



QUERY INTERFACE AND QUERY LANGUAGE FOR DOMAIN SPECIFIC WEB  
SERVICE DISCOVERY SYSTEM

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

BY

HİLAL ÖZDİL

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

SEPTEMBER 2011

Approval of the thesis:

**QUERY INTERFACE AND QUERY LANGUAGE FOR DOMAIN SPECIFIC WEB  
SERVICE DISCOVERY SYSTEM**

submitted by **HİLAL ÖZDİL** in partial fulfillment of the requirements for the degree of  
**Master of Science in Computer Engineering Department, Middle East Technical Uni-  
versity** 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 Department, METU

---

Assist. Prof. Dr. Pınar Şenkul  
Computer Engineering Department, METU

---

Prof. Dr. Nihan Kesim Çiçekli  
Computer Engineering Department, METU

---

Assoc. Prof. Dr. Halit Oğuztüzün  
Computer Engineering Department, METU

---

Assoc. Prof. Dr. Erdoğan Dođdu  
Computer Engineering Department, TOBB ETU

---

**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: HİLAL ÖZDİL

Signature :

# ABSTRACT

## QUERY INTERFACE AND QUERY LANGUAGE FOR DOMAIN SPECIFIC WEB SERVICE DISCOVERY SYSTEM

Özdil, Hilal

M.S., Department of Computer Engineering

Supervisor : Assist. Prof. Dr. Pınar Şenkul

September 2011, 80 pages

As the number of the published web services increase, discovery of the web services with the desired functionality and quality is becoming a challenging process. Selecting the appropriate web services among the ones that offer the same functionality is also a challenging task. The web service repositories like UDDI (Universal Description Discovery and Integration) support only the syntactic searches. Quality of service parameters for the published web services can not be queried over these repositories. We have proposed a query language that aims to overcome these problems. It enables its users to query the web services both syntactically and semantically. We also allow the users to specify the quality of service criteria which the desired web services should satisfy. We have developed a graphical query interface to assist the users in query sentence formulation process. The proposed work is developed as a sub-module of the Domain Specific Web Service Discovery with Semantics (DSWSD-S) System. Aforementioned query language and the query interface are explained in detail in this thesis.

Keywords: Web Service, Query Language, Query Interface, Semantic Query, Quality of Service

## ÖZ

### ALANA ÖZGÜ WEB SERVİS KEŞİF SİSTEMİ İÇİN SORGULAMA ARAYÜZÜ VE SORGULAMA DİLİ

Özdil, Hilal

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Yar. Doç. Dr. Pınar Şenkul

Eylül 2011, 80 sayfa

Web servis teknolojisinin gelişmesi ve kullanımının artmasıyla birlikte yayınlanan web servisler arasından istenilen yetenek ve kalitede servislerin bulunması problemi ortaya çıkmıştır. UDDI (Universal Description Discovery and Integration) gibi servis depoları sadece sözdizimsel aramayı desteklemektedir. Ayrıca servislerin kalite puanı bilgileri de bu gibi depolarda sorgulanamamaktadır. Bu çalışmada, web servislerin hem sözdizimsel hem de anlamsal olarak sorgulanabilmesini, ayrıca arama sırasında servis kalite puanı gibi kriterlerin belirtilebilmesini sağlayan bir web servis sorgulama dili ve sorgulama arayüzü geliştirmeyi amaçladık. Sunulan çalışma, DSWSD-S (Domain Specific Web Service Discovery with Semantics) adlı alana özgü web servis keşif sisteminin bir alt parçasıdır. Geliştirilen sorgulama dili ve kullanıcı arayüzü bu tezde detaylandırılmıştır.

Anahtar Kelimeler: Web Servis, Sorgulama Dili, Sorgulama Arayüzü, Anlamsal Sorgu, Servis Kalitesi

*To my lovely family*

## **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 lovely, supportive family and my friends for their belief in me. I also would like to thank Aselsan Inc. for their support on this whole duration and work.



# TABLE OF CONTENTS

ABSTRACT . . . . .	iv
ÖZ . . . . .	v
ACKNOWLEDGMENTS . . . . .	vii
TABLE OF CONTENTS . . . . .	viii
LIST OF TABLES . . . . .	xii
LIST OF FIGURES . . . . .	xiii
CHAPTERS	
1 INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	2
1.2 Contributions . . . . .	2
1.3 Thesis Organization . . . . .	3
2 RELATED WORKS . . . . .	4
2.1 Web Service Query Languages And Structures . . . . .	4
2.1.1 UDDI Query API . . . . .	4
2.1.2 Grid Operation Language (GOL) . . . . .	8
2.1.3 Quality of Service Query Language (QQL) . . . . .	10
2.1.4 Unified Service Query Language (USQL) . . . . .	12
2.2 Graphical User Interfaces For Web Service Querying . . . . .	13
2.2.1 Graphical User Interface In FRWS Approach . . . . .	13
2.2.2 Graphical User Interface For QQL . . . . .	15
3 OVERVIEW OF DOMAIN SPECIFIC WEB SERVICE DISCOVERY WITH SEMANTICS SYSTEM . . . . .	18
3.1 Overall Design of DSWSD-S System . . . . .	18
3.2 Domain-Specific Crawler Layer . . . . .	19

3.3	Domain-Specific Service Discovery Layer . . . . .	21
4	GRAPHICAL USER INTERFACE IN DSWS-D SYSTEM . . . . .	22
4.1	Introduction . . . . .	22
4.2	Search Criteria Definition Panel . . . . .	23
4.3	Result Monitoring Panel . . . . .	27
5	QUERY LANGUAGE IN DSWS-D SYSTEM . . . . .	30
5.1	Why Simplified-USQL? . . . . .	30
5.2	Simplified-USQL (S-USQL) . . . . .	31
5.2.1	Language Structure . . . . .	31
5.2.1.1	S-USQL Request . . . . .	31
5.2.1.2	S-USQL Response . . . . .	32
5.2.2	Language Elements . . . . .	33
5.2.2.1	Removed Elements and Structures . . . . .	33
	<i>priorityLevel</i> Attribute and <i>tPriorityLevel</i> Type	33
	<i>ServiceProvider</i> Element . . . . .	34
	<i>Security</i> Element . . . . .	34
	<i>tCurrency</i> Type . . . . .	34
	<i>tPriceContext</i> Type . . . . .	34
	<i>tProcessingTimeUnit</i> Type . . . . .	34
	<i>ServiceTaxonomy</i> Element and <i>tTaxonomyScheme</i> Type . . . . .	35
	<i>tQoSOperator</i> and <i>tQoS</i> Types . . . . .	35
	<i>tOrderByOption</i> and <i>tOrderByDirection</i> Types	35
	<i>tServiceType</i> Type . . . . .	35
	<i>tNetworkType</i> Type . . . . .	36
	<i>tOperation</i> Type . . . . .	36
	<i>AdditionalOperationProperties</i> Element Group	36
5.2.2.2	Retained Elements and Structures . . . . .	36
	<i>tPercentage</i> Type . . . . .	36
	<i>Error</i> Element . . . . .	36

5.2.2.3	Modified and Additional Elements and Structures . . . . .	37
	<i>basicRequirementAttributes</i> Attribute Group . . . . .	37
	<i>AdditionalServiceProperties</i> Element Group . . . . .	37
	<i>tSyntacticOperator</i> and <i>tSemanticOperator</i> Types . . . . .	37
	<i>tSyntactic</i> and <i>tSemantic</i> Types . . . . .	38
	<i>tParameter</i> Type . . . . .	38
	<i>tWeight</i> Type . . . . .	39
	<i>ServiceName</i> Element . . . . .	39
	<i>ServiceDescription</i> Element . . . . .	40
	<i>ServiceDomain</i> Element . . . . .	40
	<i>Price</i> Element . . . . .	40
	<i>Availability</i> Element . . . . .	40
	<i>Reliability</i> Element . . . . .	41
	<i>ProcessingTime</i> Element . . . . .	41
	<i>Throughput</i> Element . . . . .	41
	<i>QoS</i> Element . . . . .	42
	<i>Operation</i> Elements . . . . .	42
	<i>Service</i> Elements . . . . .	44
	<i>Services</i> Element . . . . .	45
	<i>USQLRequest</i> Element . . . . .	45
	<i>USQLResponse</i> Element . . . . .	46
	<i>USQL</i> Element . . . . .	46
5.3	Simplified USQL (S-USQL) Parser . . . . .	47
6	QUERY EXECUTION PROCESS AND INTERACTION WITH OTHER MODULES OF DSWSD-S . . . . .	50
6.1	Introduction . . . . .	50
6.2	Modules Responsible For Query Execution . . . . .	50
6.2.1	Graphical User Interface Module . . . . .	50
6.2.2	S-USQL Parser Module . . . . .	51
6.2.3	Similarity Degree Calculator Web Service . . . . .	51

6.2.4	Database Operations Handler Web Service . . . . .	51
6.3	Query Execution Process . . . . .	52
7	CASE STUDIES AND ANALYSIS . . . . .	54
7.1	Query with Single Keyword . . . . .	54
7.2	Query with Single Keyword and Domain Specification . . . . .	56
7.3	Query with Custom Quality of Service Criteria . . . . .	57
7.4	Query with Default Quality of Service Criteria . . . . .	59
7.5	Query with Multiple Keywords and Custom Quality of Service Criteria	61
7.6	Query Execution Time Performance Analysis . . . . .	62
8	CONCLUSION . . . . .	67
	REFERENCES . . . . .	69
	APPENDICES	
A	XSD OF S-USQL . . . . .	71
B	SIMILARITY DEGREE CALCULATOR WEB SERVICE . . . . .	79
C	DATABASE OPERATIONS HANDLER WEB SERVICE . . . . .	80

## LIST OF TABLES

### TABLES

Table 7.1	Query Execution Times for Sample Query-1 . . . . .	63
Table 7.2	Query Execution Times for Sample Query-2 . . . . .	63
Table 7.3	Query Execution Times for Sample Query-3 . . . . .	64
Table 7.4	Query Execution Times for Sample Query-4 . . . . .	64
Table 7.5	Query Execution Times for Sample Query-5 . . . . .	65
Table B.1	Operations Offered By Similarity Degree Calculator Web Service . . . . .	79
Table C.1	Operations Offered By Database Operations Handler Web Service . . . . .	80

## LIST OF FIGURES

### FIGURES

Figure 2.1	Relationships Between UDDI Data Structures . . . . .	6
Figure 2.2	General Architecture of FRWS Approach . . . . .	9
Figure 2.3	General Form Of A GOL Sentence . . . . .	10
Figure 2.4	Sample QQL Query . . . . .	11
Figure 2.5	Conceptual Model Of USQL . . . . .	12
Figure 2.6	Graphical User Interface Of Service Grid . . . . .	14
Figure 2.7	Graphical User Interface Of Retrieval Result . . . . .	15
Figure 2.8	First Step Of QQL Query Formulation . . . . .	16
Figure 2.9	Second Step Of QQL Query Formulation . . . . .	16
Figure 2.10	Third Step Of QQL Query Formulation . . . . .	16
Figure 2.11	Fourth Step Of QQL Query Formulation . . . . .	17
Figure 3.1	DSWSD-S System Architecture . . . . .	19
Figure 3.2	DSWSD-S Database Architecture . . . . .	20
Figure 3.3	Domain-Specific Crawler Layer Architecture . . . . .	20
Figure 4.1	Interaction Between GUI Node and Crawler Nodes . . . . .	23
Figure 4.2	Main Page of DSWSD-S GUI . . . . .	23
Figure 4.3	Keyword Specification Widget . . . . .	24
Figure 4.4	Matching Operator Choice Widget . . . . .	25
Figure 4.5	QoS Parameters Panel . . . . .	25
Figure 4.6	Domain Ontologies Tabs . . . . .	27
Figure 4.7	Maximum Number of Results Limitation Widget . . . . .	28

Figure 4.8	User Action Widgets . . . . .	28
Figure 4.9	Results Table . . . . .	29
Figure 4.10	No Result Warning . . . . .	29
Figure 5.1	USQL Request Structure . . . . .	32
Figure 5.2	S-USQL Request Structure . . . . .	32
Figure 5.3	USQL Response Structure . . . . .	33
Figure 5.4	S-USQL Response Structure . . . . .	33
Figure 5.5	tPercentage . . . . .	36
Figure 5.6	Error . . . . .	37
Figure 5.7	basicRequirementAttributes . . . . .	37
Figure 5.8	AdditionalServiceProperties . . . . .	38
Figure 5.9	tSyntacticSemanticOperators . . . . .	38
Figure 5.10	tSyntacticSemantic . . . . .	39
Figure 5.11	tParameter . . . . .	39
Figure 5.12	tWeight . . . . .	39
Figure 5.13	ServiceName . . . . .	40
Figure 5.14	ServiceDescription . . . . .	40
Figure 5.15	ServiceDomain . . . . .	40
Figure 5.16	Price . . . . .	41
Figure 5.17	Availability . . . . .	41
Figure 5.18	Reliability . . . . .	41
Figure 5.19	ProcessingTime . . . . .	41
Figure 5.20	Throughput . . . . .	42
Figure 5.21	QoS . . . . .	42
Figure 5.22	RequestOperation . . . . .	43
Figure 5.23	ResponseOperation . . . . .	43
Figure 5.24	RequestService . . . . .	44
Figure 5.25	ResponseService . . . . .	44

Figure 5.26 ResponseServices . . . . .	45
Figure 5.27 SUSQLRequest . . . . .	45
Figure 5.28 SUSQLResponse . . . . .	46
Figure 5.29 SUSQL . . . . .	46
Figure 6.1 Interaction Between Modules . . . . .	52
Figure 7.1 Sample Query-1 . . . . .	54
Figure 7.2 Response for Sample Query-1 . . . . .	55
Figure 7.3 Screen for Sample Query-1 . . . . .	55
Figure 7.4 Sample Query-2 . . . . .	56
Figure 7.5 Response for Sample Query-2 . . . . .	56
Figure 7.6 Screen for Sample Query-2 . . . . .	57
Figure 7.7 Sample Query-3 . . . . .	57
Figure 7.8 Response for Sample Query-3 . . . . .	58
Figure 7.9 Screen for Sample Query-3 . . . . .	58
Figure 7.10 Sample Query-4 . . . . .	59
Figure 7.11 Response for Sample Query-4 . . . . .	60
Figure 7.12 Screen for Sample Query-4 . . . . .	60
Figure 7.13 Sample Query-5 . . . . .	61
Figure 7.14 Response for Sample Query-5 . . . . .	61
Figure 7.15 Screen for Sample Query-5 . . . . .	62
Figure 7.16 Query Execution Time Performance Graph . . . . .	66



# CHAPTER 1

## INTRODUCTION

As the number of the published web services increase, they become more prevalent nowadays. Accordingly, discovery of the web services with the desired functionality and quality is becoming a challenging process. Selecting the appropriate web services among the ones that offers the same functionality is also a rocky road.

In the literature, most of the web service discovery systems collect the web services in a central registry. As the number of the published web services increase, these registries become overloaded and a scalability issue occurs. Another drawback of these solutions is that, they are incompatible with each other. There are heterogeneous service registries like UDDI[9] or ebXML[12] and all of them provides different specifications. Therefore, the web service discovery work becomes more complicated for the web service consumers. The users need to spend too much time to visit different user interfaces of different solutions. In addition, he/she needs to understand the usage of their interface. The user is also expected to know or learn quickly the query language or structure that these solutions provide for web service discovery.

UDDI (Universal Description, Discovery and Integration) and ebXML (Electronic Business using eXtensible Markup Language) are the most popular service registries. Both of them provide ways to publish and consume services, but their specifications are limited. These registries have the following drawbacks:

- They only support keyword-based searches and do not have a rich query interface to enable the user to specify his/her needs.
- They do not control the life-cycle of the services they advertised, they are disconnected

from them. Therefore, these registries do not know whether the services advertised are alive or reliable to use.

- These registries do not offer quality of service information, therefore selecting the appropriate service among the ones offering same functionality becomes a hard task.

To overcome these problems, a web service discovery framework is designed, namely Domain Specific Web Service Discovery with Semantics (DSWSD-S) [7,8]. The system aims to make the web service discovery process more efficient, faster, reliable and scalable. It consists of the web crawlers each of which specialized on specific domains by means of ontology definitions. The system also offers user-friendly graphical user interface to enable its users to specify their requirements that the desired web services should satisfy.

Within the work done in this thesis, we propose the aforementioned graphical user interface. In addition, we developed a web service query language to be used in DSWSD-S System.

## **1.1 Motivation**

While designing the DSWSD-S System, we found that the web service discovery solutions proposed in the literature does not consider the user factor. Therefore, the main motivation of this thesis is to design a search interface and query language to enable the users to specify their requirements easily. The search tools provided in the literature are not user-friendly and they are not easy-to-use. The work proposed in this thesis, directs the users to specify the search criteria and requirements, which the desired web services should satisfy, while encapsulating them from the technical details of the query language used in background. The users construct web service queries easily without any knowledge on the query language implemented in system via the guidance provided by the graphical user interface.

## **1.2 Contributions**

The proposed work in this thesis is developed as a part of the Domain Specific Web Service Discovery with Semantics (DSWSD-S) System. The main contributions of this thesis are as follows:

- A user-centric query interface for web service discovery is described.
- A web service query language is developed for DSWSD-S System.
- Comparison between service query languages and query mechanisms are discussed.
- The DSWSD-S System and the interaction of the proposed work with other parts of the system is given.

### **1.3 Thesis Organization**

This thesis is organized as follows:

Chapter 2 presents the related work on web service selection and web service query languages.

Chapter 3 explains the overall design of Domain Specific Web Service Discovery with Semantics (DSWSD-S) System and its architectural layers.

In Chapter 4 the graphical user interface designed for DSWSD-S System is described in detail.

Chapter 5 is about the Simplified-Unified Query Language (S-USQL). The language elements and structures are detailed in this chapter.

In Chapter 6 we explain the interactions between the proposed work and the other parts of the DSWSD-S System in the scope of a query execution process.

Chapter 7 represents the usage of the proposed query language and the graphical user interface with samples.

The thesis ends up in the conclusion part given in Chapter 8.

## **CHAPTER 2**

### **RELATED WORKS**

Effective web service querying topic is studied by several researchers in the literature. However, a very small part of them studied on the user-interaction in web service querying process. In this chapter, we will explain the studies based on the web service selection and the query languages defined for this purpose. In addition to that we will describe the studies that emphasized on graphical user interface designs that enable the users to search for web services. In Section 2.1, query languages and structures defined in the literature to query the services are described. Section 2.2 explains the graphical user interface works proposed for efficient service search and selection.

#### **2.1 Web Service Query Languages And Structures**

##### **2.1.1 UDDI Query API**

Universal Description, Discovery and Integration (UDDI) [9] is a standardized directory service where businesses can publish and discover web services. UDDI is used by the service providers to advertise the services they offer and it is used by the service consumers to discover the web services that satisfy their requirements.

With the UDDI specifications the followings are defined:

- SOAP APIs to enable the UDDI users to query and to publish web services.
- XML Schema schemata of the registry data model and the SOAP message formats.
- SOAP APIs' WSDL definitions .

