software Engineering and Advanced Applications, 80-87, 2005.

[15] M. Tian, A. Gramm, H. Ritter and J. Schiller. *Efficient Selection and Monitoring of QoS-aware Web Services with the WS-QoS Framework*. IEEE/WIC/ACM International Conference on Web Intelligence, 152-158, 2004.

[16] K. Kritikos and D. Plexousakis. *Semantic QoS Metric Matching*. European Conference on Web Services, 265-274, 2006.

[17] A. D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. IEEE International Conference on Web Services, 789-796, 2006.

[18] Q.X. Du, C.H. Chi, S. Chen and J. M. Deng. *Modeling Service Quality for Dynamic QoS Publishing*. IEEE International Conference on Services Computing, 307-314, 2008.

[19] C. Herssens, I.J. Jureta and S. Faulkner. *Dealing with Quality Tradeoffs during Service Selection*. International Conference on Autonomic Computing, 77-86, 2008.

[20] Yuqiang Li, Qianxing Xiong and Xin Qi. *A New Algorithm about QoS of Web Service*. IEEE International Conference on Web Services, 1:1-3, 2010.

[21] D. Canturk and P.Senkul, *Using semantic information for distributed web service discovery*. International Journal of Web Science. 2011

[22] Emra Askaroglu, Hilal Ozdil and P.Senkul, *Automatic QoS Evaluation for Domain Specific Web Service Discovery*. in progress.

[23] D. Canturk and P.Senkul, *Semantic Annotation of Web Services with Lexicon-Based Alignment*. IEEE 7th World Congress on Services. July 2011.

[24] Enrique Lafuente Hernandez, *Evaluation Framework for Quality of Service in Web Services: implementation in a pervasive environment*. Master thesis in INSA Lyon. 2010.

[25] Vuong Xuan Tran,Hidekazu Tsuji and Ryosuke Masuda. *A new QoS ontology and its QoS-based ranking algorithm for Web services*. Simulation Modelling Practice and Theory 17, 1378-1398, 2009.

[26] D. Canturk and P.Senkul, *A Distributed Service Discovery System Consisting Of Domain Specific Sub-Systems*. Ph. D. Thesis Report. Spring 2011.

APPENDIX A

CASE STUDIES

A.1 QOS PARAMETER VALUES OF SERVICES

In this section, QoS parameters of services 250, 251, 252, 253, 254, 255, 256, 263, 264, 265 are given. These values are also recorded in the same time period as the services given in Chapter 6. While calculating QoS values of these services, the ages are given by the date values of each QoS parameter sets.

In addition, progress charts of response times and throughputs of services 250, 251, 252, 253, 254, 255, 256, 263, 264, 265 are given in this appendix.

Table A.1: QoS Parameter Values of Service 250

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
250	546,875	2435,9375	1	1	07.03.2011 22:04
250	265,625	2561,875	1	1	09.03.2011 22:45
250	343,75	4359,6875	1	1	10.03.2011 23:20
250	546,875	3133,125	1	1	12.03.2011 10:56
250	515,625	2761,25	1	1	14.03.2011 15:53
250	296,875	2937,1875	1	1	15.03.2011 16:03
250	703,125	3772,8125	1	1	19.03.2011 13:05
250	484,375	4458,75	1	1	20.03.2011 12:51
250	593,75	4782,5	1	1	23.03.2011 23:18
250	328,125	2954,0625	1	1	24.03.2011 22:29
250	500	3003,4375	1	1	29.03.2011 23:23
250	296,875	2896,875	1	1	01.04.2011 23:41
250	484,375	2335,3125	1	1	03.04.2011 12:56
250	328,125	2196,5625	1	1	04.04.2011 18:15
250	578,125	4086,25	1	1	05.04.2011 16:03