- UDDI registry definitions of identifiers and categories which can be used to identify and categorize the UDDI registrations.

UDDI uses WSDL to describe interfaces to web services. Additionally, cross platform programming features are addressed by adopting SOAP, known as XML Protocol messaging specifications.

Conceptually, a business can publish three types of information into a UDDI repository:

1. **White Pages:** Contains the basic contact information about a business. It is composed of business name, address, contact information, and unique identifiers. This information enables requestors to find the published web services based upon the business identification.
2. **Yellow Pages:** Contains the information which describes a web service using categorizations in other words using taxonomies. This information enables requestors to find the published web services based upon its categorization.
3. **Green Pages:** Contains the technical information which defines the behaviors and offered operations of a web service. This information contains pointers to the grouping information of the web services and where the web services are located.

As we mentioned before, UDDI defines a set of XML Schema definitions that describe the data models and SOAP message formats used by different specification APIs. The specifications include:

- UDDI Replication
- UDDI Operators
- UDDI Programmer's API
- UDDI Data Structures

Inside the UDDI Programmer's API, there two different API specification: UDDI Publishing API and the UDDI Inquiry API. UDDI Publishing API is used to create, store or update

information about a web service registered in a UDDI registry. UDDI Inquiry API is used to discover and obtain the details of a web service registered in a UDDI registry.

In the scope of this thesis, we have emphasized on the UDDI Inquiry API. Before going into details of the API, explaining the data structures briefly would be helpful to understand the operations defined in the API. The relationships between the UDDI data structures are given in Figure 2.1.

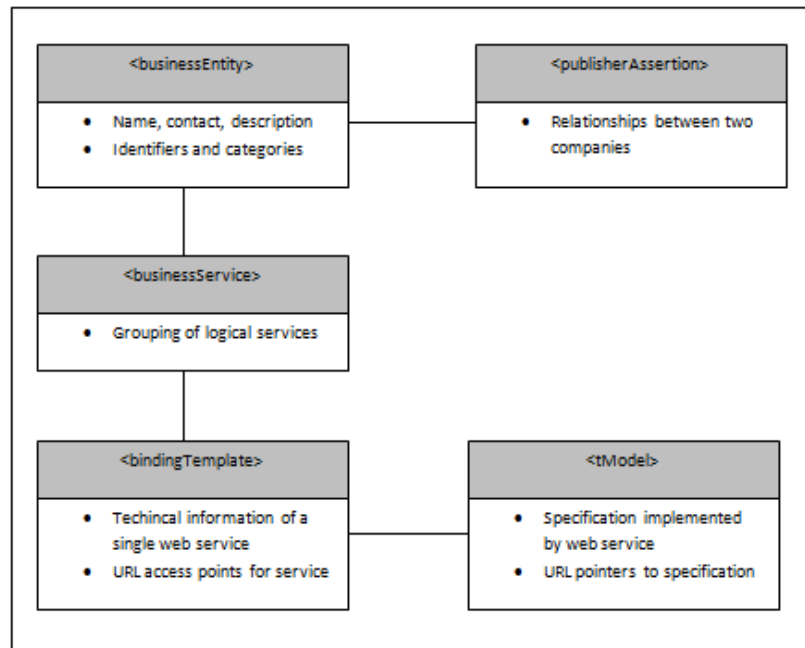


Figure 2.1: Relationships Between UDDI Data Structures

- *businessEntity*: structure composes of a business's basic information which are business contact information, categorization, identifiers, descriptions and relationships to other businesses.
- *publisherAssertion*: structure indicates public relationships between two *businessEntity* structures. A relationship between two *businessEntity* structures becomes visible to the public when both businesses created the same relationship as separate *publisherAssertion* documents.
- *businessService*: structure is used to represent the set of services that are offered by a business. It is contained inside *businessEntity*. One *businessEntity* structure may contain one or more *businessService* structures. The *businessService* structure is a reusable

structure, basically it can be used by different *businessEntity* structures.

- *bindingTemplate*: structure contains the technical descriptions and the access point URL of a service, but it does not contain the service's specifications. A *businessService* composes of one or more *bindingTemplate* structures.
- *tModel*: structure is used to define the behavior of a web service. A *tModel* contains the needed information to interact with a web service. The *tModel* structure does not provide the web service's specification directly, it contains pointers to the locations of the actual specifications.

The SOAP messages which are defined within UDDI Inquiry API start with the "find" word. Below the operations that the API offers for web service discovery are given:

- *find\_binding*: This operation is used to find the *bindingTemplate* elements in the UDDI. It returns a structure named *bindingDetail* which holds a list of the resulting *bindingTemplate* elements.
- *find\_business*: This operation is used to find the *businessEntity* elements. It returns a structure named *businessList* which holds a list of the resulting *businessEntity* elements.
- *find\_relatedBusinesses*: This operation is used to find *businessEntity* elements that have a relationship with the *businessEntity* specified as parameter. It returns a structure named *relatedBusinessesList* which holds a list of the resulting related *businessEntity* elements.
- *find\_service*: This operation is used to find the *businessService* elements in the UDDI. It returns a structure named *serviceList* which holds a list of the resulting *businessService* elements.
- *find\_tModel*: This operation is used to find the *tModel* elements in the UDDI. It returns a structure named *tModelList* which holds a list of the resulting *tModel* elements.

All of the operations listed above, retrieves some common arguments: authentication information, qualifiers for the search and name. Here is the important argument is the qualifiers which tells us the UDDI's search capability. Some of the qualifiers which can be used in a UDDI query are:

- *approximateMatch*
- *caseInsensitiveMatch*
- *caseSensitiveMatch*

As can be seen, the qualifiers are all defined for syntactic matching of the keywords. UDDI does not offer semantic matching feature and related qualifiers. In addition to this, UDDI does not offer any information about the quality of service parameters about the services registered to it.

### **2.1.2 Grid Operation Language (GOL)**

Hai Zhuge and Jie Liu [5] are proposed an approach and named it as Flexible Retrieval of Web Services (FRWS). FRWS is composed of three layers: the graphical user interface that helps the users in query sentence production, an SQL-like query language and finally the UDDI repositories and an orthogonal service space structure over these repositories, that they call as Service Grid. In this Service Grid space, they established the multi-valued specialization relationships between services.

The proposed Service Grid model can be expressed in 3 dimensions: classification-type, category and content. The classification-type dimension reflects the service classification standards where the category dimension indicates the detail-classified hierarchy related to a classification type. Finally, functionality of the services are indicated through the content dimension. The similarity degrees between services, the similarity degrees between tModels and the multi-valued specialization relationships are kept in Service Grid structure.

There are five different specialization relationship types between tModels and between web services. These are:

1. Identical-specialization relationship
2. Partial-specialization relationship
3. Extension-specialization relationship
4. Revision-specialization relationship



## 5. Non-specialization relationship

The details and the conditions that differ these relationships are out of the scope this thesis, so they will not be detailed here. The detailed formulations of the conditions for these relationships can be found at [5].

The general architecture of the FRWS approach is given in Figure 2.2.

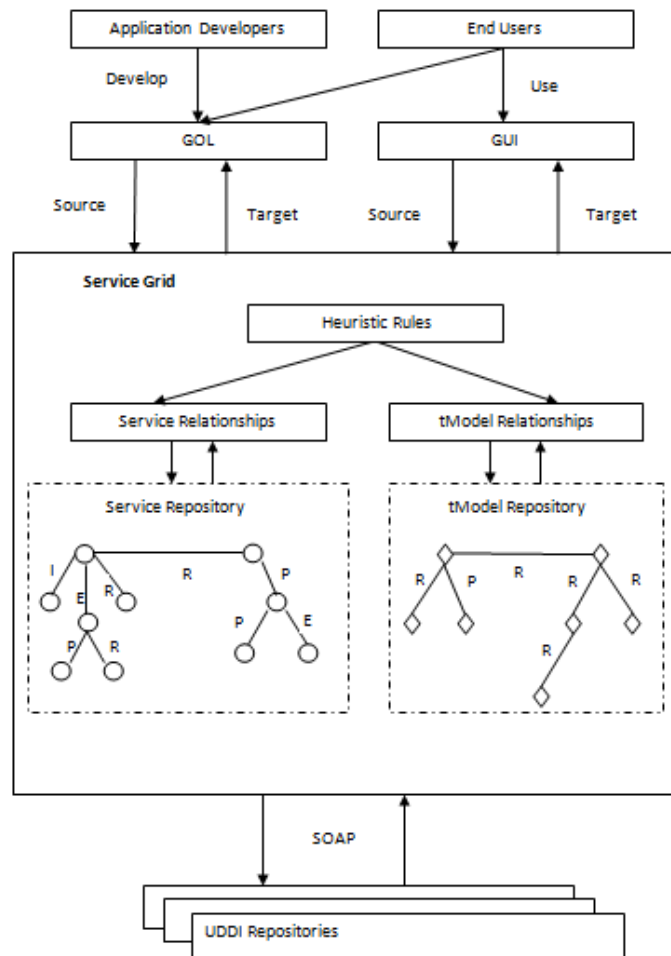


Figure 2.2: General Architecture of FRWS Approach

To retrieve the desired web services in FRWS approach, Hai Zhuge and Jie Liu defined an SQL-like, XML based query language and named it as Grid Operation Language (GOL). The general form of a GOL sentence is given in Figure 2.3.

Here, the Service-Repository element defines the registered web services. The registered

```

<GOL Statement>:: = Get [ALL | DISTINCT]<Target Service>
    FROM<Service-Repository>
    [WHERE<Condition-Expressions>]
    [GROUP BY<Element>] [HAVING<Condition-Expressions>]
    [SORT BY <Element>][ASC | DESC];
<Target Service>:: = Web Service | tModel
<Service-Repository>:: = Local | Universal
<Condition-Expressions>:: = <Condition-Expression>AND | OR<Condition-Expression>
<Condition-Expression>:: = WS1 → name = <name-expressions> | WS1 →I→ WS2 | WS1 →E→ WS2 | WS1 →P→ WS2 | WS1
→R→ WS2 | WS1 →→ WS2 | sd(WS1, WS2) > β

```

Figure 2.3: General Form Of A GOL Sentence

web services can be either local registered services or universal registered services. The Condition-expressions element is a Boolean expression. It is composed of the source name, the multi-valued specialization relationships and the similarity degree between the source and the target.

### 2.1.3 Quality of Service Query Language (QQL)

In [6] Delnavaz Mobedpour, Chen Ding and Chi-Hung Chi propose a web service query language that takes quality of service parameters in consideration. The proposed QoS Query Language (QQL) enables its users to specify the quality of service parameters both numerically and linguistically.

According to the QQL definition a QoS query is represented as a six-tuple: *qID*, *uID*, *sbTime*, *timeConstraints*, *qosConstraints*, *dataSource*. If we look over these fields in detail:

- *qID* refers to the query identifier.
- *uID* refers to requestor identifier.
- *timeConstraints* contains different constraints based on the time. These are *invocation start date*, *end date*, *duration of each usage*, and *frequency of usage*. *Invocation start date* field shows the date when the service is invoked firstly and *invocation end date* field represents the date when the service invoked lastly. *Duration of usage* field refers to the amount of time how long the service is used in each invocation. Finally, the *frequency of usage* field is defined to show how often the service is invoked during the period from invocation start date to invocation end date.

- *dataSource* refers to the data source from where the selection will be done.
- *qosConstraints* contains the constraints about the quality of service choices of the requestor. By default, it is defined with eight properties: *name*, *type*, *unit*, *tendency*, *preference*, *relaxation order*, *weight* and *values*. *Name* attribute shows the name of the quality of service parameter. *Type* refers to the data type of the parameter where *Unit* show the measurement unit of the quality of service parameter. *Tendency* refers to the user's choice on the parameter. If the *Tendency* is positive then this means the requestor prefers a higher value and if it is negative then this means the user prefers a lower value. *Preference* attribute shows the user's choice about the order of the quality of service parameters. If the requestor specified *N* quality of service parameters, then this means there *N* different *Preference* value where *1* refers to the most preferred one. *Relaxation order* refers to the relaxation order choice of the requestor. Its values are complement of the *Preference* values. Basically, if a parameter is preferred mostly then this means it should be relaxed lastly. The *Weight* attribute represents the preference order. It is not specified by the requestor explicitly, instead it is calculated by normal-

```

<?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>
  </QoSConstraints>
  <dataSource>provider</ dataSource >
</QoSQuery>

```

Figure 2.4: Sample QQL Query

izing the *Preference* value into [0,1] range. Finally, the *Values* attribute represents the user's requirement on the attribute's value.

A sample QQL query sentence is given in Figure 2.4.

By enabling its users to specify the quality of service criteria both numerically and linguistically, the QQL language differs from the similar works in the literature with this feature. However, QQL has some drawback too. As the UDDI Query API and GOL, QQL supports only syntactic queries, it does not offer needed structures for semantic searches.

### 2.1.4 Unified Service Query Language (USQL)

Aphrodite Tsalgatidou and Michael Pantazoglou defined the Unified Service Query Language (USQL) with the proposed work in [1,2,3]. USQL is the base language that we use and extend in this thesis.

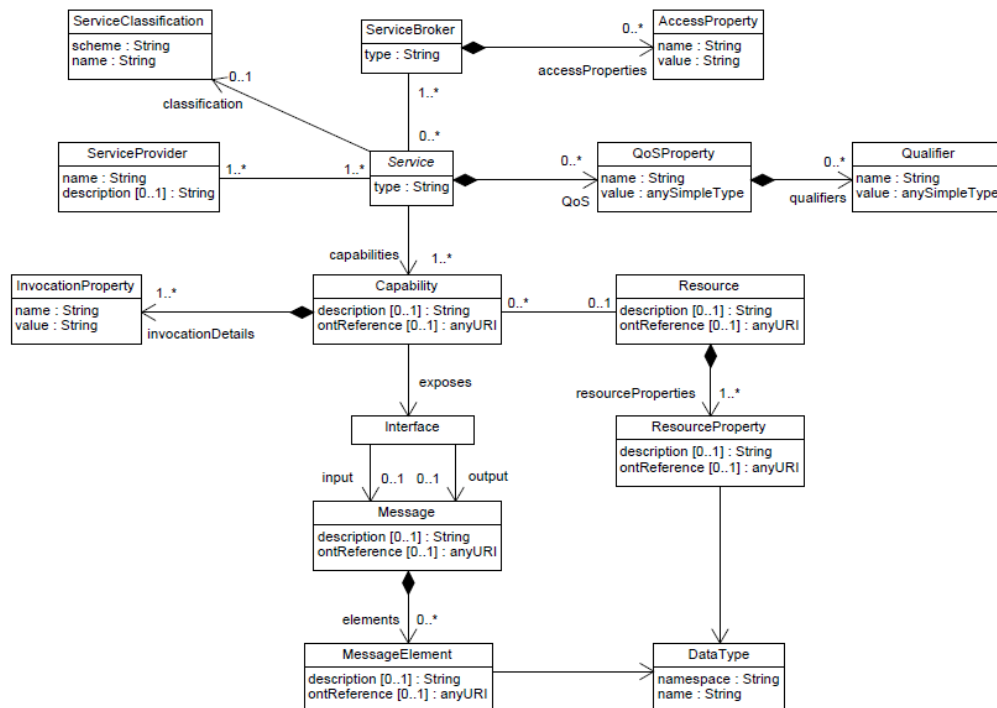


Figure 2.5: Conceptual Model Of USQL

USQL is an XML-based query language that allows its users to query heterogenous services

(web services, grid services, peer-to-peer services) syntactically and semantically. The language offers query and response structures independent from the service type and the technical details of the service registry where the services are registered. In addition to this, USQL language definition supports specifying the quality of service criteria.

Due to its support in querying heterogeneous service types it allows to specify the service provider information. It also allows to define the domain information in order to make semantic matching. In addition to this, USQL enables its users to define input and output search criteria for the operations for the desired services.

The specification of the USQL is based on the conceptual model given in Figure 2.5. According to the conceptual model, a service provider provides one or more services, which are published on one or more registries. The registries are called as brokers in their conceptual model. A service belongs to a classification and offers one or more capabilities. The capabilities expose an interface defining the input messages they take and output messages they return. Basically, the capability elements are used to define the functional properties of the services.

Besides the functional properties, a service can have the non-functional properties as quality of service parameters. Each quality of service parameter of a service has a name, that informs of its semantics, and a value.

The specification of the USQL is given in [3]. However, the language is defined with general usage purposes and it does not have a corresponding query interface or query parser engine. In this thesis, we have defined a sub-set of USQL and implemented a query parser engine for the Domain Specific Web Service Discovery with Semantics (DSWSD-S) System. The elements and structures defined within the USQL specification are explained and compared to the ones defined in our query language, S-USQL, in Chapter 5 in detail.

## **2.2 Graphical User Interfaces For Web Service Querying**

### **2.2.1 Graphical User Interface In FRWS Approach**

In [5], a graphical user interface for FRWS approach to retrieve the desired web services is proposed. The needed source and target information, and the multi-valued specialization

relationship is retrieved from the user via this graphical user interface. A corresponding GOL statement is formulated with these information. The interface for Service Grid is given in Figure 2.6.

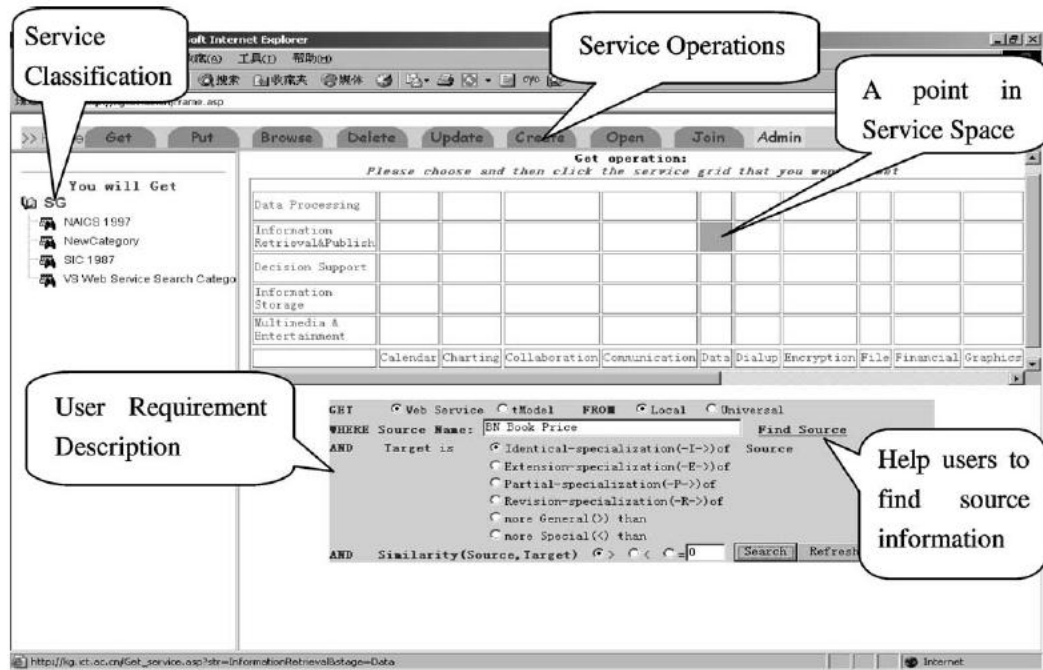


Figure 2.6: Graphical User Interface Of Service Grid

The web service query steps via this interface are as follows:

1. The user specifies the source type if it will be the web service or the tModel.
2. The user selects the registered service type, whether they are registered locally or universally.
3. The user specifies the source name. The interface also helps him/her to find the source name by clicking the “Find Source” link.
4. The user chooses the desired multi-valued specialization relationship between the source and the target.
5. The user specifies the desired similarity degree between the source and the target.
6. The user starts the execution via clicking on the “Search” button.