Table A.2: QoS Parameter Values of Service 251

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
251	375	2485,9375	1	1	07.03.2011 22:04
251	343,75	2277,1875	1	1	09.03.2011 22:45
251	312,5	2712,1875	1	1	10.03.2011 23:20
251	328,125	4114,375	1	1	12.03.2011 10:56
251	453,125	3373,4375	1	1	14.03.2011 15:53
251	359,375	2787,8125	1	1	15.03.2011 16:03
251	421,875	3394,375	1	1	19.03.2011 13:05
251	421,875	4299,6875	1	1	20.03.2011 12:51
251	390,625	4288,125	1	1	23.03.2011 23:18
251	296,875	2885,625	1	1	24.03.2011 22:29
251	359,375	3740,3125	1	1	29.03.2011 23:23
251	281,25	3016,875	1	1	01.04.2011 23:41
251	328,125	2340,625	1	1	03.04.2011 12:56
251	406,25	2345,3125	1	1	04.04.2011 18:15
251	515,625	6215,625	1	1	05.04.2011 16:03

Table A.3: QoS Parameter Values of Service 252

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
252	515,625	3186,5625	1	1	07.03.2011 22:04
252	343,75	2285,3125	1	1	09.03.2011 22:45
252	250	2722,8125	1	1	10.03.2011 23:20
252	437,5	4906,875	1	1	12.03.2011 10:56
252	390,625	3095,3125	1	1	14.03.2011 15:53
252	375	2838,125	1	1	15.03.2011 16:03
252	406,25	3304,0625	1	1	19.03.2011 13:05
252	468,75	4289,6875	1	1	20.03.2011 12:51
252	406,25	4360,3125	1	1	23.03.2011 23:18
252	281,25	2877,1875	1	1	24.03.2011 22:29
252	421,875	2800,625	1	1	29.03.2011 23:23
252	281,25	2792,1875	1	1	01.04.2011 23:41
252	343,75	2426,25	1	1	03.04.2011 12:56
252	328,125	2054,0625	1	1	04.04.2011 18:15
252	468,75	5142,5	1	1	05.04.2011 16:03

Table A.4: QoS Parameter Values of Service 253

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
253	343,75	4560,3125	1	1	07.03.2011 22:04
253	343,75	3103,4375	1	1	09.03.2011 22:45
253	265,625	3237,8125	1	1	10.03.2011 23:20
253	375	5814,6875	1	1	12.03.2011 10:56
253	421,875	4026,25	1	1	14.03.2011 15:53
253	406,25	2779,375	1	1	15.03.2011 16:03
253	468,75	4615,9375	1	1	19.03.2011 13:05
253	531,25	5078,75	1	1	20.03.2011 12:51
253	406,25	4410	1	1	23.03.2011 23:18
253	296,875	2977,5	1	1	24.03.2011 22:29
253	375	4628,125	1	1	29.03.2011 23:23
253	281,25	2856,875	1	1	01.04.2011 23:41
253	375	2331,5625	1	1	03.04.2011 12:56
253	328,125	2075,9375	1	1	04.04.2011 18:15
253	484,375	4319,0625	1	1	05.04.2011 16:03

Table A.5: QoS Parameter Values of Service 254

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
254	1031,25	31955	1	1	07.03.2011 22:04
254	3234,375	13457,1875	1	1	09.03.2011 22:45
254	1000	10726,5625	1	1	10.03.2011 23:20
254	1796,875	12327,5	1	1	12.03.2011 10:56
254	843,75	10613,4375	1	1	14.03.2011 15:53
254	3203,125	12298,4375	1	1	15.03.2011 16:03
254	2078,125	11038,75	1	1	19.03.2011 13:05
254	1921,875	10980,3125	1	1	20.03.2011 12:51
254	2171,875	12334,6875	1	1	23.03.2011 23:18
254	1625	13069,375	1	1	24.03.2011 22:29
254	3703,125	23683,4375	1	1	29.03.2011 23:23
254	1953,125	10841,5625	1	1	01.04.2011 23:41
254	1875	12122,5	1	1	03.04.2011 12:56
254	875	11125	1	1	04.04.2011 18:15
254	2250	9939,0625	1	1	05.04.2011 16:03