When the user clicks on the “Search” button, the inputs he/she entered are gathered to formulate a GOL statement. The Service Grid searches and retrieves the desired web services or tModels from the selected registry and lists to the user in the interface given in Figure 2.7.

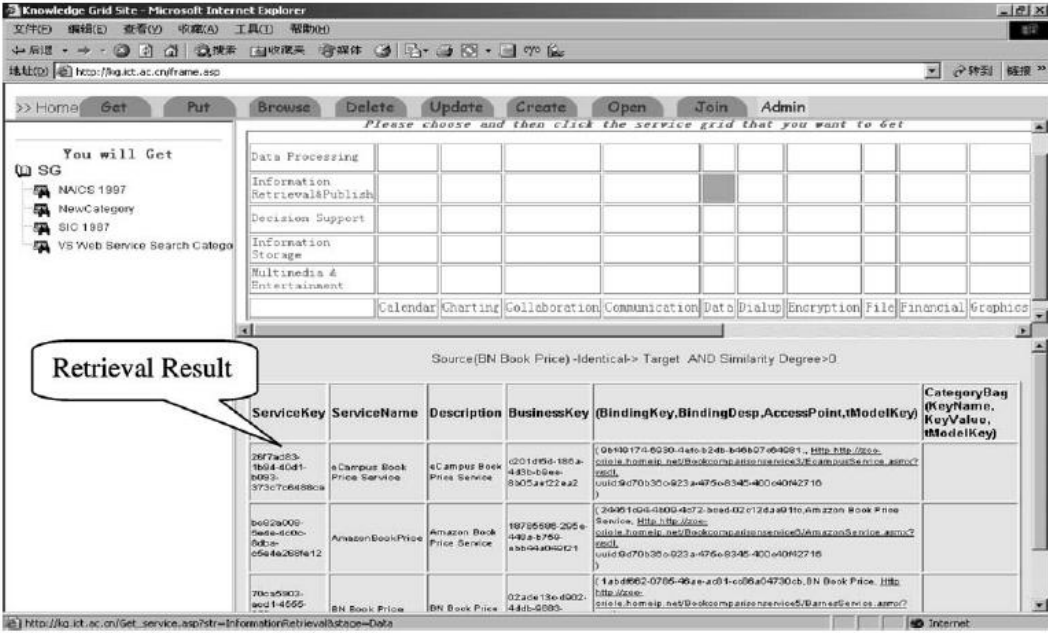


Figure 2.7: Graphical User Interface Of Retrieval Result

The FRWS approach, queries the desired web services and the tModels, whose criteria are specified as GOL statements, over the UDDI search API. As in the UDDI repositories, this approach supports only the syntactic queries. It also does not consider the quality of service parameters which play very important role in order to differentiate the resulting services with the same functionality. In addition to this, the proposed graphical user interface forces users to specify a source web service or tModel to trigger a query process. The user must specify a source and the relationship of it with the desired target. However, this is a challenging task for a user and makes the proposed graphical user interface a hard-to-use one.

2.2.2 Graphical User Interface For QQL

In [6], Delnavaz Mobedpour, Chen Ding and Chi-Hung Chi propose a graphical user interface to enable the users to formulate QQL queries easily. They aimed to avoid putting too much burden on the user to define the query sentences. Therefore, they propose the graphical

user interface given in Figures 2.8, 2.9, 2.10 and 2.11 to guide the user in query sentence production.



Figure 2.8: First Step Of QQL Query Formulation

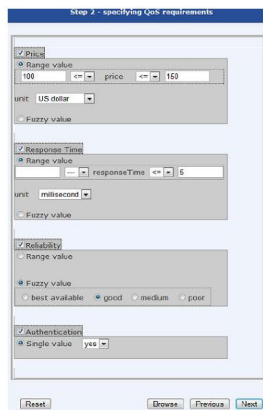


Figure 2.9: Second Step Of QQL Query Formulation

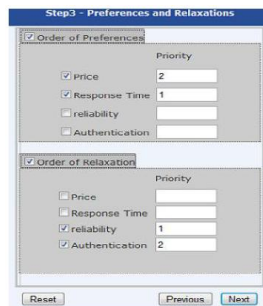


Figure 2.10: Third Step Of QQL Query Formulation

By using the interface given in Figure 2.8, the user specifies the quality of service parameters that he/she intended the desired web services should satisfy. For every quality of service parameter the user chooses, a *QoSConstraint* element is created in the query. By using the interface given in Figure 2.9, the user defines the data types and the values for the quality of service parameters he/she selected in Figure 2.8. After defining their types and values, he/she



defines the preference by using the interface given in Figure 2.10.

After specifying all these values, the user defines the time constraints via the interface given in Figure 2.11.



Figure 2.11: Fourth Step Of QQL Query Formulation

The proposed user interface is designed as a user-friendly and easy-to-use interface. It guides its users in query formulation so the user does not have to deal with the technical details of the QQL. Basically, the graphical user interface has a user-centric design and differs from the similar works done in the literature with this feature.

## **CHAPTER 3**

### **OVERVIEW OF DOMAIN SPECIFIC WEB SERVICE DISCOVERY WITH SEMANTICS SYSTEM**

The graphical user interface and query language, which are presented in this thesis, are developed as a part of the Domain Specific Web Service Discovery with Semantics (DSWSD-S) System [7,8]. In this chapter, DSWSD-S System and its architectural layers will be described. In Section 3.1 overall design of the DSWSD-S System is given. Section 3.2 explains the domain-specific crawler layer and Section 3.3 is about domain-specific service discovery layer.

#### **3.1 Overall Design of DSWSD-S System**

DSWSD-S System is a web service discovery system that consists of domain-specific subsystems. By saying domain-specific subsystems, we mean that the subsystems are specialized for an ontology definition. Each domain-specific subsystem crawls over the web, discovers the web services related to its own ontology and keeps these discovered web services in its local database. The system aims to provide the capabilities listed below:

- Controlling the life-cycle of the web services to keep them up-to-date
- Making the search process on different registries easier by encapsulating them on a common search interface
- Being scalable to keep up with the increasing number of web services and domains
- Providing automated quality of service calculation

- Providing both syntactic and semantic web service discovery queries

The DSWSD-S System is composed of two layers: the domain-specific crawler layer and the domain-specific service discovery layer. These layers will be detailed in Sections 3.2 and 3.3.

The overall architecture of the system is given in Figure 3.1.

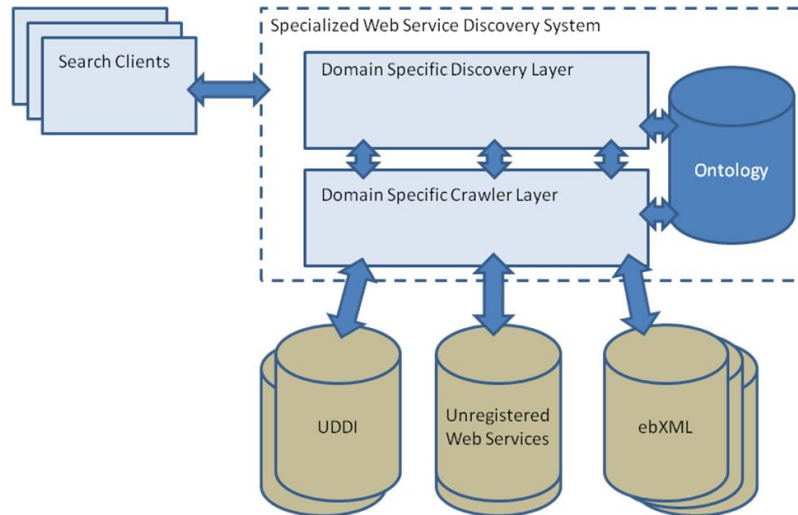


Figure 3.1: DSWSD-S System Architecture

### 3.2 Domain-Specific Crawler Layer

The main responsibility of a domain-specific crawler layer is to construct the local database, whose architecture is given in Figure 3.2, of the domain-specific subsystem. Local database generation process starts with the acquisition of the web service addresses from the web services' WSDL files. After that, the content of the web service is downloaded and controlled if it is related to the subsystem's own ontology. If so, the crawler validates the web service by calling its operations with appropriate parameters to see whether the web service is alive or not. Next, the validated web service is sent to the extraction module. The extraction module deals with the semantic annotation process. With semantic annotation, it is determined whether the web service fulfills the capability it advertised or not. After the verification phase, verified web services are added to the crawler's local database. The execution of domain-specific crawler layer is given in Figure 3.3.

In order to keep the web services up-to-date, aforementioned crawling process continues in a

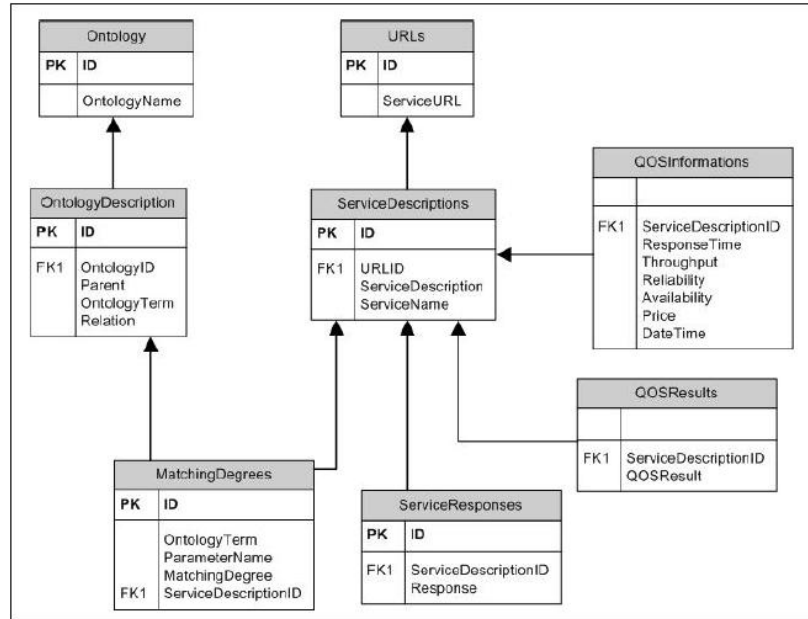


Figure 3.2: DSWS-D Database Architecture

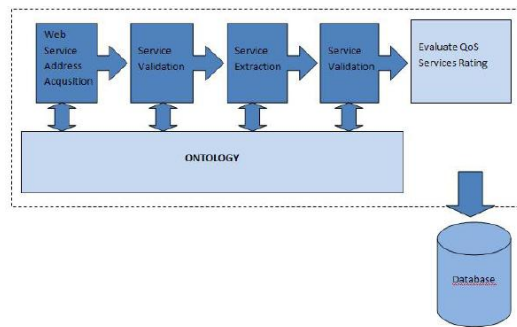


Figure 3.3: Domain-Specific Crawler Layer Architecture

cycle. The crawlers update their local databases by adding the newly published web services, updating the ones' status kept in the database and removing the web services that lost their aliveness.

The quality of service calculation also takes place in this layer. An automated quality of service calculation module traverses the local database in order to produce the quality scores of the web services kept. There are five different quality of service criteria handled in DSWS-D S System: availability, reliability, response time, throughput and price. To produce the total quality score from these parameters, the quality calculation module uses an algorithm that gives weights to all of these quality aspects. It also considers the age factor. The explanation

of the algorithm is out of the scope of this thesis, therefore we will not mention about its details.

### **3.3 Domain-Specific Service Discovery Layer**

The domain-specific service discovery layer enables the users to query the DSWS-D System by providing a graphical user interface. This graphical user interface designed in a form that it allows the user to specify the keyword(s) he/she looks for, the domain information that he/she is interested in and the service quality criteria that the desired services should have. The keyword(s) and the other information gathered from the user are transformed into an S-USQL query sentence. Then this query sentence is passed to a parser module that can interpret the requirements in the query. The parser decides the crawler that will be searched according the parameters specified in the query sentence. After reading the web services fulfilling the user's requirements from the corresponding crawler's database, it sends the results to the graphical user interface to be shown to the user.

This layer has a central structure. Basically, while there are more than one crawler layers, there is only one discovery layer. The domain-specific service discovery layer knows all the necessary information to access all the existing crawlers in the system. This layer and the modules contained in it are the scope of this thesis. And they are explained in detail in the following Chapters 4 and 5.

## CHAPTER 4

### GRAPHICAL USER INTERFACE IN DSWSD-S SYSTEM

#### 4.1 Introduction

The graphical user interface is the only interaction point with the user in DSWSD-S System. The user can find the web services that match both syntactically and semantically with the keywords he/she entered via the graphical user interface. The user is also enabled to specify the Quality of Service (QoS) criteria that the searched services are intended to obey.

In this thesis, we developed a user-friendly query interface, to enable the users to reach the web services easily. It encapsulates the technical details of the query execution process from the user while directing him/her to enter the needed information for a web service search.

As mentioned in Section 3 there are several crawler nodes in DSWSD-S System. All of these crawler nodes run on dedicated computers. Each of the nodes has its own local database that keeps the validated and verified web services that match the crawler's own domain (ontology). However in DSWSD-S System there is only one graphical user interface node. It communicates with whole crawler group. The interaction between the graphical user interface node and crawler nodes is given in Figure 4.1.

In the system's initialization process, the graphical user interface knows only one crawler access information. It gets this information from a configuration file. After that it communicates with this crawler node and retrieves all other crawler's access information too. Then it retrieves the domain information from all these nodes and shows to the user so that the user can query the domain that he/she is interested in. The graphical user interface node also periodically reads the crawlers' access information to keep the query interface up-to-date. With this periodical polling mechanism, addition or deletion of a domain-specific crawler is

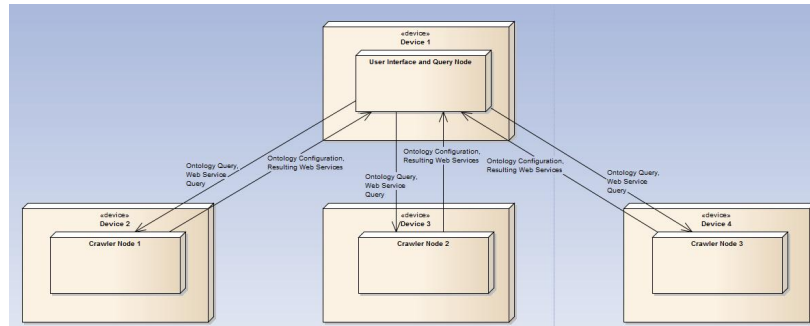


Figure 4.1: Interaction Between GUI Node and Crawler Nodes

reflected to the query interface immediately.

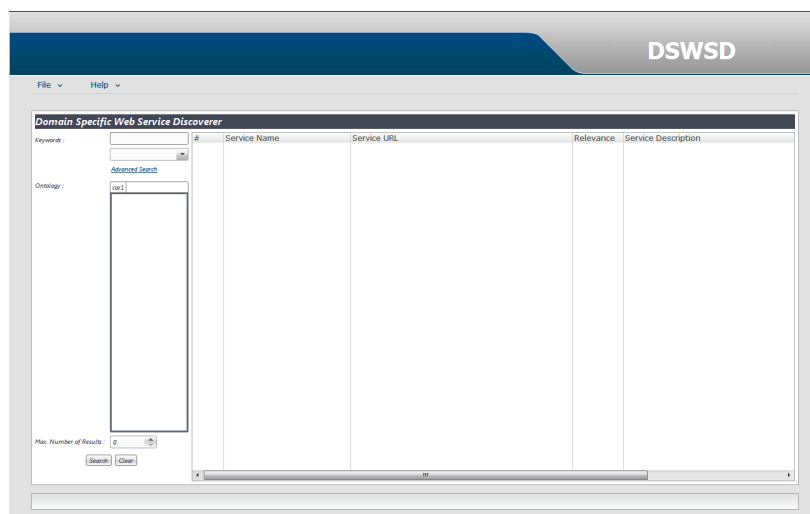


Figure 4.2: Main Page of DSWS-D GUI

When the user enters the DSWS-D System, he/she is presented with the screen given in Figure 4.2. The left panel is designed to retrieve the parameters and criteria from user, which will be used in query production. The right one is the panel where the user can monitor the search results. These panels will be explained in detail in the following sections.

## 4.2 Search Criteria Definition Panel

The left panel in the graphical user interface is the search criteria and parameters definition panel. If we examine this panel in detail, as shown in Figure 4.3, there is a text box widget at the upper part of the panel in order to allow the user to enter the keywords he/she searches for

in a web service's name.

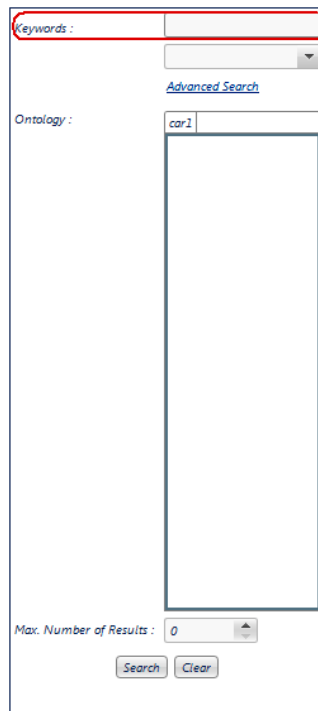
The image shows a web interface for keyword specification. At the top, there is a text input field labeled "Keywords:" with a red border. Below it is a dropdown menu. A blue link labeled "Advanced Search" is positioned below the dropdown. Underneath the link is another text input field labeled "Ontology:" containing the text "car1". At the bottom of the widget, there is a "Max. Number of Results:" label followed by a small input field containing the number "0". Below this are two buttons: "Search" and "Clear".

Figure 4.3: Keyword Specification Widget

Just below this keyword box, there is a choice widget to enable the user to specify the operation used in matching process as shown in Figure 4.4. By using this widget user can specify if he/she wants his/her keyword should be matched syntactically or semantically or in both way. If the user does not choose any choice from this widget, the keyword will be matched both syntactically and semantically.

Just below this keyword box, there is an “*Advanced Search*” link. Whenever the user clicks this link, a hidden panel, quality of service parameters panel, is opened. In Figure 4.5, this QoS panel is shown.

By using this panel, the user can choose whether to use the default QoS parameter values or to specify his/her custom values. There are five different quality of service criteria handled in DSWSD-S System: availability, reliability, response time, throughput and price.

Availability of a web service represents the accessibility of it. Basically, it refers to whether a web service is ready for use or not at the invocation time. In case of a call, if it returns any response this means that the web service is available. If does not return any response or raises



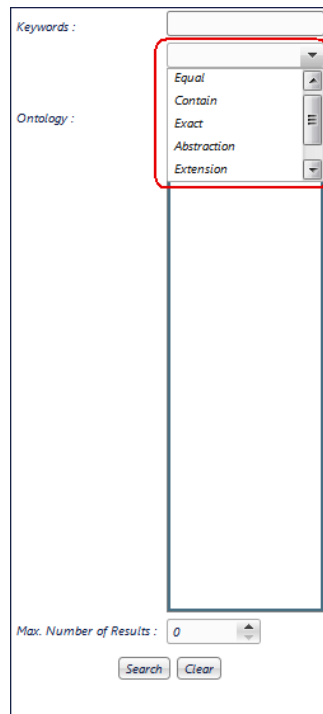


Figure 4.4: Matching Operator Choice Widget

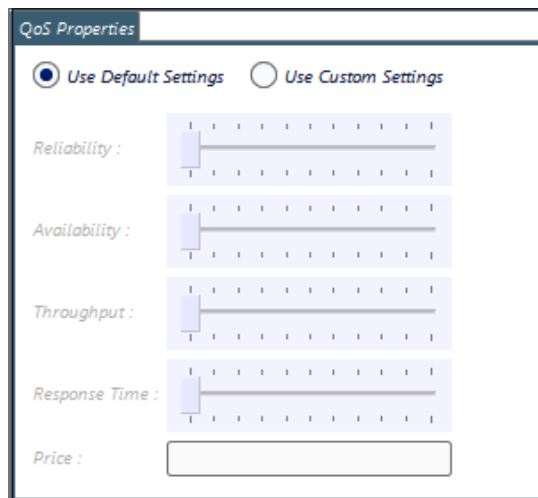


Figure 4.5: QoS Parameters Panel

any exception then it means that the invoked service is unavailable.

Reliability is the quality of service parameter of a web service that represents the stability of the web service. If it returns the same result when invoked with the same parameter at different time periods then it means the web service is reliable. If it responses with different results

every time for the same parameter than it means the web service is not reliable. Reliability of a service can be thought as the consistency of its quality.

Response Time of a web service represents the desired processing time of the operations that the web service offers. Meanly, it is the amount of the delay between times the service is invoked and it returns response.

Throughput of a web service represents the total number of calls that the web service can handle concurrently.

Finally the price of a web service refers to the desired cost that should be paid for calling it. Actually, the price is not a real quality of service parameter, but it is also used to filter the web services found. Therefore its value is retrieved from the user by using the same panel with the quality of service parameters mentioned above.

Except from the price parameter, all other parameters are defined in the range [0-100]. Therefore, the user is allowed to give the parameter values in these ranges by using scale type widgets in graphical user interface. The parameter values are used as weights of them in quality of service score calculation algorithm. As mentioned before, the user is enabled to use whether default weights for these parameters or to define his/her own weights. If the user chooses to use the default weights, then all of the parameters' weights are considered to be equal as having the value 50. In cases where the user did not click on "*Advanced Search*" link, means does not intend to specify quality of service parameters, the weights are set to their default values again.

Just below the quality of service parameters definition panel, which becomes visible by pressing "*Advanced Search*" link, the domains (ontologies) associated with the crawler nodes in the system are pointed out as tabs as shown in Figure 4.6.

Neither the keyword specification nor the ontology choice is mandatory. If the user specifies both the keyword and the domain information, then the matching degree between the keyword and the domain ontology is calculated. The web services that belong to the selected domain and that satisfy the calculated matching degree are retrieved from the database and presented to the user. If he/she does not specify the keyword but chooses the domain ontology, then all of the web services kept in local database of the crawler node are retrieved and presented to the user. The third option is that he/she enters the keyword but does not specify the domain

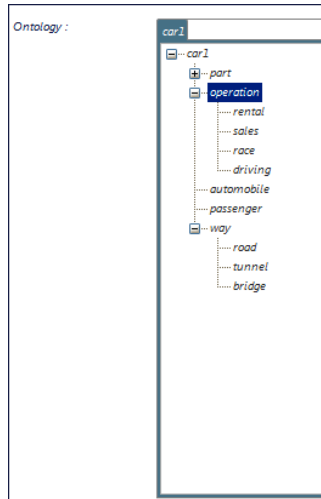


Figure 4.6: Domain Ontologies Tabs

ontology to search for. In this case, the keyword is matched to all existing crawler nodes' domain ontologies, and the crawler node whose domain ontology has the highest matching degree is selected for searching. Actually there is one more option as neither keyword nor domain ontology selection is done. As can be guessed, in this case no search is done.

The user can limit the maximum number of best matching results to be shown by using the widget at the lower part of the left panel given in Figure 4.7. However if he/she limits the maximum number of results as 0, then the system interprets this as “bring all the services found”.

At the bottom of the left panel, buttons for the actions that the user can trigger are located as shown in Figure 4.8. When the user clicks on the “*Search*” button at the bottom part of the left panel, the query execution is triggered with all the parameters he/she entered. And if the user clicks on the “*Clear*” button, all the parameters and criteria he/she entered is cleared.

### 4.3 Result Monitoring Panel

The results that are found after a query execution are shown to the user at the right panel of the graphical user interface. Table-type widget is thought to be more appropriate and chosen to present the resulting web services. The web service name, web service URL, the web service description information and the matching degree of the keyword and the found web

Keywords :   
  
[Advanced Search](#)

Ontology :

Max. Number of Results : 0

Figure 4.7: Maximum Number of Results Limitation Widget

Keywords :   
  
[Advanced Search](#)

Ontology :

Max. Number of Results : 0

Figure 4.8: User Action Widgets

service (relevance of the keyword with the web service) are presented to the user in the related columns of the table as shown in Figure 4.9.

#	Service Name	Service URL	Relevance	Service Description
1	showSearchFormData	http://www.geckobyte.com/webservices/Selector.aspx	0.895	System.Object showSearchFormData(String searchT)
2	GetTyreWheelSize	http://www.proflityrecenter.nl/5/rtbc4f53ggymv5SguirdSovj/7/tyreSizeService	0.895	CascadingDropDownNameValue[] GetTyreWheelS
3	GetWheelVehicleSize	http://www.proflityrecenter.nl/5/rtbc4f53ggymv5SguirdSovj/7/tyreSizeService	0.895	CascadingDropDownNameValue[] GetWheelVehic
4	getWheelsByVehicleAndTire	http://twg.corpronetwork.com/tools.aspx	0.895	Wheel[] getWheelsByVehicleAndTire(String Usern
5	GetMyVotersRoll	http://www.esactmobile.co.za/interactive/interactivewebservice.aspx	0.895	System.Xml.XmlNode GetMyVotersRoll(String Userf
6	getWheelsByPlusSize	http://twg.corpronetwork.com/tools.aspx	0.895	Wheel[] getWheelsByPlusSize(String Username, Str
7	getWheelSearchString	http://www.geckobyte.com/webservices/GeckRestricted.aspx	0.895	System.String getWheelSearchString(String BrandN
8	showWheelPads	http://www.geckobyte.com/webservices/Selector.aspx	0.895	System.Object showWheelPads(User thisUser, Deale
9	GetFrontWheelStyleDropDownValues	http://www.avtireking.com/DesktopModules/TK_KitFinder/TK_KitService.aspx	0.895	CascadingDropDownNameValue[] GetFrontWheelSt
10	GetFrontWheelSizeDropDownValues	https://www.avtireking.com/DesktopModules/TK_KitFinder/TK_KitService.aspx	0.895	CascadingDropDownNameValue[] GetFrontWheelSi
11	GetRearWheelSizeDropDownValues	https://www.avtireking.com/DesktopModules/TK_KitFinder/TK_KitService.aspx	0.895	CascadingDropDownNameValue[] GetRearWheelSi
12	getWheel_rimWidth	http://www.geckobyte.com/webservices/USSelectorDB.aspx	0.895	eBayDropDown[] getWheel_rimWidth(String rimDia
13	getWheel	http://www.geckobyte.com/webservices/BelleTireGeckData.aspx	0.895	BelleWheel getWheel(int32 dealerID, int32 geckWh
14	GetRearWheelStyleDropDownValues	http://www.avtireking.com/DesktopModules/TK_KitFinder/TK_KitService.aspx	0.895	CascadingDropDownNameValue[] GetRearWheelSt
15	showSearchForm	http://www.geckobyte.com/webservices/Selector.aspx	0.895	System.Object showSearchForm(Dealer thisDealer, I
16	getVehicleSearchData	http://www.geckobyte.com/webservices/Selector.aspx	0.895	System.Object getVehicleSearchData(String searchT
17	getWheelsByVehicle	http://twg.corpronetwork.com/tools.aspx	0.895	Wheel[] getWheelsByVehicle(String Username, Stri
18	getWheelsByBrandName	http://twg.corpronetwork.com/tools.aspx	0.895	Wheel[] getWheelsByBrandName(String Username,
19	CastVoteByRollID	http://www.esactmobile.co.za/interactive/interactivewebservice.aspx	0.895	int32 CastVoteByRollID(String Username, String Pa
20	GetWheelModelList	http://www.wheelspecs.com/WheelModels.aspx	0.895	System.String[] GetWheelModelList(String prefixTex
21	Insert_Record	http://gis.hqgid.org/UT_Connection/Service.aspx	0.895	System.String Insert_Record(String id, String input_c
22	ShowWheelSearchInfo	http://www.geckobyte.com/webservices/Selector.aspx	0.895	System.String ShowWheelSearchInfo(User thisUser,
23	getWheelSearchString	http://www.geckobyte.com/webservices/Infopla.aspx	0.895	System.String getWheelSearchString(String BrandN

Figure 4.9: Results Table

If no result web service can be found after a query execution, then the user is informed about the situation via a warning text as given in Figure 4.10.

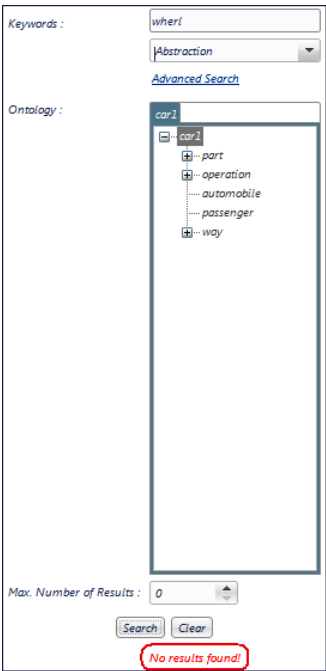


Figure 4.10: No Result Warning

## CHAPTER 5

### QUERY LANGUAGE IN DSWSD-S SYSTEM

#### 5.1 Why Simplified-USQL?

In the literature there are several web service query languages as mentioned in Chapter 2. However, all of them have some certain drawbacks. Some of them do not support semantic searches while some of them do not offer quality of service criteria specification. In order to overcome these problems and satisfy the needs of the DSWSD-S System, we have developed the Simplified-USQL (S-USQL) language.

The reasons why we have developed a USQL-based [1,2,3] new language is as follows:

- USQL is defined for general usage purposes and it does not have a corresponding language parser engine.
- USQL aims to discover different types of services like web services, grid services and peer-to-peer services, but in DSWSD-S System we are only interested in discovery of web services.
- In DSWSD-S System, some of the informations which can be specified in a USQL query is not provided. Therefore, we have defined a sub-set of the USQL language.

By defining S-USQL language, we also aimed:

- To make the graphical query interface, that is offered within the DSWSD-S System, replaceable with another graphical user interface that complies with the defined language.
- To make the web service crawler engines interoperable with a standard query language.

- To make the DSWS-D System scalable. The graphical user interface makes queries and receives responses with the defined language, therefore another system that complies with the S-USQL can be easily integrated to DSWS-D System.

## **5.2 Simplified-USQL (S-USQL)**

Simplified-USQL, briefly S-USQL, is a subset of the USQL language, which is mentioned in Section 2.1.4. Actually, it is a specialized subset to satisfy the needs of the DSWS-D System. Like its super-set USQL, S-USQL is an XML-based language definition. All of the language elements are defined via an XSD file.

While creating the S-USQL language, some of the element definitions from USQL, which are not taken into consideration in DSWS-D System, are removed from the language specification. In addition to this, some new elements are also added, like new quality of service parameters.

### **5.2.1 Language Structure**

S-USQL is used for producing queries and responses in a web service discovery process. The information specified by the requestor of the web services are transformed into related structures defined in S-USQL. This information, consists of the search criteria and the parameters that the resulting web services should satisfy. S-USQL Request element is used for the formulation of the queries and the S-USQL Response element is used for representing the resulting web services. The structures of these elements are explained in the following sections.

#### **5.2.1.1 S-USQL Request**

The specification of USQL Request message abstract model is given as Figure 5.1.

As seen in the Figure 5.1, the USQL Request message comprises two parts: projection and criteria parts. Projection part enables the requestor to specify the information that he/she wants to see in the resulting services. The criteria part enables him/her to specify the search criteria and parameters that will be applied in a query process.

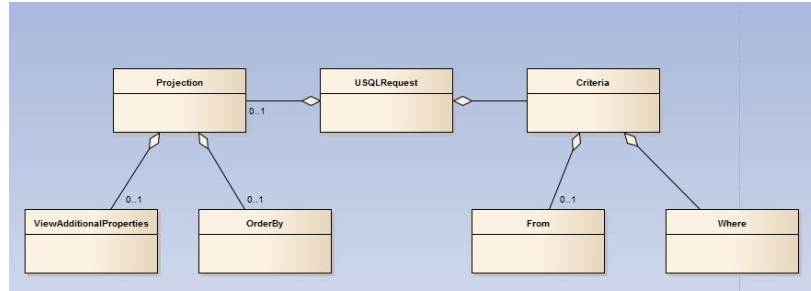


Figure 5.1: USQL Request Structure

In S-USQL, criteria part is taken into language definition while the projection part is removed. Due to the static representation in DSWS-D System, the user can not tell the system which fields he/she wants to display in resulting web services. The fields to be shown to the user from the result web services are pre-defined. So the abstract model of the S-USQL Request element is structured as in Figure 5.2.

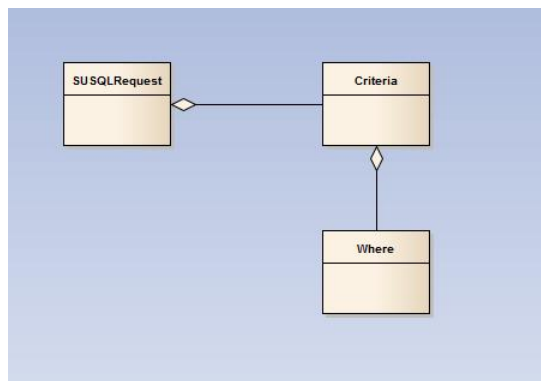


Figure 5.2: S-USQL Request Structure

### 5.2.1.2 S-USQL Response

The specification of USQL Response message abstract model is given as Figure 5.3.

As seen in the Figure 5.3, the USQL Response message comprises two parts: resulting services or error parts. These two parts are alternative to each other. An USQL Response message can contain only one of them at the same time. Resulting services part is the part conveying all the result services found in respective query process. The error part is the one that is used when the query is not completed successfully.



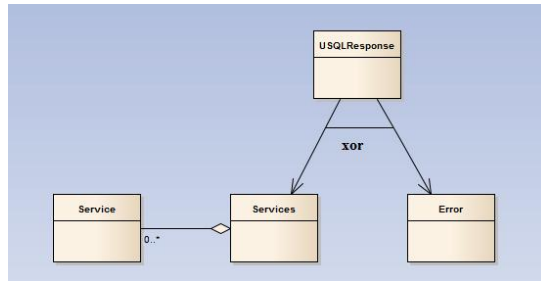


Figure 5.3: USQL Response Structure

In S-USQL, both of these part definitions are covered. Therefore, the S-USQL Response is structured as in Figure 5.4.

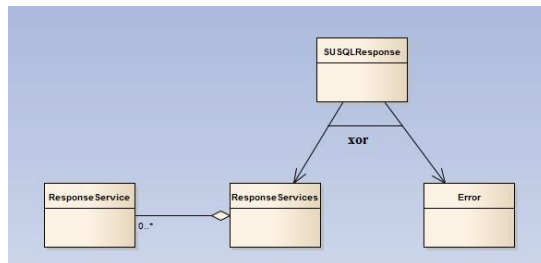


Figure 5.4: S-USQL Response Structure

## 5.2.2 Language Elements

In this section, we will explain the S-USQL language elements and structures in detail. We will also discuss how these elements are combined to produce query sentences and their corresponding responses. When constructing a sub-set of the USQL language as S-USQL, some of the element and attribute definitions that are not handled in DSWS-D System are removed. We will firstly explain these elements and attributes and give the reasons of the removal. After that we will explain the retained and newly added elements.

### 5.2.2.1 Removed Elements and Structures

**priorityLevel Attribute and tPriorityLevel Type** *priorityLevel* attribute is used to indicate the priority of an element in USQL. It is defined in the type *tPriorityLevel*. This type contains two values as low and high. Due to the fact that we do not prioritize any search criteria in

DSWSD-S System, *priorityLevel* element and corresponding *tPriorityLevel* type definitions are not contained in S-USQL definition.

***ServiceProvider Element*** *ServiceProvider* element is used to allow the users to specify the provider of the services he/she looks for in USQL. However, in DSWSD-S System crawler nodes' local databases, service provider information is not kept. Therefore, the user is not enabled to specify such a search criteria in a web service query. Because of this reason *ServiceProvider* element's definition is removed from S-USQL language.

***Security Element*** *Security* element is used to specify the desired security protocol for the requested service in USQL. However, in DSWSD-S System security protocol information is not taken into consideration while searching and extracting the web services. Therefore the *Security* element's definition is removed from S-USQL language.

***tCurrency Type*** *tCurrency* type is used to specify the desired currency of the price of the requested service in USQL. However, the currency information of the web services is not kept in DSWSD-S System crawler nodes local databases. Therefore, *tCurrency* type is not included in S-USQL definition.

***tPriceContext Type*** *tPriceContext* type is used to indicate the amount of the time that the price should be paid for the service in USQL. Actually, it is an enumeration that consists of *per call*, *per day*, *per week*, *per month* and *per year* values. In DSWSD-S System we do not take this context information into consideration and removed the type definition from S-USQL language.

***tProcessingTimeUnit Type*** *tProcessingTimeUnit* type is used to specify the unit of the desired response time of the requested web service in USQL. However, in DSWSD-S System we always use the milliseconds unit. In addition to this, instead of retrieving a concrete response time value from the user, we retrieve a weight value for the response time parameter. We use this weight in quality of service score calculation. Therefore, *tProcessingTimeUnit* type definition is not included in S-USQL language.