Table A.6: QoS Parameter Values of Service 255

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
255	1000	12699,0625	1	1	07.03.2011 22:04
255	1078,125	16450,9375	1	1	09.03.2011 22:45
255	1000	12373,4375	1	1	10.03.2011 23:20
255	1062,5	12450,9375	1	1	12.03.2011 10:56
255	1656,25	27228,4375	1	1	14.03.2011 15:53
255	1562,5	20090	1	1	15.03.2011 16:03
255	906,25	14153,75	1	1	19.03.2011 13:05
255	1046,875	16984,6875	1	1	20.03.2011 12:51
255	2218,75	34481,875	1	1	23.03.2011 23:18
255	2296,875	13539,6875	1	1	24.03.2011 22:29
255	1125	18921,25	1	1	29.03.2011 23:23
255	906,25	12495	1	1	01.04.2011 23:41
255	921,875	11852,8125	1	1	03.04.2011 12:56
255	1453,125	15608,125	1	1	04.04.2011 18:15
255	1062,5	13576,875	1	1	05.04.2011 16:03

Table A.7: QoS Parameter Values of Service 256

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
256	968,75	13742,5	1	1	07.03.2011 22:04
256	906,25	11695	1	1	09.03.2011 22:45
256	890,625	11451,25	1	1	10.03.2011 23:20
256	1046,875	11336,875	1	1	12.03.2011 10:56
256	968,75	11766,5625	1	1	14.03.2011 15:53
256	953,125	11581,875	1	1	15.03.2011 16:03
256	1171,875	13997,1875	1	1	19.03.2011 13:05
256	1125	13929,0625	1	1	20.03.2011 12:51
256	1125	21945,9375	1	1	23.03.2011 23:18
256	1000	11541,875	1	1	24.03.2011 22:29
256	937,5	11408,75	1	1	29.03.2011 23:23
256	937,5	12115	1	1	01.04.2011 23:41
256	875	10454,0625	1	1	03.04.2011 12:56
256	875	11469,6875	1	1	04.04.2011 18:15
256	968,75	12847,5	1	1	05.04.2011 16:03

Table A.8: QoS Parameter Values of Service 263

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
263	859,375	15473,125	1	1	07.03.2011 22:04
263	1218,75	12020,3125	1	1	09.03.2011 22:45
263	734,375	15449,375	1	1	10.03.2011 23:20
263	1156,25	21047,8125	1	1	12.03.2011 10:56
263	781,25	13208,4375	1	1	14.03.2011 15:53
263	890,625	17443,75	1	1	15.03.2011 16:03
263	687,5	8745	1	1	19.03.2011 13:05
263	781,25	12369,375	1	1	20.03.2011 12:51
263	1000	13951,875	1	1	23.03.2011 23:18
263	906,25	12094,0625	1	1	24.03.2011 22:29
263	843,75	9887,8125	1	1	29.03.2011 23:23
263	843,75	10602,5	1	1	01.04.2011 23:41
263	734,375	9080	1	1	03.04.2011 12:56
263	718,75	9027,8125	1	1	04.04.2011 18:15
263	968,75	12088,4375	1	1	05.04.2011 16:03

Table A.9: QoS Parameter Values of Service 264

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
264	1343,75	10497,8125	1	1	07.03.2011 22:04
264	984,375	12485	1	1	09.03.2011 22:45
264	1437,5	12891,875	1	1	10.03.2011 23:20
264	921,875	10235,3125	1	1	12.03.2011 10:56
264	1687,5	12971,25	1	1	14.03.2011 15:53
264	1750	35775,9375	1	1	15.03.2011 16:03
264	3875	21814,375	1	1	19.03.2011 13:05
264	812,5	12309,0625	1	1	20.03.2011 12:51
264	953,125	11436,25	1	1	23.03.2011 23:18
264	1109,375	13881,25	1	1	24.03.2011 22:29
264	984,375	13649,0625	1	1	29.03.2011 23:23
264	1078,125	13381,875	1	1	01.04.2011 23:41
264	875	9791,5625	1	1	03.04.2011 12:56
264	890,625	11020,9375	1	1	04.04.2011 18:15
264	984,375	12478,125	1	1	05.04.2011 16:03