***ServiceTaxonomy Element and tTaxonomyScheme Type*** *ServiceTaxonomy* element is used to enable the users to specify the taxonomies for the requested services in USQL. *tTaxonomyScheme* type keeps the taxonomies that can be specified in USQL. The type definition is an enumeration that consists of *North American Industry Classification System 1997 Release*, *North American Industry Classification System 2002 Release*, *United Nations Standard Products and Service Code System Version 7.3*, *United Nations Standard Products and Service Code System Version 6.0501* and *United Nations Standard Products and Service Code System Version 3.1* values. In DSWS-D System crawler nodes' local databases, service taxonomy information is not kept. Therefore, the user is not allowed to specify such a search criteria in a web service query. Because of this reason *ServiceTaxonomy* element and corresponding *tTaxonomyScheme* type definitions are not kept in S-USQL language.

***tQoSOperator and tQoS Types*** *tQoSOperator* is used to specify the operator that will be used for quality of service parameter values. It is defined as an enumeration also and consists of *equal*, *not equal*, *greater*, *less*, *equal or greater* and *equal or less* values. However, in DSWS-D System quality of service calculation algorithm is designed to work with *equal* operator. Therefore, retrieving the operator choice from the user is not meaningful in our system. Due to this reason, *tQoSOperator* operator type definition is removed from S-USQL definition. *tQoS* type is used to define generic quality of service parameters. It contains the attributes that have the *tQoSOperator* operator values mentioned. Due to the operator's removal, this parameter type definition is also removed from S-USQL language.

***tOrderByOption and tOrderByDirection Types*** The user is allowed to specify the desired ordering of the resulting services via using *tOrderByOption* and *tOrderByDirection* types in USQL definition. But in DSWS-D System, we present the resulting services in a descending order of their relevance with the keywords searched. Therefore, this ordering type definitions are not contained in S-USQL language.

***tServiceType Type*** The user is enabled to specify the desired service type by using *tServiceType* type in USQL definition. The type is defined as an enumeration and consists of *WebService*, *GridService* and *P2PService* values. DSWS-D System is designed to find only the web services, it does not support different service types. Therefore, we excluded this type

definition from S-USQL language.

***tNetworkType* Type** *tNetworkType* type is designed to list the different peer-to-peer network types in USQL definition. In current version of it, it only enumerates the *JXTA* network. As mentioned in previous paragraph, different service types and their corresponding network types are not considered in DSWS-D System. Therefore, this type definition is removed from S-USQL language.

***tOperation* Type** In USQL, *tOperation* type is defined to indicate a response operation's type. It contains the *name* element and the *degreeOfMatch* attributes. In S-USQL, we have removed this type and contained these fields directly in the *ResponseOperation* element that will be explained in following paragraphs.

***AdditionalOperationProperties* Element Group** *AdditionalOperationProperties* is an element group to indicate a response operation's additional properties. It composes of *Price*, *Availability*, *Reliability*, *ProcessingTime* and *Security* elements. However, in DSWS-D System, we do not offer such an information to the user. Therefore, this element group definition is removed from S-USQL definition.

### 5.2.2.2 Retained Elements and Structures

***tPercentage* Type** *tPercentage* is a generic simple type definition for defining percentages. This type definition is included in S-USQL without any modification. The structure of the type is given in Figure 5.5.

```
<xs:simpleType name="tPercentage">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="1" />
  </xs:restriction>
</xs:simpleType>
```

Figure 5.5: *tPercentage*

**Error Element** *Error* element is used to declare the erroneous situations in a query process. In USQL definition, it contains *code* and *desc* attributes where the former represents the error code and the latter one represents the error description. We use this element in S-USQL without any modification. The *Error* element definition is given in Figure 5.6.

```
<xs:element name="Error">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="code" type="xs:string" />
      <xs:element name="desc" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 5.6: Error

### 5.2.2.3 Modified and Additional Elements and Structures

**basicRequirementAttributes Attribute Group** *basicRequirementAttributes* is an attribute group that composes of the common attributes in elements used to define the search criteria. In USQL definition, it contains the *nullAccepted*, *minDegreeOfMatch* and *priorityLevel* attributes. Depending on the deletion of the *priorityLevel* attribute, it is modified in S-USQL and constructed as given in Figure 5.7.

```
<xs:attributeGroup name="basicRequirementAttributes">
  <xs:attribute name="nullAccepted" type="xs:boolean" use="optional" default="false" />
  <xs:attribute name="minDegreeOfMatch" type="tns:tPercentage" use="optional" default="1.0" />
</xs:attributeGroup>
```

Figure 5.7: basicRequirementAttributes

**AdditionalServiceProperties Element Group** *AdditionalServiceProperties* is an element group that comprises the additional properties that can be contained in a response service. In USQL definition, it comprises the removed *ServiceProvider* and *Security* elements in addition to *ServiceDescription*, *Price*, *Availability* and *Reliability* elements. Due to the removal of *ServiceProvider* and *Security* elements and existence of an element named *QoS* that groups *Price*, *Availability* and *Reliability* and some other quality of service elements, the *AdditionalServiceProperties* element definition is modified as given in Figure 5.8.

```

<xs:group name="AdditionalServiceProperties">
  <xs:sequence>
    <xs:element ref="tns:ServiceDescription" minOccurs="0" />
    <xs:element ref="tns:QoS" minOccurs="0" />
  </xs:sequence>
</xs:group>

```

Figure 5.8: AdditionalServiceProperties

***tSyntacticOperator* and *tSemanticOperator* Types** In USQL definition, there are two different operator type definitions for parameters: *tSyntacticOperator* and *tSemanticOperator*. *tSyntacticOperator* is used to specify the operator type whenever the syntactic matching will be used. It is an enumeration that consists of *equal*, *contain*, *not equal* and *not contain* values. *tSemanticOperator* is used to specify the operator type whenever the semantic matching will be used. It is an enumeration that consists of *exact*, *abstraction*, *extension* and *sibling* values. In S-USQL we have combined these two operator types in one type named *tSyntacticSemanticOperators*. It is defined as an enumeration consisting of all of the values listed for *tSyntacticOperator* and *tSemanticOperator*. After this combination, our *tSyntacticSemanticOperators* type is constructed as given in Figure 5.9.

```

<xs:simpleType name="tSyntacticSemanticOperators">
  <xs:restriction base="xs:string">
    <xs:enumeration value="equal"/>
    <xs:enumeration value="contain"/>
    <xs:enumeration value="exact"/>
    <xs:enumeration value="abstraction"/>
    <xs:enumeration value="extension"/>
    <xs:enumeration value="sibling"/>
  </xs:restriction>
</xs:simpleType>

```

Figure 5.9: tSyntacticSemanticOperators

***tSyntactic* and *tSemantic* Types** *tSyntactic* and *tSemantic* types are generic parameter types defined in USQL and used to indicate whether a parameter will be matched syntactically or semantically. In USQL, some elements as *ServiceName* or *ServiceDescription* is defined in the type *tSyntactic*. However, in DSWS-D System, we match these elements both syntactically and semantically. Therefore, we have combined these two types in a one and named it as *tSyntacticSemantic*. This is also the reason of the operator combination explained in previous paragraph. The type definition of *tSyntacticSemantic* is given in Figure 5.10.

```

<xs:complexType name="tSyntacticSemantic">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attributeGroup ref="tns:basicRequirementAttributes"/>
      <xs:attribute name="valueIs" type="tns:tSyntacticSemanticOperators" use="optional" default="equal"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

Figure 5.10: tSyntacticSemantic

**tParameter Type** *tParameter* type is used to define the parameter types of the operations. In USQL definition, it comprises three attributes and an attribute group: *name*, *type*, *semantics* attributes and *basicRequirementAttributes* attribute group. Here *name* and *type* attributes have the modified type *tSemantic* and *semantics* has the modified type *tSemantic*. Therefore, *name* and *type* attributes' types are turned to *tSyntacticSemantic* and *semantics* attribute is removed. After these modifications *tParameter* type definition turned out to the one given in Figure 5.11.

```

<xs:complexType name="tParameter">
  <xs:sequence>
    <xs:element name="name" type="tns:tSyntacticSemantic" minOccurs="0"/>
    <xs:element name="type" type="tns:tSyntacticSemantic" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="tns:basicRequirementAttributes"/>
</xs:complexType>

```

Figure 5.11: tParameter

**tWeight Type** In DSWS-D System, the quality of service calculation algorithm uses weights, values in range [0-100], for the quality parameters. To specify the weights in S-USQL we define a *tWeight* type. All of the quality of service elements in S-USQL are defined in this type. The type definition is given in Figure 5.12.

```

<xs:simpleType name="tWeight">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="100" />
  </xs:restriction>
</xs:simpleType>

```

Figure 5.12: tWeight

**ServiceName Element** *ServiceName* element is used to specify the search criteria related to the desired service name. In USQL definition, it is defined in the modified type *tSyntactic*. So we define it in type *tSyntacticSemantic* and match it both syntactically and semantically. After this modification, the element is defined as given in Figure 5.13.

```
<xs:element name="ServiceName" type="tns:tSyntacticSemantic" />
```

Figure 5.13: ServiceName

**ServiceDescription Element** *ServiceDescription* element is used to specify the search criteria related to the desired service description. In USQL definition, it is defined in the modified type *tSyntactic*. We changed its type to *tSyntacticSemantic*. After this modification, the element is defined as given in Figure 5.14.

```
<xs:element name="ServiceDescription" type="tSyntacticSemantic" />
```

Figure 5.14: ServiceDescription

**ServiceDomain Element** *ServiceDomain* element is used to allow the users to specify the domain of the requested services. In USQL definition, it is defined in the modified type *tSemantic*. We changed its type to *tSyntacticSemantic*. After this modification, the element definition turned out to the one given in Figure 5.15.

```
<xs:element name="ServiceDomain" type="tSyntacticSemantic" />
```

Figure 5.15: ServiceDomain

**Price Element** *Price* element is used to enable the users to specify the desired price of the requested service. In USQL definition, it is defined as a complex type, but we prefer to define it as a simple type element. According to the USQL definition of *Price* element, it comprises removed *tCurrency* and *tPriceContext* typed attributes. But as we said before, we do not take into consideration these information. Therefore, we narrowed its definition as given in Figure 5.16.



```
<xs:element name="Price" type="xs:float" />
```

Figure 5.16: Price

**Availability Element** *Availability* element is used to enable the users to specify the availability status of the requested service. According to the USQL definition of *Availability* element, it comprises an attribute typed *tQoS*. Instead of the *tQoS* type we defined a new type namely *tWeight*. Therefore, the *Availability* element definition is turned out to the one given in Figure 5.17.

```
<xs:element name="Availability" type="tns:tWeight" />
```

Figure 5.17: Availability

**Reliability Element** *Reliability* element is used to allow the users to specify the reliability status of the requested service. According to the USQL definition of *Reliability* element, it comprises an attribute typed *tQoS*. Again we changed the type to *tWeight*. Therefore, the element is defined as given in Figure 5.18.

```
<xs:element name="Reliability" type="tns:tWeight" />
```

Figure 5.18: Reliability

**ProcessingTime Element** *ProcessingTime* element is used to define the desired processing time of a requested service. In USQL definition, it consists of two attributes. One of them is typed *tQoS* and other one is typed *tProcessingTimeUnit*. None of these type definitions are included in S-USQL definition. So, *ProcessingTime* element is defined in type *tWeight*. After the modifications, the element is defined as given in Figure 5.19.

```
<xs:element name="ProcessingTime" type="tns:tWeight" />
```

Figure 5.19: ProcessingTime

**Throughput Element** In DSWS-D System, we handle five different quality of service parameters: availability, reliability, response time, price and throughput. The former four was already defined in USQL. On the other hand, it does not have an element that corresponds to throughput. Therefore, to meet our system's needs we define the *Throughput* element in S-USQL language as given in Figure 5.20.

```
<xs:element name="Throughput" type="tns:tWeight" />
```

Figure 5.20: Throughput

**QoS Element** *QoS* element is defined as a container for other quality of service elements in USQL. It composes of the *Price*, *Availability*, *Reliability*, *ProcessingTime* and *Security* elements. As mentioned before, security protocols are not handled in DSWS-D System, therefore we took out the *Security* element from the definition. In addition to this, we added a new quality of service element to here *Throughput* and a new attribute indicating the quality score of the service *qosScore*, which will be explained in the following sections. After these changes, the *QoS* element definition is formed as given in Figure 5.21.

```
<xs:element name="QoS">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Price" minOccurs="0" />
      <xs:element ref="tns:Availability" minOccurs="0" />
      <xs:element ref="tns:Reliability" minOccurs="0" />
      <xs:element ref="tns:ProcessingTime" minOccurs="0" />
      <xs:element ref="tns:Throughput" minOccurs="0" />
      <xs:element name="qosScore" type="xs:double"/>
      <xs:any namespace="##other" minOccurs="0" processContents="skip" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 5.21: QoS

**Operation Elements** There are two different *Operation* element definitions in USQL, one for requested services' operations and one for response services' operations. These two same-named element is defined in separate XSD files. But we defined the S-USQL in a single XSD file. Therefore, to avoid the conflicts between the names, we renamed these *Operation* elements as *RequestOperation* and *ResponseOperation*. The request *Operation* element in USQL is designed to enable the users to specify the search criteria based on the operations that the

requested services offer. It contains the *Name*, *Capability*, *Inputs*, *Outputs* elements and *minDegreeOfMatch*, *priorityLevel* attributes according to the USQL definition. *Name* is defined in the modified type *tSyntactic*, so we use *tSyntacticSemantic* type instead. And *Capability* is defined in modified type *tSemantic*, it also changed to *tSyntacticSemantic*. *Inputs* and *Outputs* elements are contained in S-USQL without any modification. Finally, the *priorityLevel* attribute is taken out as it has been removed from S-USQL language. After all these, our *RequestOperation* element is formed as given in Figure 5.22.

```

<xs:element name="RequestOperation">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element name="Name" type="tSyntacticSemantic" minOccurs="0" />
      <xs:element name="Capability" type="tSyntacticSemantic" minOccurs="0" />
      <xs:element name="Inputs" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="input" type="tns:tParameter" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attributeGroup ref="tns:basicRequirementAttributes" />
        </xs:complexType>
      </xs:element>
      <xs:element name="Outputs" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="output" type="tns:tParameter" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attributeGroup ref="tns:basicRequirementAttributes" />
        </xs:complexType>
      </xs:element>
      <xs:element ref="tns:QoS" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="minDegreeOfMatch" type="tns:tPercentage"
      use="optional" default="1.0" />
  </xs:complexType>
</xs:element>

```

Figure 5.22: RequestOperation

The response *Operation* element in USQL is designed to show the operations that complies with the user's search criteria. It contains the removed element group *AdditionalOperationProperties*. To indicate the name of the operation an attribute *name* with type *xs:string* is added and to indicate the degree of the match another attribute named *degreeOfMatch* is added. After these modifications, the structure of our *ResponseOperation* element is turned out to the one given in Figure 5.23.

**Service Elements** Like the aforementioned *Operation* elements, there are two different *Service* element definitions in USQL, one for requested services and one for resulting services. Again we needed a refactor in these elements' names, and changed them as *RequestService* and *ResponseService*. The request *Service* element is designed as a container for the search

```

<xs:element name="ResponseOperation">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="degreeOfMatch" type="tns:tPercentage" use="required" />
  </xs:complexType>
</xs:element>

```

Figure 5.23: ResponseOperation

criteria elements. In USQL definition, it contains the *ServiceName*, *ServiceDescription*, *ServiceProvider*, *ServiceTaxonomy*, *ServiceDomain*, *QoS*, *Operation* elements and *minDegreeOfMatch* attribute. However, we excluded the *ServiceProvider* and *ServiceTaxonomy* elements while constructing the S-USQL. Therefore, the our *RequestService* is defined as given in Figure 5.24.

```

<xs:element name="RequestService">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element ref="tns:ServiceName" minOccurs="0" />
      <xs:element ref="tns:ServiceDescription" minOccurs="0" />
      <xs:element ref="tns:ServiceDomain" minOccurs="0" />
      <xs:element ref="tns:QoS" minOccurs="0" />
      <xs:element ref="tns:RequestOperation" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="minDegreeOfMatch" type="tns:tPercentage" use="optional" default="1" />
  </xs:complexType>
</xs:element>

```

Figure 5.24: RequestService

```

<xs:element name="ResponseService">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="tns:ServiceName" />
      <xs:element name="descriptionDocUrl" type="xs:anyURI"/>
      <xs:group ref="tns:AdditionalServiceProperties" />
      <xs:element name="interface" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="tns:ResponseOperation" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="name" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="degreeOfMatch" type="tns:tPercentage" use="required" />
  </xs:complexType>
</xs:element>

```

Figure 5.25: ResponseService

And the response *Service* element is used to indicate the services found after a query process. In USQL definition, it composes of *name*, *descriptionDocUrl*, *resourceInstance*, *Addition-*

*alServiceProperties*, *interface* elements and *ServiceAttributes*, *type*, *networkType* attributes. Here *interface* element is a list of the *Operation* elements. In DSWSD-S System, we do not keep the resource information related to services. And the *networkType* element is not meaningful in our system, because we do not interested in the service types (like grid or peer-to-peer) other than web services. After the modifications, our *ResponseService* element definition is done as given in Figure 5.25.

**Services Element** In USQL definition *Services* element is a container element that keeps the list of response type *Service* elements. In S-USQL, we renamed this element as *ResponseServices* to make its name compatible with the *ResponseService* elements it keeps. *ResponseServices* element definition is given in Figure 5.26.

```
<xs:element name="ResponseServices">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:ResponseService" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 5.26: ResponseServices

```
<xs:element name="SUSQLRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Where">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="tns:RequestService" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="numberOfResults" type="xs:int"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 5.27: SUSQLRequest

**USQLRequest Element** As mentioned in Section 5.2.1.1, *USQLRequest* element is composed of projection and criteria parts in USQL definition and we do not include the projection part in our *SUSQLRequest* element definition. In USQL specification, *ViewAdditionalProperties* element corresponds to the projection part. *From*, *Where* and *OrderBy* elements correspond to criteria part. We include the criteria part but with some changes. We do not enable the users to make their search on specific registries. In addition, we have removed the

aforementioned ordering types from SUSQL. In USQL definition, there is not a structure that enabling the user to limit the number of the results, so we add an attribute *numberOfResultsAfter* to element definition. After these modifications, our *SUSQLRequest* element is formed as given in Figure 5.27.

**USQLResponse Element** *USQLResponse* contains the services found, which meet the requirements specified in a *SUSQLRequest*, after a query process. As mentioned in Section 5.2.1.2 it contains two parts: resulting services list (represented with *Services* element) or error part (represented with *Error* element). These parts are alternative for each other. At the same time a *USQLResponse* can not contain both of them. We add this element definition to our S-USQL definition with a change in its name as *SUSQLResponse*. The definition of *SUSQLResponse* is given in Figure 5.28.

```
<xs:element name="SUSQLResponse">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="tns:ResponseServices" />
      <xs:element name="Error">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="code" type="xs:string" />
            <xs:element name="desc" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Figure 5.28: SUSQLResponse

```
<xs:element name="SUSQL">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="tns:SUSQLRequest" />
      <xs:element ref="tns:SUSQLResponse" />
    </xs:choice>
    <xs:attribute name="version" type="xs:float" use="required" fixed="1.0" />
  </xs:complexType>
</xs:element>
```

Figure 5.29: SUSQL

**USQL Element** *USQL* element is the root element in USQL specification. It contains the *USQLRequest*, *USQLResponse* elements and an attribute showing the version of the USQL. Here, the root element can contain only one of the elements listed at the same time. Meanly, the request and the response elements are alternative to each other in use. We also define a

root element containing our own request and response elements. Our root element is named *SUSQL* and the structure of it is given in Figure 5.29.

### 5.3 Simplified USQL (S-USQL) Parser

In DSWS-D System, the crawler nodes does not know the S-USQL language, therefore they can not interpret the S-USQL queries. To achieve the interoperability between the S-USQL requestor (graphical interface) and the responder (crawler nodes' local databases) we design a middleware parser. This parser module, parses the *SUSQLRequest* queries coming from requestors and transforms the information parsed to the input parameters of the web service operations that are handling the crawler nodes' local database issues. The parser also transforms the web service responses to the structures needed to produce a *SUSQLResponse*.

As in the graphical user interface module, the parser module also interacts with all the existing crawler nodes in DSWS-D System. It uses the *ServiceDomain* element in the *SUSQLRequest* in order to decide the crawler node to be searched.

The first thing the parser does when it receives a *SUSQL* containing a *SUSQLRequest* is extracting the *ServiceName* and the *ServiceDomain* elements. *ServiceName* element corresponds to the keywords that are searched for. The *ServiceDomain* corresponds to the domain that the user specifically intends to search for. After that, it extracts the operator information that indicates whether the matching will be done syntactically or semantically.

If the operator type is *Equal* or *Contain*, the parser interprets this as *do syntactic matching*. After deciding syntactic matching, there are three options for the parser to continue with according to the existence of the keyword and domain information.

If the keyword is not specified, then the parser does not make any search on the crawler nodes. The parser creates a *SUSQL* containing a *SUSQLResponse* and puts an *Error* element into it. In the *desc* attribute of the *Error* element, it explains the reason why the search can not be done.

If the keyword is specified but the domain information is empty, then the parser matches this keyword to all the existing crawler nodes' domain information and decides the most suitable crawler node by ranking them on their matching degree with the keyword. After that, if the

selected operator type is *Equal*, the parser searches for an ontology term that is same with the keyword and retrieves the web services from crawler node's local database related to this ontology term. However, if the operator type is specified as *Contain*, then the parser looks for the ontology terms that contains the keyword as a substring. After that, it retrieves the web services from crawler node's local database related to the ontology terms containing the keyword.

Finally, if the keyword and the domain information are both specified, then the parser executes the same process mentioned in the previous paragraph from the point where the domain related to the keyword is decided.

If the operator type is *Exact*, *Abstraction*, *Extension* or *Sibling*, the parser interprets this as *do semantic matching*. After deciding semantic matching, there are four options for the parser to continue with according to the existence of the keyword and domain information.

If both of them are empty, then the parser again creates a *SUSQL* containing a *SUSQLResponse* and puts an *Error* element into it. In the *desc* attribute of the *Error* element, it explains the reason why the search can not be done.

In the case where the keyword is empty but the domain information is not, the parser retrieves all the web services kept in corresponding crawler node.

If the keyword is set but the domain information is empty, then the parser matches this keyword to all the existing crawler nodes' domain information and decides the most suitable crawler node by ranking them on their matching degree with the keyword. After that, if the operator type is *Exact*, the parser searches for the ontology terms whose matching degree with the keyword is 1.00. The parser retrieves the web services from the crawler's local database related to this ontology terms. However, if the operator type is *Abstraction*, *Extension* or *Sibling*, then the parser calculates the matching degree between the keyword and all the ontology terms related to the selected domain by using the operator type. Then it filters these ontology terms according to their matching degree with the keyword. If the matching degree is greater than 0.5 threshold, then the parser retrieves the web services related to these ontology terms.

Finally, if both the parameters are set, then the parser executes the same process mentioned in the previous paragraph from the point where the domain related to the keyword is decided.



After retrieving the candidate web services from related crawler node's local database, the parser applies the quality of service criteria on them, if they are specified in the query. After this quality of service filtering, the parser constitutes the resulting web service list. For every web service found, it creates a corresponding *ResponseService* element to be contained in the *SUSQLResponse* that will be returned to the requestor of the web services.

## **CHAPTER 6**

# **QUERY EXECUTION PROCESS AND INTERACTION WITH OTHER MODULES OF DSWSD-S**

### **6.1 Introduction**

In order to respond to a user's web service discovery request in DSWSD-S System, there are different modules that should interact with each other. Some of the modules are responsible for getting the user requests, some of them are parsing these queries and some of them are finding the desired web services. In the following sections, the modules that take place in a web service query process and the interactions between these modules will be explained.

### **6.2 Modules Responsible For Query Execution**

There are four modules that take part in a web service query process: the graphical user interface module, the S-USQL parser module, the similarity degree calculator web service and the web service that handles a crawler node's database operations.

#### **6.2.1 Graphical User Interface Module**

The graphical user interface module is the one where the web service query process is triggered. It is explained in Chapter 4 in detail.

### **6.2.2 S-USQL Parser Module**

The S-USQL parser module is the one where the mapping between the S-USQL elements and the database operations handler web service method parameters are done. It is explained in Section 5.3.

### **6.2.3 Similarity Degree Calculator Web Service**

The similarity degree calculator web service basically calculates the similarity degree between two different texts. Actually, it is a wrapper for the similarity degree calculation library developed within DSWS-D System on top of a third party library, WordNet's similarity calculation library. The library was developed in a programming environment that is different and incompatible with the one we use. Therefore, an interoperability problem is raised. To overcome this problem we design a web service that wraps the similarity calculation operations. This web service runs on the same machine with the graphical user interface and the S-USQL parser modules. The list of the operations offered by this web service is given in Appendix B

### **6.2.4 Database Operations Handler Web Service**

The database operations handler web service is the one that allows to reach the crawler node's local databases and retrieve web services from there. There are different modules that need to access to the crawler node's databases in DSWS-D System: graphical user interface module, quality of service calculator module and the web service verifier module. To supply the needs of these different modules on the databases, this web service is designed. It offers operations that enables its requestors to retrieve the web services, their descriptions, domain ontologies corresponds to the crawler nodes' domains...etc. It also offers operations to update the information kept in the databases. The list of the operations offered by this web service is given in Appendix C. For every crawler node existing in DSWS-D System, there is a corresponding database operation handler web service instance. Basically, on every crawler node machine, one instance of this web service runs.

### 6.3 Query Execution Process

As aforementioned in Section 6.2.1 the graphical user interface is the trigger point for a web service query. The user enters the criteria that he/she wants the desired web service should comply with and then starts the process via clicking on the *Search* button on the graphical user interface as shown in Figure fig:guiButtonWidgets. After the user entered the necessary information and clicked on the button, the graphical user interface module constructs an S-USQL request and sends this request to the S-USQL parser module.

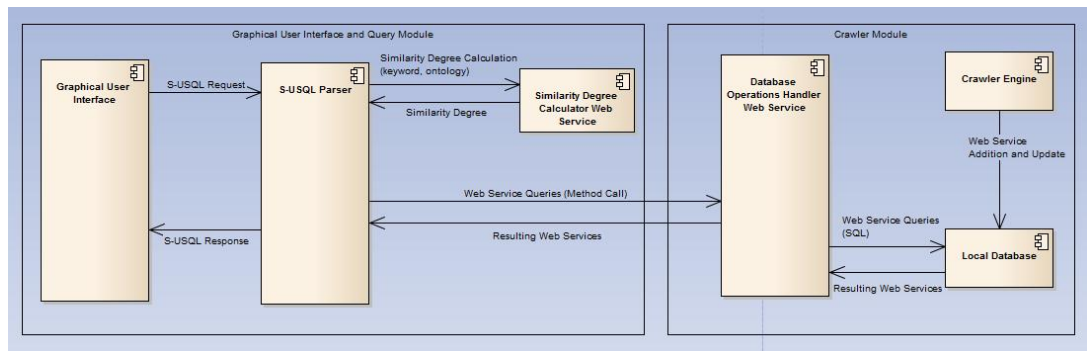


Figure 6.1: Interaction Between Modules

The S-USQL parser module, parses the S-USQL request. As explained in Section 5.3, it extracts the keyword and domain ontology information from the request. After extraction, it reads the selected domain ontology description from the crawler node via calling the database operation handler web service. Then it needs to calculate the matching degree between the keyword and the ontology terms contained in the selected domain ontology. To achieve this, the parser module calls the similarity degree calculator web service. It passes the keyword and the ontology term to the web service as parameters. The similarity degree calculator web service calculates the matching degree and returns the value to the parser module. For the ontology terms whose matching degree with the keyword is greater than 0.5 threshold, the parser module calls the database operation handler web service again. This time, it requests the web services that are related with the ontology term. After this step, the parser module extracts the quality of service parameters from the request, if they have been specified. And sends the resulting web service list and the quality of service parameters to the database operation handler web service for the last time. The returned web service list is the final result set that will be presented to the user in the graphical user interface. To send the resulting web

services to graphical user interface, the parser module creates an S-USQL response and lists all the web services found in it as response services.

The graphical user interface uses the information came in the S-USQL response, and shows them on the result table shown in Figure 4.9 to the user.

The query process mentioned is given in Figure 6.1.

## CHAPTER 7

### CASE STUDIES AND ANALYSIS

In this chapter, we show some query samples and respective results to explain the usage of S-USQL language. In addition to this, we made a query execution time performance analysis with the given sample queries.

#### 7.1 Query with Single Keyword

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLRequest>
    <Where numberOfResults="0">
      <RequestService minDegreeOfMatch="0.25">
        <ServiceName valueIs="all">wheel</ServiceName>
        <ServiceDomain></ServiceDomain>
      </RequestService>
    </Where>
  </SUSQLRequest>
</SUSQL>
```

Figure 7.1: Sample Query-1

In the query given in Figure 7.1, the web services whose name contains the word “*wheel*” are requested. Depending on the absence of domain ontology specification, the word will be matched to all the existing crawler nodes’s domain ontologies. After this matching the crawler node whose ontology matches with the highest degree will be selected to be searched.

In this example there are two crawler nodes in the DSWSD-S System, one is interested in the “*Car*” domain and the other one is interested in “*Aviation*” domain. After matching with both these domains, it is found out that the word “*wheel*” is more relevant to “*Car*” domain. Therefore the web service query is directed to the corresponding crawler node. The crawler node finds 4297 web services and it returns all of them because no limit on the result number

is specified in the query. Some part of the response document containing the resulting web services is given in Figure 7.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLResponse>
    <ResponseServices>
      <ResponseService degreeOfMatch="0.895">
        <ServiceName>Deredden_Auto</ServiceName>
        <descriptionDocUrl>http://roservices.net/spectrum/spectrumws_v3.1.0/util.asmx</descriptionDocUrl>
        <ServiceDescription>Spectrum Deredden_Auto(Spectrum spec)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.895">
        <ServiceName>UpdateCarResults</ServiceName>
        <descriptionDocUrl>http://www.msn.momondo.com/Momondo.asmx</descriptionDocUrl>
        <ServiceDescription>System.String UpdateCarResults()</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.895">
        <ServiceName>StartCarSearch</ServiceName>
        <descriptionDocUrl>http://www.msn.momondo.com/Momondo.asmx</descriptionDocUrl>
        <ServiceDescription>Void StartCarSearch()</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.895">
        <ServiceName>GetSubFamiliasCuidadosAuto</ServiceName>
        <descriptionDocUrl>http://www.galpennergia.com/PT/_layouts/GalpPublicWebService.asmx</descriptionDocUrl>
        <ServiceDescription>System.String GetSubFamiliasCuidadosAuto(String currentFamilia)</ServiceDescription>
      </ResponseService>
      ....
      <ResponseService degreeOfMatch="0.895">
        <ServiceName>PagesList</ServiceName>
        <descriptionDocUrl>http://www.thrifty.com.au/SimpleWebService.asmx</descriptionDocUrl>
        <ServiceDescription>System.String PagesList(String sCarCode)</ServiceDescription>
      </ResponseService>
    </ResponseServices>
  </SUSQLResponse>
</SUSQL>
```

Figure 7.2: Response for Sample Query-1

The resulting web services are presented to the user in the result table. The screenshot respective to this query and its response is given in Figure 7.3.

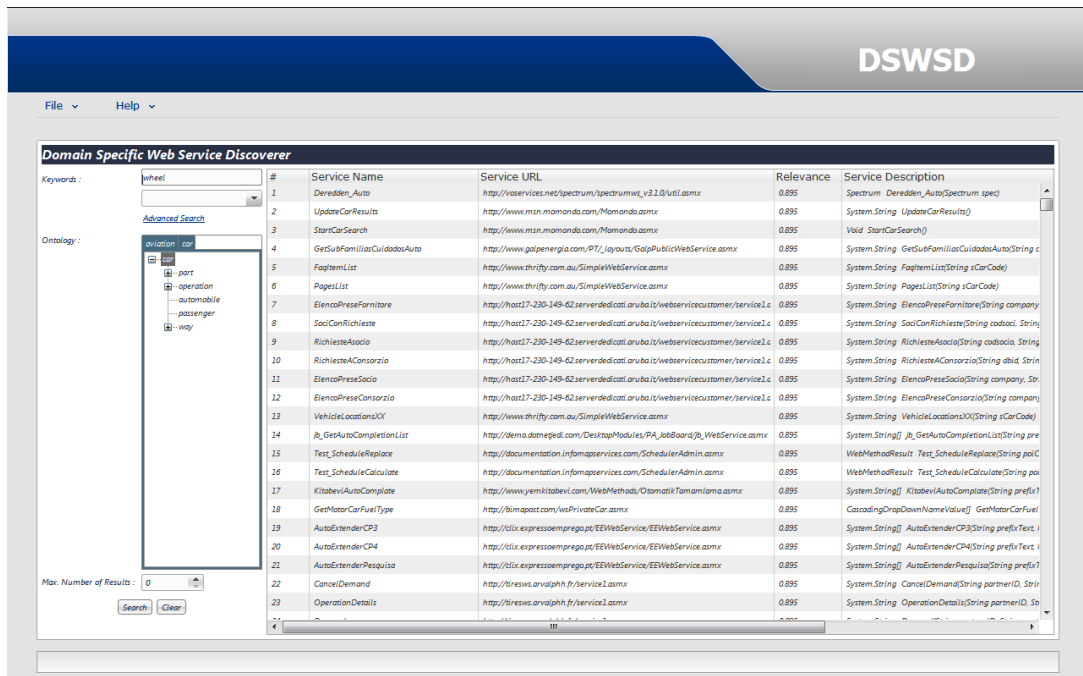


Figure 7.3: Screen for Sample Query-1

## 7.2 Query with Single Keyword and Domain Specification

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLRequest>
    <Where numberOfResults="0">
      <RequestService minDegreeOfMatch="0.25">
        <ServiceName valueIs="all">flight</ServiceName>
        <ServiceDomain>aviation</ServiceDomain>
      </RequestService>
    </Where>
  </SUSQLRequest>
</SUSQL>
```

Figure 7.4: Sample Query-2

With the query given in Figure 7.4, the user requests the web services that match with the word “flight” and are related to the domain “Aviation”. Due to the absence of the matching type, the keyword is matched with all the ontology terms in the “Aviation” ontology both syntactically and semantically. The terms whose matching degree is greater than the threshold is extracted. After that, the web services related to these ontology terms are retrieved. 1878 resulting web services are found in this query. Again due to the absence of limitation in total result number in the query, all the web services are presented to the user. Some part of the response document where the resulting web services are contained is given in Figure 7.5.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLResponse>
    <ResponseServices>
      <ResponseService degreeOfMatch="0.99">
        <ServiceName>CheckAirTempAvai4Station</ServiceName>
        <descriptionDocUrl>http://gsserver.ceegs.ohio-state.edu/website/gaugews/WSService.asmx</descriptionDocUrl>
        <ServiceDescription>Boolean CheckAirTempAvai4Station(String Date, Inventory StInvInfo)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.99">
        <ServiceName>StrategyHATCampbell</ServiceName>
        <descriptionDocUrl>http://agsys.cra-cin.it/webservices/airtemperature/airtemperature.asmx</descriptionDocUrl>
        <ServiceDescription>Double[] StrategyHATCampbell(Double CapbellTimingVariation, Double AirTemperatureMax, Double AirTemperatureMin)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.98">
        <ServiceName>FlightPlans</ServiceName>
        <descriptionDocUrl>https://flightwise.com/ws/PlaneXMLbeta4.asmx</descriptionDocUrl>
        <ServiceDescription>FlightPlan[] FlightPlans(String query)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.98">
        <ServiceName>FlightStatus</ServiceName>
        <descriptionDocUrl>https://flightwise.com/ws/PlaneXMLbeta4.asmx</descriptionDocUrl>
        <ServiceDescription>FIDynamic FlightStatus(String ident)</ServiceDescription>
      </ResponseService>
      ....
      <ResponseService degreeOfMatch="0.98">
        <ServiceName>FlightPath</ServiceName>
        <descriptionDocUrl>https://flightwise.com/ws/PlaneXMLbeta4.asmx</descriptionDocUrl>
        <ServiceDescription>adsPosition[] FlightPath(String ident)</ServiceDescription>
      </ResponseService>
    </ResponseServices>
  </SUSQLResponse>
</SUSQL>
```

Figure 7.5: Response for Sample Query-2



The screenshot respective to this query and its response is given in Figure 7.6.