Table A.10: QoS Parameter Values of Service 265

Service ID	Response Time	Throughput	Reliability	Availability	DateTime
265	828,125	16235,625	1	1	07.03.2011 22:04
265	750	41108,4375	1	1	09.03.2011 22:45
265	718,75	6670,3125	1	1	10.03.2011 23:20
265	937,5	9487,1875	1	1	12.03.2011 10:56
265	609,375	8781,25	1	1	14.03.2011 15:53
265	687,5	6240,3125	1	1	15.03.2011 16:03
265	578,125	9095,9375	1	1	19.03.2011 13:05
265	500	8323,4375	1	1	20.03.2011 12:51
265	687,5	16667,5	1	1	23.03.2011 23:18
265	562,5	7364,0625	1	1	24.03.2011 22:29
265	781,25	7204,0625	1	1	29.03.2011 23:23
265	828,125	7435	1	1	01.04.2011 23:41
265	843,75	10508,125	1	1	03.04.2011 12:56
265	796,875	7557,8125	1	1	04.04.2011 18:15
265	640,625	5510,9375	1	1	05.04.2011 16:03

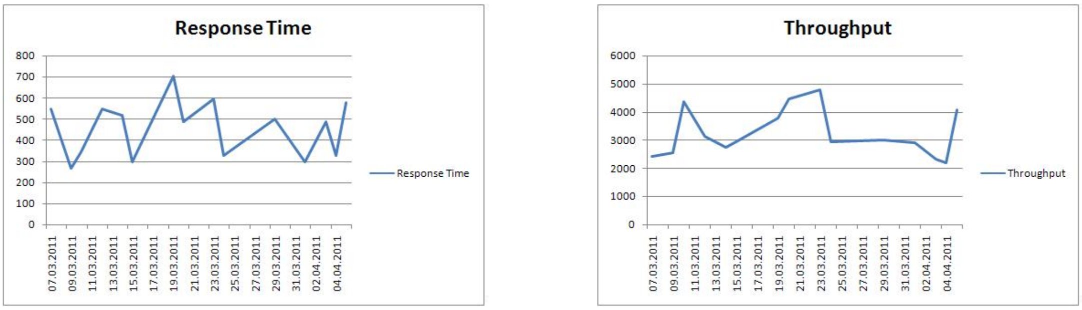


Figure A.1: Response Time and Throughput of Service 250

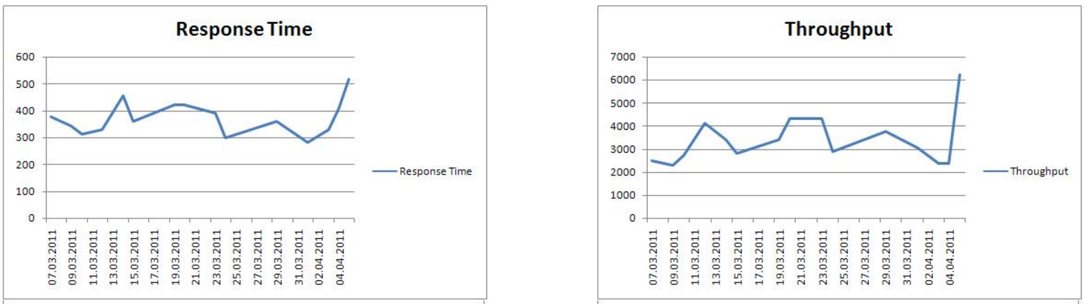


Figure A.2: Response Time and Throughput of Service 251

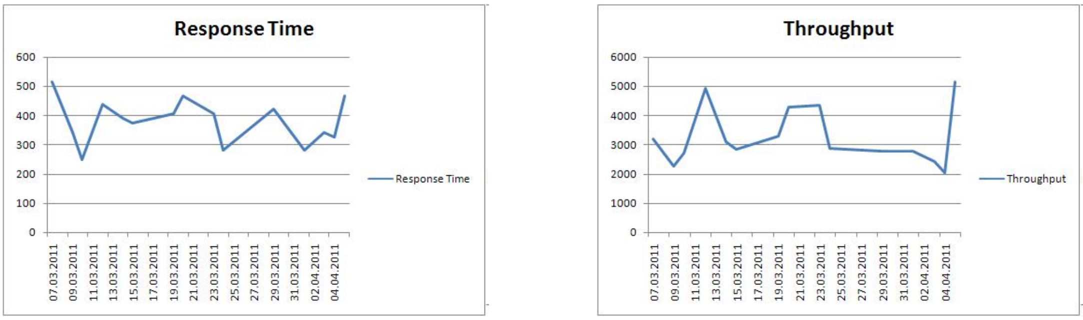


Figure A.3: Response Time and Throughput of Service 252

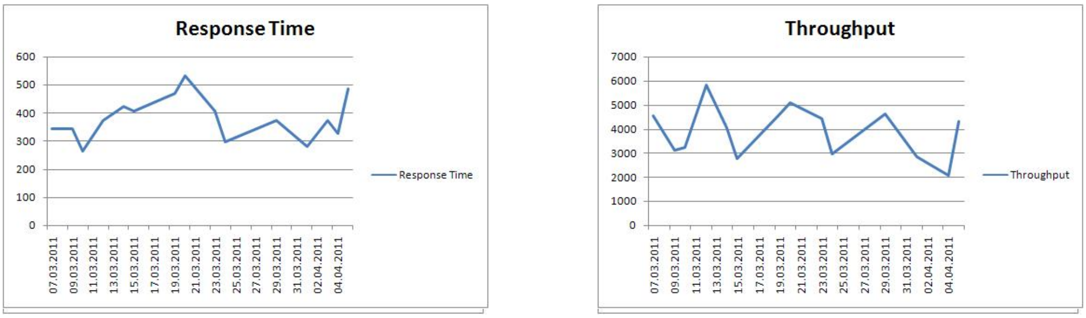


Figure A.4: Response Time and Throughput of Service 253

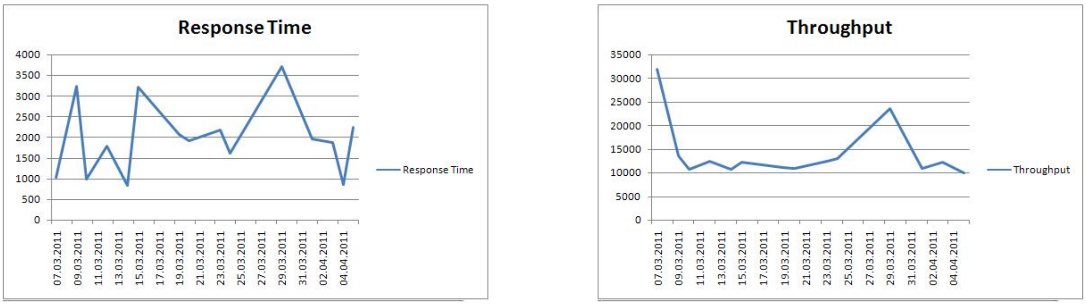


Figure A.5: Response Time and Throughput of Service 254

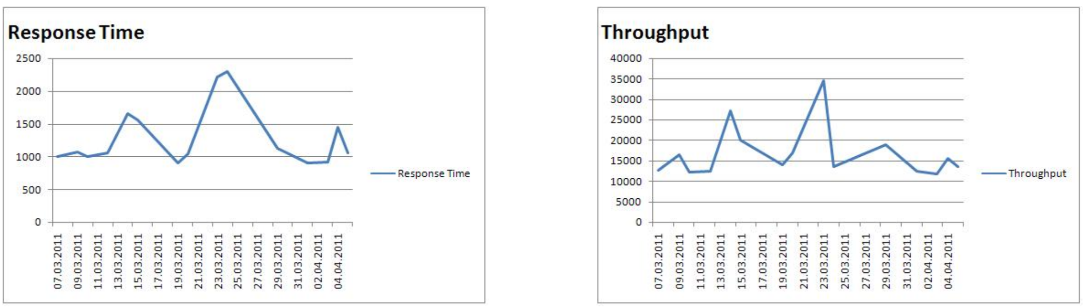


Figure A.6: Response Time and Throughput of Service 255

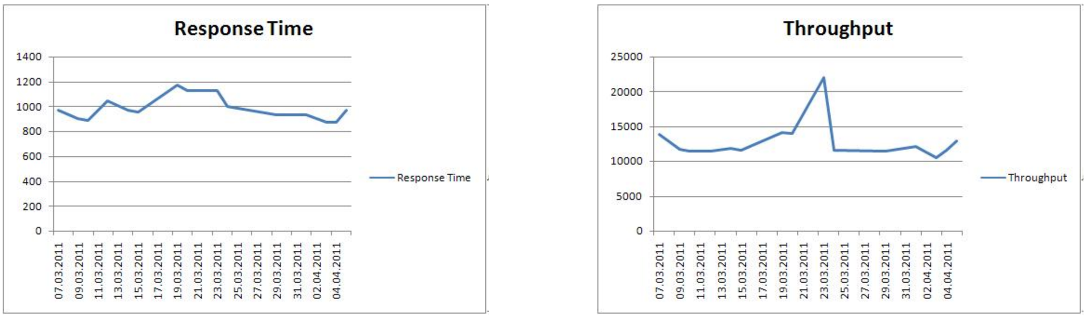


Figure A.7: Response Time and Throughput of Service 256

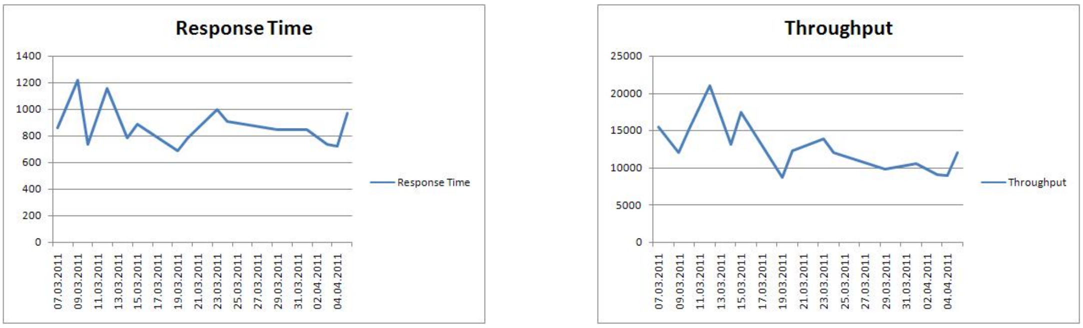


Figure A.8: Response Time and Throughput of Service 263

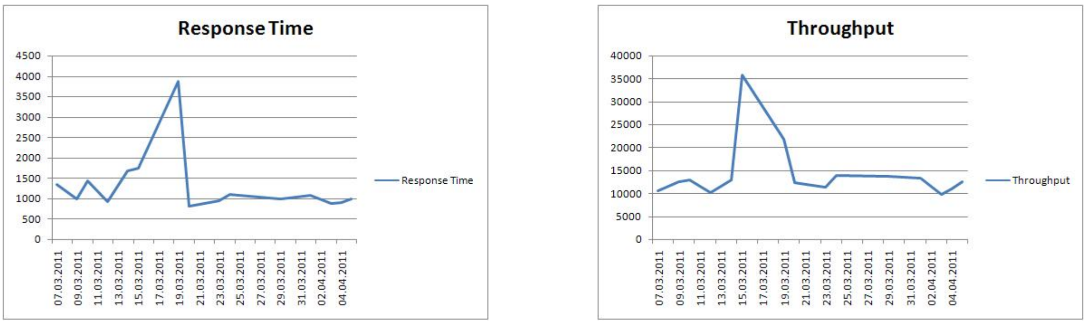


Figure A.9: Response Time and Throughput of Service 264

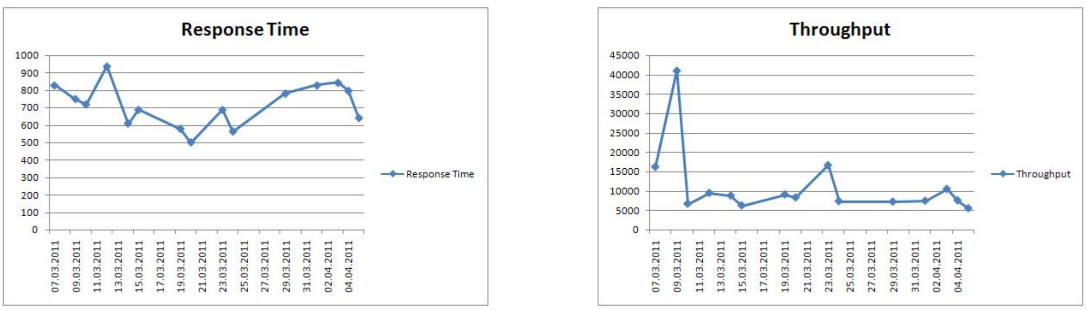


Figure A.10: Response Time and Throughput of Service 265