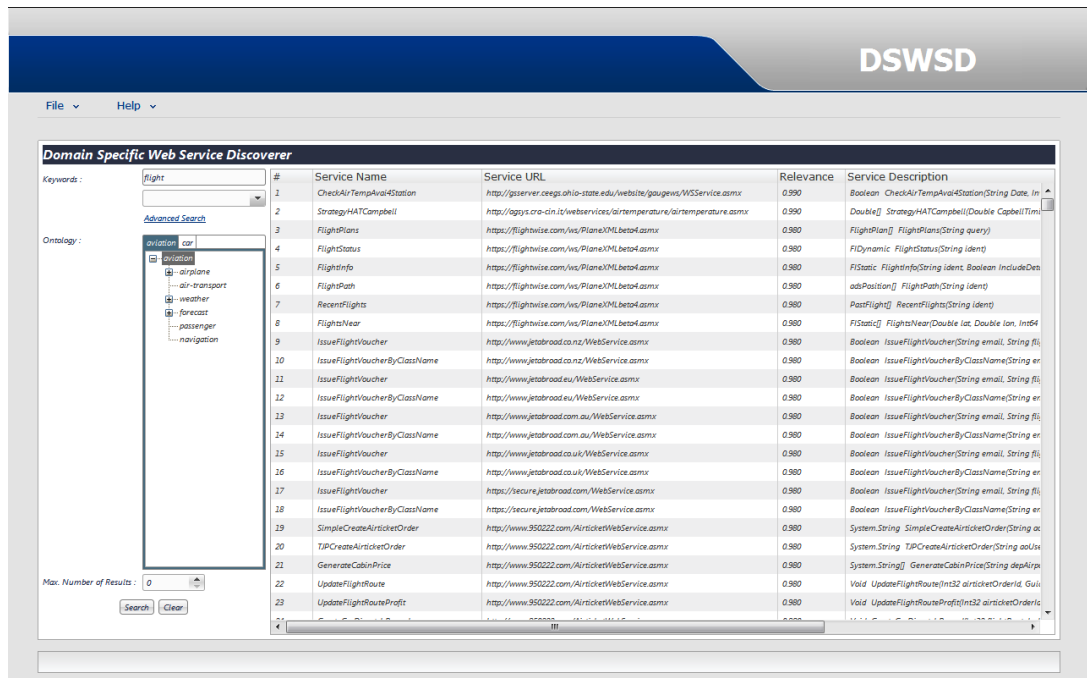


Figure 7.6: Screen for Sample Query-2

### 7.3 Query with Custom Quality of Service Criteria

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLRequest>
    <Where numberOfResults="0">
      <RequestService minDegreeOfMatch="0.25">
        <ServiceName valueIs="all">trip</ServiceName>
        <ServiceDomain>car</ServiceDomain>
        <QoS>
          <Availability>0</Availability>
          <Reliability>80</Reliability>
          <ProcessingTime>20</ProcessingTime>
          <Throughput>0</Throughput>
          <qosScore>0.0</qosScore>
        </QoS>
      </RequestService>
    </Where>
  </SUSQLRequest>
</SUSQL>
```

Figure 7.7: Sample Query-3

With the query given in Figure 7.7, the user requests the web services that match with the word “trip” and are related to the domain “Car”. Due to the absence of the matching type, the keyword is matched with all the ontology terms in the “Car” ontology both syntactically and

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLResponse>
    <ResponseServices>
      <ResponseService degreeOfMatch="0.7">
        <ServiceName>GetCarModels</ServiceName>
        <descriptionDocUrl>http://www.dotzmycar.com/(S(h45zbb55osp3j1454kpp0145))/TyreSizeService.aspx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue[]
          GetCarModels(String knownCategoryValues, String category)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.7">
        <ServiceName>GetCarBrands</ServiceName>
        <descriptionDocUrl>http://www.dotzmycar.com/TyreSizeService.aspx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue[]
          GetCarBrands(String knownCategoryValues, String category)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.7">
        <ServiceName>GetCarYears</ServiceName>
        <descriptionDocUrl>http://www.dotzmycar.com/TyreSizeService.aspx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue[]
          GetCarYears(String knownCategoryValues, String category)</ServiceDescription>
      </ResponseService>
      ...
      <ResponseService degreeOfMatch="0.5735622">
        <ServiceName>GetDiameterClearance</ServiceName>
        <descriptionDocUrl>http://www.4wheelparts.com/aux_incl/ajax/wheelService.aspx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue[]
          GetDiameterClearance(String knownCategoryValues, String category)</ServiceDescription>
      </ResponseService>
    </ResponseServices>
  </SUSQLResponse>
</SUSQL>

```

Figure 7.8: Response for Sample Query-3

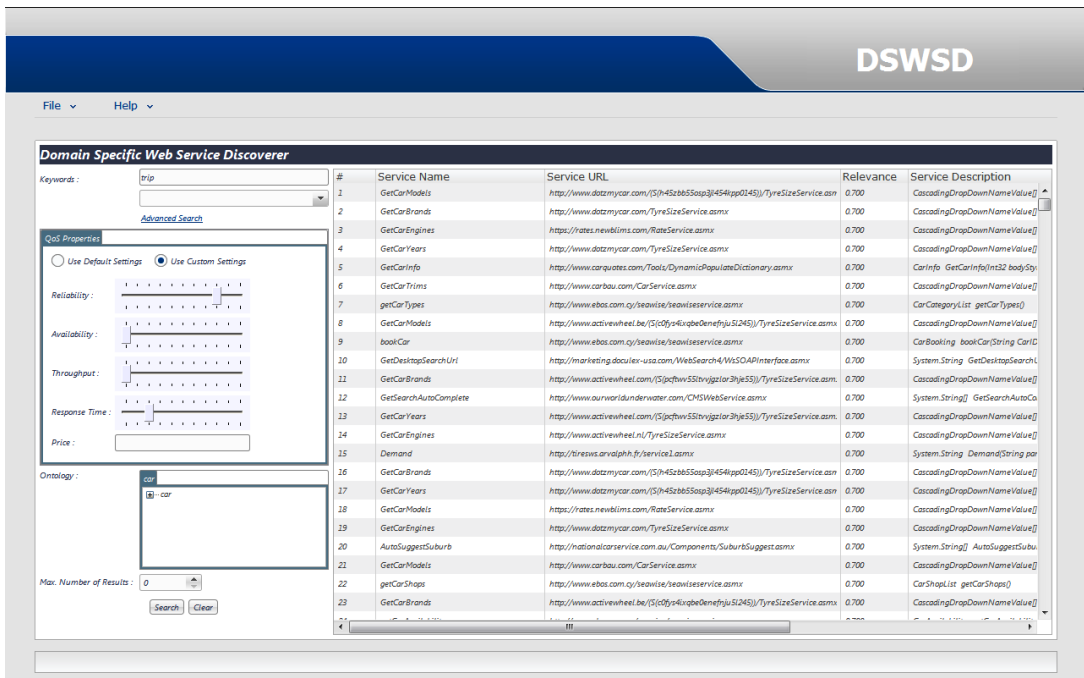


Figure 7.9: Screen for Sample Query-3

semantically. In addition to these, he/she specifies some quality of service parameters. The user wants the reliability parameter’s weight to be 80% and the response time parameter’s weight to be 20%. After matching the “*trip*” word with the “*Car*” ontology, 835 candidate web services are found. Then a quality of service filtering is applied to these services, and found out that 673 web services complies with the quality of service criteria specified in the query. Due to the absence of limitation in total result number in the query, all the web services are presented to the user. Some part of the response document where the resulting web services are contained is given in Figure 7.8.

The screenshot respective to this query and its response is given in Figure 7.9.

## 7.4 Query with Default Quality of Service Criteria

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLRequest>
    <Where numberOfResults="0">
      <RequestService minDegreeOfMatch="0.25">
        <ServiceName valueIs="all">trip</ServiceName>
        <ServiceDomain>car</ServiceDomain>
        <QoS>
          <Availability>50</Availability>
          <Reliability>50</Reliability>
          <ProcessingTime>50</ProcessingTime>
          <Throughput>50</Throughput>
          <qosScore>0.0</qosScore>
        </QoS>
      </RequestService>
    </Where>
  </SUSQLRequest>
</SUSQL>
```

Figure 7.10: Sample Query-4

With the query given in Figure 7.10, the user requests the web services that match with the word “*trip*” and are related to the domain “*Car*” as in Sample Query-3. Due to the absence of the matching type, the keyword is matched with all the ontology terms in the “*Car*” ontology both syntactically and semantically. In this query, the user specifies quality of service parameters and chooses to use their default weights. As known from the previous query, after matching the “*trip*” word with the “*Car*” ontology, 835 candidate web services are found. Then a quality of service filtering is applied to these services, and found out that 723 web services complies with the quality of service criteria specified in the query. Due to the absence of limitation in total result number in the query, all the web services are presented to the user. Some part of the response document where the resulting web services are contained is given

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <Response>
    <ResponseServices>
      <ResponseService degreeOfMatch="0.7">
        <ServiceName>Validate_SporeCarRegNo</ServiceName>
        <descriptionDocUrl>http://www.softwaremaker.net/webservices/
          swm/validator/validator.asmx</descriptionDocUrl>
        <ServiceDescription>Int16 Validate_SporeCarRegNo
          (String ps_InputNo)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.7">
        <ServiceName>GetCarModels</ServiceName>
        <descriptionDocUrl>https://rates.newblims.com/
          RateService.asmx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue []
          GetCarModels (String knownCategoryValues, String category)
        </ServiceDescription>
      </ResponseService>
      ....
      <ResponseService degreeOfMatch="0.5884875">
        <ServiceName>GetLinks</ServiceName>
        <descriptionDocUrl>http://www.haitirewired.org/WebServVolunteersRewired/
          Service.asmx</descriptionDocUrl>
        <ServiceDescription>System.Xml.XmlElement GetLinks ()</ServiceDescription>
      </ResponseService>
    </ResponseServices>
  </Response>
</SUSQL>

```

Figure 7.11: Response for Sample Query-4

in Figure 7.11.

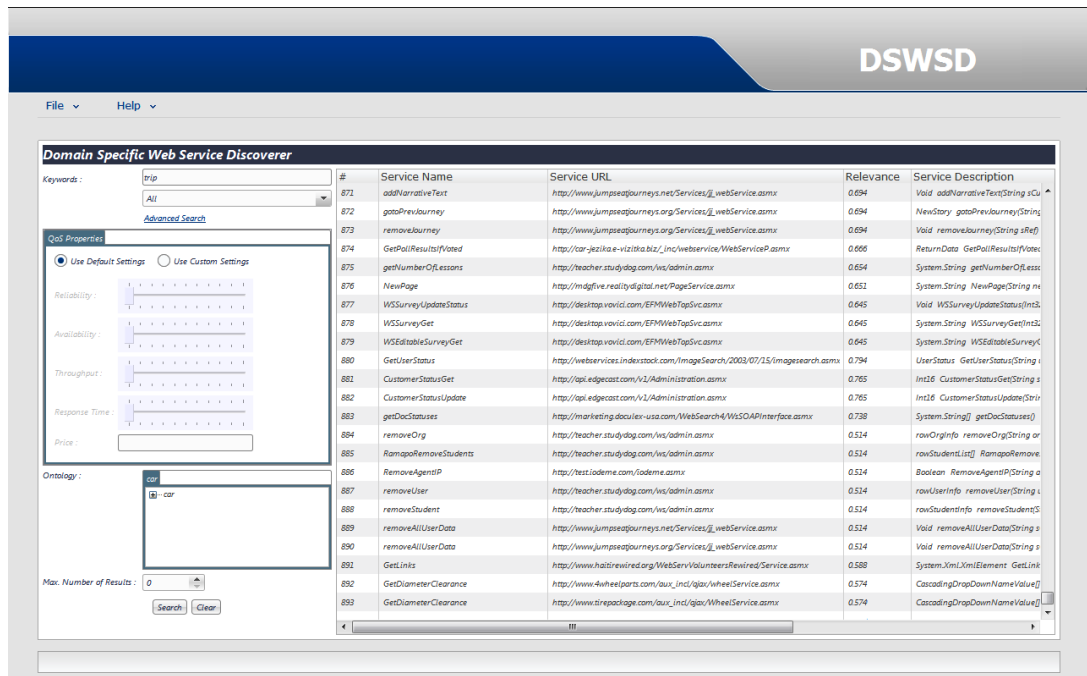


Figure 7.12: Screen for Sample Query-4

The screenshot respective to this query and its response is given in Figure 7.12.

## 7.5 Query with Multiple Keywords and Custom Quality of Service Criteria

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLRequest>
    <Where numberOfResults="0">
      <RequestService minDegreeOfMatch="0.25">
        <ServiceName valueIs="all">hub cap</ServiceName>
        <ServiceDomain>car</ServiceDomain>
        <QoS>
          <Availability>0</Availability>
          <Reliability>50</Reliability>
          <ProcessingTime>0</ProcessingTime>
          <Throughput>0</Throughput>
          <qosScore>0.0</qosScore>
        </QoS>
      </RequestService>
    </Where>
  </SUSQLRequest>
</SUSQL>
```

Figure 7.13: Sample Query-5

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SUSQL xmlns="urn:dswsd:SUSQL" version="1.0">
  <SUSQLResponse>
    <ResponseServices>
      <ResponseService degreeOfMatch="0.72">
        <ServiceName>AutoSuggestSuburb</ServiceName>
        <descriptionDocUrl>http://www.nationalcarservice.com.au/Components
          /SuburbSuggest.aspx</descriptionDocUrl>
        <ServiceDescription>System.String[]
          AutoSuggestSuburb(String prefixText, Int32 count, String contextKey)</ServiceDescription>
      </ResponseService>
      <ResponseService degreeOfMatch="0.72">
        <ServiceName>GetCarBrands</ServiceName>
        <descriptionDocUrl>http://www.dotzmycar.com/(S(h45zbb55osp3j1454kpp0145))
          /TyreSizeService.aspx</descriptionDocUrl>
        <ServiceDescription>CascadingDropDownNameValue[]
          GetCarBrands(String knownCategoryValues, String category)</ServiceDescription>
      </ResponseService>
      ....
      <ResponseService degreeOfMatch="0.6097548">
        <ServiceName>WSAnalysisGetChart</ServiceName>
        <descriptionDocUrl>http://desktop.vovici.com
          /EFMWebTopSvc.aspx</descriptionDocUrl>
        <ServiceDescription>WSChartImageInfo
          WSAnalysisGetChart(Int32 editableSurveyId, String wsbId,
            Int32 chartWidth, Int32 chartHeight)</ServiceDescription>
      </ResponseService>
    </ResponseServices>
  </SUSQLResponse>
</SUSQL>
```

Figure 7.14: Response for Sample Query-5

In the query given in Figure 7.13, the web services whose name contains the “hub cap” words and related to the “Car” domain are requested. Here, the reliability weight of the desired services are intended to be 50%. After matching the “hub cap” word with the “Car” ontology,

1128 candidate web services are found. Then a quality of service filtering is applied to these services, and found out that 452 web services comply with the quality of service criteria specified in the query. But not all of them is shown to the user. Because as seen in Figure 7.13, the total number of results are limited to 40. Therefore, the 452 resulting services are sorted in a descending order according to their matching degrees and top 40 of them are represented to the user. Some part of the response document containing the resulting 40 web services is given in Figure 7.14.

The screenshot respective to this query and its response is given in Figure 7.15.

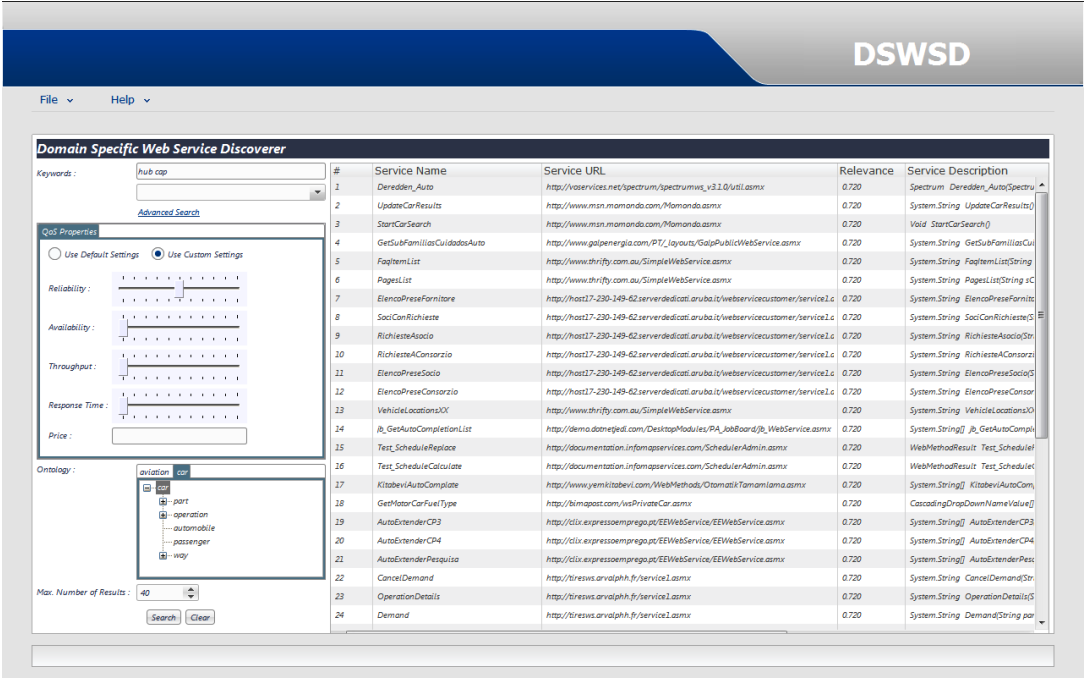


Figure 7.15: Screen for Sample Query-5

### 7.6 Query Execution Time Performance Analysis

To analyze the performance of different query cases, we have taken some measurements about the query execution times. All the measurements are taken at 25th of August, 2011, at times 15.00, 16.00, 17.00, 18.00 and 19.00.

In following Tables 7.1, 7.2, 7.3, 7.4, 7.5 the measurements taken for sample queries are given.

Table 7.1: Query Execution Times for Sample Query-1

Duration in ms	Time: 15.00	Time: 16.00	Time: 17.00	Time: 18.00	Time: 19.00	Average
Formulate Query	312	300	291	311	341	311
Parse Query	2964	3456	3153	3282	4496	3470.2
Retrieve Services	30093	25055	28595	27025	31767	28507
Evaluate QoS on Services	0	0	0	0	0	0
Formulate Response	6	6	6	7	7	6.4
Retrieved WS Count	<b>4297 candidate web services</b>					
QoS Filtered WS Count	<b>4297 resulting web services</b>					

Table 7.2: Query Execution Times for Sample Query-2

Duration in ms	Time: 15.00	Time: 16.00	Time: 17.00	Time: 18.00	Time: 19.00	Average
Formulate Query	311	299	306	315	303	306.8
Parse Query	2	1	1	1	1	1.2
Retrieve Services	7526	6650	7082	8257	11326	8168.2
Evaluate QoS on Services	0	0	0	0	0	0
Formulate Response	3	2	3	2	3	2.6
Retrieved WS Count	<b>1878 candidate web services</b>					
QoS Filtered WS Count	<b>1878 resulting web services</b>					

As can be seen from the tables, the duration for parsing the sample query-1 takes much more longer than other three queries. The reason of this is that, the user does not specify a domain

Table 7.3: Query Execution Times for Sample Query-3

Duration in ms	Time: 15.00	Time: 16.00	Time: 17.00	Time: 18.00	Time: 19.00	Average
Formulate Query	303	321	308	312	308	310.4
Parse Query	2	1	1	1	1	1.2
Retrieve Services	3378	3120	2945	4094	2760	3259.4
Evaluate QoS on Services	13470	14533	12415	15522	12404	13668.8
Formulate Response	1	2	2	1	1	1.4
Retrieved WS Count	<b>835 candidate web services</b>					
QoS Filtered WS Count	<b>673 resulting web services</b>					

Table 7.4: Query Execution Times for Sample Query-4

Duration in ms	Time: 15.00	Time: 16.00	Time: 17.00	Time: 18.00	Time: 19.00	Average
Formulate Query	296	285	292	302	280	291
Parse Query	1	1	1	1	1	1
Retrieve Services	3168	3349	2967	3990	2870	3268.8
Evaluate QoS on Services	6370	7255	7110	7855	6210	6960
Formulate Response	1	2	2	1	1	1.4
Retrieved WS Count	<b>835 candidate web services</b>					
QoS Filtered WS Count	<b>723 resulting web services</b>					

information in sample query-1. Therefore, the parser tries to find the most appropriate domain for the keyword by matching it with all the existing crawler nodes' domain ontologies in



Table 7.5: Query Execution Times for Sample Query-5

Duration in ms	Time: 15.00	Time: 16.00	Time: 17.00	Time: 18.00	Time: 19.00	Average
<b>Formulate Query</b>	347	309	315	325	306	320.4
<b>Parse Query</b>	1	1	1	1	1	1
<b>Retrieve Services</b>	9243	4006	3542	3532	4048	4874.2
<b>Evaluate QoS on Services</b>	24738	21307	16590	15531	18049	19243
<b>Formulate Response</b>	1	1	2	2	1	1.4
<b>Retrieved WS Count</b>	<b>1128 candidate web services</b>					
<b>QoS Filtered WS Count</b>	<b>452 resulting web services</b>					

DSWSD-S System. As a result of this matching an increase in the parsing process time occurs.

If we look over the service retrieval times in detail, it can be seen that the retrieval time increases in direct proportion to the number of resulting web services.

In the first two sample queries, the quality of service parameters evaluation time is zero. Actually, this result is the one that is expected, because in these queries the user does not specify any quality of service criteria. However, in sample query-3, sample query-4 and sample query-5, these parameters specified and it takes longer to evaluate the quality of service parameters in sample query-5 due to the number of the resulting web services found. Again, the quality of service evaluation time increases in direct proportion to the number of the web services on whom the quality of service filtering is intended to be done. The choice of the user about the quality of service specification, custom or default weight usage, also effects the quality of service evaluation time as can be seen from the tables. In the sample query-3 and sample query-4, the user searches the same keyword on the same domain. However, he/she prefers to defines his/her own quality of service parameters in sample query-3, while he/she prefers to use the defaults in sample query-4. Whenever the user specifies his/her custom

weights, the total quality score of the candidate web services are recalculated and this causes an increase in the quality of service evaluation time.

We calculated the average times for each step for the sample queries and produced the overall performance graph given in Figure 7.16.

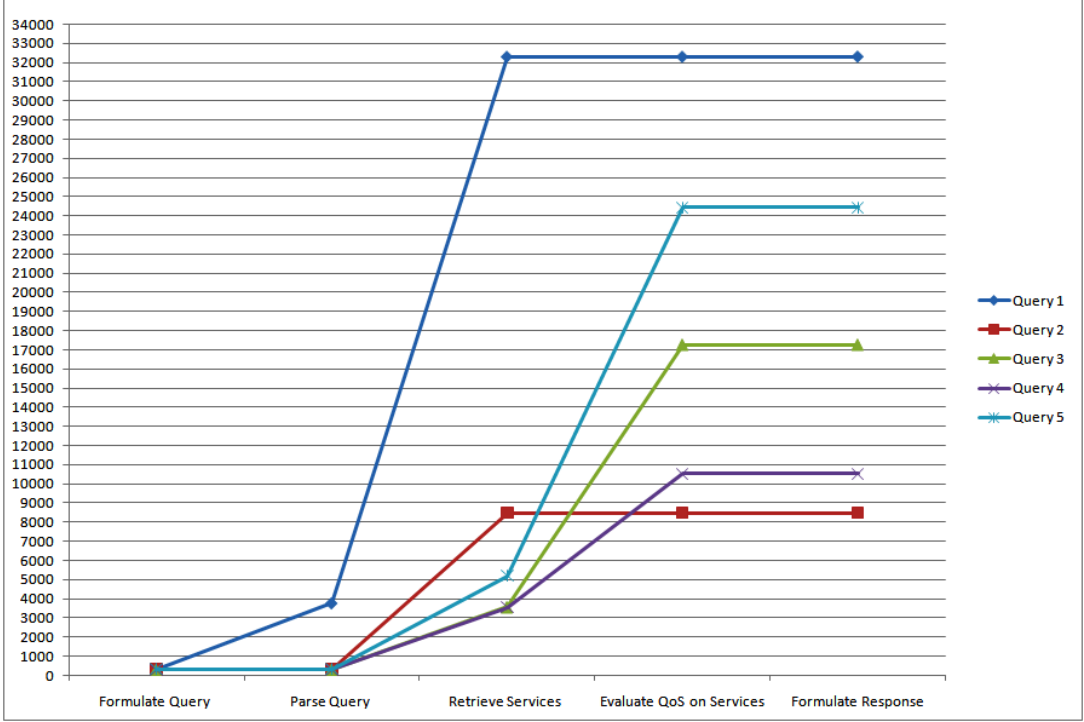


Figure 7.16: Query Execution Time Performance Graph

## **CHAPTER 8**

### **CONCLUSION**

In this thesis, we have developed a sub-module of the DSWSD-S System that takes place in domain-specific service discovery process. The main idea of the work proposed is to design and implement a supportive graphical user interface that enables the users to find the desired web services satisfying the user's needs. In addition to this, with the S-USQL query language we have defined in this work, we aimed to make the DSWSD-S System comply with the standards in the literature and as a result of this make it a scalable, interoperable and flexible system.

The proposed works about the graphical query interfaces in the literature are not developed based on the user-interaction. Therefore, the web service consumers are expected to know the technical details of the query structures. This makes the web service discovery process a challenging task for the end-users. The graphical user interface that is proposed within this thesis stands out with its user-friendly and supportive design. It guides the user to specify the search criteria and the parameters in an easy way that the users are encapsulated from the technical details of the query language, S-USQL, used in the DSWSD-S System.

In the future, proposed graphical user interface can be extended with new search criteria specification widgets. New widgets can be added to the interface for retrieving the input and output parameters for the desired web service operations, which can already be specified via S-USQL, from the user.

In addition to this, both the graphical user interface and the query language can be extended with the fuzzy quality of service parameters. By saying fuzzy quality of service parameters, we mean specifying the quality of service parameters linguistically instead of specifying them numerically. Because expressing the needs in a linguistic way can be more meaningful and

easy for the users. These new fuzzy parameter types can be “low”, “medium”, “high” and so on. S-USQL can also be extended to allow formulating more complex query sentences that mix both the numerical and fuzzy parameter specifications. For example the user can specify the availability parameter’s weight as 50 while specifying the reliability parameter as “high” within the same query sentence.

## REFERENCES

- [1] Aphrodite Tsalgatidou, Michael Pantazoglou. *The Unified Service Query Language Technical Report*. <http://www.s3lab.com/usql-tr.pdf>, Last accessed on July 18, 2011.
- [2] Aphrodite Tsalgatidou, Thomi Pilioura. *Unified Publication and Discovery of Semantic Web Services*. ACM Transactions on the Web (TWEB), vol. 3 Issue 3, June 2009.
- [3] A. Tsalgatidou, M. Pantazoglou, G. Athanasopoulos. *Specification of the Unified Service Query Language v1.0(USQL)*. EU Sixth Framework Programme - SODIUM Project. <http://www.atc.gr/sodium/upload/publications/D8-Specification%20of%20USQL.zip>, Last accessed on July 18, 2011.
- [4] M. Tian, A. Gramm, H. Ritter, J. Schiller. *Efficient Selection and Monitoring of QoS-aware Web services with the WS-QoS Framework*. Proc. Of IEEE/WIC/ACM International Conference on Web Intelligence, 2004.
- [5] Hai Zhuge, Jie Liu. *Flexible Retrieval of Web Services*. Elsevier Journal of Systems and Software, vol. 70, no. 1- 2, Pages 107-116, 2004.
- [6] Delnavaz Mobedpour, Chen Ding, and Chi-Hung Chi. *A QoS Query Language for User-Centric Web Service Selection*. IEEE International Conference on Services Computing, 1:1-8, 2010.
- [7] Deniz Cantürk and Pınar Şenkul. *Using Semantic Information for Distributed Web Service Discovery*. International journal of Web Science, in press.
- [8] Deniz Cantürk and Pınar Şenkul. *Service Acquisition and Validation in a Distributed Service Discovery System Consisting of Domain-Specific Sub-Systems*. Proc. Of ICEIS, 1:93-99, 2010.
- [9] Universal Description, Discovery and Integration (UDDI). <http://uddi.xml.org/>, Last accessed on July 18, 2011.
- [10] Extensible Markup Language (XML) v1.0. <http://www.w3.org/TR/2008/REC-xml-20081126/>, Last accessed on July 18, 2011.
- [11] XML Schema Definition (XSD). <http://www.w3.org/XML/Schema>, Last accessed on July 18, 2011.
- [12] Electronic Business using eXtensible Markup Language (ebXML). <http://www.ebxml.org/>, Last accessed on July 18, 2011.
- [13] Eyhab Al-Masri and Qusay H. Mahmoud. *WSCE: A Crawler Engine for Large-Scale Discovery of Web Services*. IEEE International Conference on Web Services (ICWS), 2007.
- [14] Eyhab Al-Masri and Qusay H. Mahmoud. *Investigating Web Services on the World Wide Web*. Proceeding of the 17th international conference on World Wide Web, 2008.

- [15] Eyhab Al-Masri and Qusay H. Mahmoud. *Crawling Multiple UDDI Business Registries*. Proceedings of the 16th international conference on World Wide Web, 2007.
- [16] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller. *A Concept for QoS Integration in Web Services*. Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops, 2003.
- [17] Stefan Schulte, Melanie Siebenhaar, Julian Eckert, Ralf Steinmetz. *Query Languages for Semantic Web Services*. <ftp://ftp.kom.tu-darmstadt.de/papers/SSES10.pdf>, Last accessed on July 18, 2011.
- [18] A. Soydan Bilgin and Munindar P. Singh. *A DAML-Based Repository for QoS-Aware Semantic Web Service Selection*. Proceedings of the IEEE International Conference on Web Services, 2004.
- [19] Hai Wang and Sheping Zhai. *Query for Semantic Web Services Using SPARQL-DL*. Second International Symposium on Knowledge Acquisition and Modeling (KAM), 2009.
- [20] Kashif Iqbal, Marco Luca Sbodio, Vassilios Peristeras and Giovanni Giuliani. *Semantic Service Discovery using SAWSDL and SPARQL*. Fourth International Conference on Semantics, Knowledge and Grid (SKG), 2008.
- [21] Tanu Malik Alex, S. Szalay, Tamas Budavari and Ani R. Thakar. *SkyQuery: A Web Service Approach to Federate Databases*. Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR), VLDB Endowment, 2003.
- [22] Sebastian Günther, Claus Rautenstrauch and Niko Zenker. *Service-Oriented Architecture: Introducing a Query Language*. Multikonferenz Wirtschaftsinformatik , 2008.
- [23] Weilong Ding, Jing Cheng, Kaiyuan Qi, Yan Li, Zhuofeng Zhao and Jun Fang. *A Domain-specific Query Language for Information Services Mash-up*. IEEE Congress on Services - Part I, 2008.
- [24] Min Liu, Weiming Shen, Qi Hao and Junwei Yan. *An Weighted Ontology-Based Semantic Similarity Algorithm For Web Service*. Journal of Expert Systems with Applications: An International Journal Archive, Volume 36 Issue 10, December 2009.
- [25] Glen Dobson, Russell Lock and Ian Sommerville. *QoSOnt: a QoS Ontology for Service-Centric Systems*. 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005.
- [26] Wolfgang Hoschek. *The Web Service Discovery Architecture*. Proceedings of the 2002 ACM/IEEE conference on Supercomputing.
- [27] Vladimir Tasic, Kruti Patel and Bernard Pagurek. *WSOL - Web Service Offerings Language*. Proceedings of the International Workshop on Web Services, E-Business, and the Semantic Web, 2002.
- [28] Don Chamberlin, Jonathan Robie and Daniela Florescu. *Quilt: An XML Query Language for Heterogeneous Data Sources*. International Workshop on the Web and Databases (WebDB) , pp. 53-62, 2000.
- [29] *WSEXPRESS: A QoS-Aware Search Engine for Web Services*. IEEE International Conference on Web Services (ICWS), 2010.

## APPENDIX A

### XSD OF S-USQL

This appendix contains the formal XML Schema Definition (XSD) of the Simplified - Unified Service Query Language (S-USQL), version 1.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:dswsd:SUSQL"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="urn:dswsd:SUSQL" xmlns="urn:dswsd:SUSQL"
elementFormDefault="qualified">

<!-- Simplified Unified Service Query Language (S-USQL) version 1.0 -->

<xs:attributeGroup name="basicRequirementAttributes">
<xs:attribute name="nullAccepted" type="xs:boolean"
use="optional" default="false" />
<xs:attribute name="minDegreeOfMatch" type="tns:tPercentage"
use="optional" default="1.0" />
</xs:attributeGroup>

<xs:group name="AdditionalServiceProperties">
<xs:sequence>
<xs:element ref="tns:ServiceDescription" minOccurs="0" />
<xs:element ref="tns:QoS" minOccurs="0" />
</xs:sequence>
</xs:group>
```

```

<xs:simpleType name="tPercentage">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="1" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tWeight">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="100" />
  </xs:restriction>
</xs:simpleType>

  <xs:simpleType name="tSyntacticSemanticOperators">
    <xs:restriction base="xs:string">
      <xs:enumeration value="equal"/>
      <xs:enumeration value="contain"/>
      <xs:enumeration value="exact"/>
      <xs:enumeration value="abstraction"/>
      <xs:enumeration value="extension"/>
      <xs:enumeration value="sibling"/>
    </xs:restriction>
  </xs:simpleType>

<xs:complexType name="tSyntacticSemantic">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attributeGroup ref="tns:basicRequirementAttributes"/>
      <xs:attribute name="valueIs" type="tns:tSyntacticSemanticOperators"
use="optional" default="equal"/>
    </xs:extension>

```



```

    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="ServiceName" type="tns:tSyntacticSemantic" />

  <xs:element name="ServiceDescription" type="tSyntacticSemantic" />

  <xs:element name="ServiceDomain" type="tSyntacticSemantic" />

  <xs:element name="Price" type="xs:float" />

  <xs:element name="Availability" type="tns:tWeight" />

  <xs:element name="Reliability" type="tns:tWeight" />

  <xs:element name="ProcessingTime" type="tns:tWeight" />
  ,
  <xs:element name="Throughput" type="tns:tWeight" />

  <xs:element name="QoS">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:Price" minOccurs="0" />
        <xs:element ref="tns:Availability" minOccurs="0" />
        <xs:element ref="tns:Reliability" minOccurs="0" />
        <xs:element ref="tns:ProcessingTime" minOccurs="0" />
        <xs:element ref="tns:Throughput" minOccurs="0" />
        <xs:element name="qosScore" type="xs:double"/>
        <xs:any namespace="##other" minOccurs="0"
processContents="skip" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```

<xs:element name="RequestOperation">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="Name" type="tSyntacticSemantic" minOccurs="0" />
      <xs:element name="Capability" type="tSyntacticSemantic" minOccurs="0" />
      <xs:element name="Inputs" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="input" type="tns:tParameter"
minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attributeGroup ref="tns:basicRequirementAttributes" />
        </xs:complexType>
      </xs:element>
      <xs:element name="Outputs" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="output" type="tns:tParameter"
minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attributeGroup ref="tns:basicRequirementAttributes" />
        </xs:complexType>
      </xs:element>
      <xs:element ref="tns:QoS" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="minDegreeOfMatch" type="tns:tPercentage"
use="optional" default="1.0" />
  </xs:complexType>
</xs:element>

```

```

<xs:complexType name="tParameter">
  <xs:sequence>
    <xs:element name="name" type="tns:tSyntacticSemantic" minOccurs="0"/>
    <xs:element name="type" type="tns:tSyntacticSemantic" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="tns:basicRequirementAttributes"/>
</xs:complexType>

<xs:element name="RequestService">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element ref="tns:ServiceName" minOccurs="0" />
      <xs:element ref="tns:ServiceDescription" minOccurs="0" />
      <xs:element ref="tns:ServiceDomain" minOccurs="0" />
      <xs:element ref="tns:QoS" minOccurs="0" />
      <xs:element ref="tns:RequestOperation" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="minDegreeOfMatch" type="tns:tPercentage"
use="optional" default="1" />
  </xs:complexType>
</xs:element>

<xs:element name="ResponseOperation">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="degreeOfMatch" type="tns:tPercentage"
use="required" />
  </xs:complexType>
</xs:element>

```

```

<xs:element name="ResponseService">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element ref="tns:ServiceName" />
      <xs:element name="descriptionDocUrl" type="xs:anyURI"/>
      <xs:group ref="tns:AdditionalServiceProperties" />
      <xs:element name="interface" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="tns:ResponseOperation" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="name" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="degreeOfMatch" type="tns:tPercentage"
use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="ResponseServices">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:ResponseService" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="SUSQLRequest">
  <xs:complexType>

```

```

<xs:sequence>
  <xs:element name="Where">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:RequestService" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="numberOfResults" type="xs:int" />
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

```

```

<xs:element name="SUSQLResponse">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="tns:ResponseServices" />
      <xs:element name="Error">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="code" type="xs:string" />
            <xs:element name="desc" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="SUSQL">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="tns:SUSQLRequest" />
    </xs:choice>
  </xs:complexType>
</xs:element>

```

```
<xs:element ref="tns:SUSQLResponse" />
</xs:choice>
<xs:attribute name="version" type="xs:float" use="required"
fixed="1.0" />
</xs:complexType>
</xs:element>
</xs:schema>
```

## APPENDIX B

### SIMILARITY DEGREE CALCULATOR WEB SERVICE

This appendix lists the operations which are offered by the Similarity Degree Calculator Web Service.

Table B.1: Operations Offered By Similarity Degree Calculator Web Service

<b>Operation Name</b>	<b>Operation Summary</b>
GetSimilarity	Calculates the similarity degree between given two sentences
GetSimilarityUsingStrategy	Calculates the similarity degree between given two sentences according to the given strategy

## APPENDIX C

### DATABASE OPERATIONS HANDLER WEB SERVICE

This appendix lists the operations which are offered by the Database Operations Handler Web Service.

Table C.1: Operations Offered By Database Operations Handler Web Service

<b>Operation Name</b>	<b>Operation Summary</b>
createDatabases	Constructs the crawler engine local database
sortServicesbyQOS	Sorts the resulting web services by their QoS score
getServiceDescriptionAndMathcingDegree	Returns the web service and the matching degree
getAllURLsFromServiceDescriptionTable	Returns the web service URLs
getOntologies	Returns the all the domain ontologies' root elements
getOntology	Returns the details of a selected ontology
insertURL	Inserts a web service URL
insertServiceResponse	Inserts the response of a web service
insertServiceDescription	Inserts a web service description
insertQOSInformation	Inserts a web service QoS information
insertMatchingDegree	Inserts the matching degree of a web service with the domain ontology
getMatchingDegreesofServices	Returns the matching degree of web services with the domain ontology