TRADEOFFS IN GOOGLE DISTANCE ONLY VS A WORDNET HYBRID FOR QOS-ENABLED WEB SERVICE COMPOSITION

by

Dhivya Veerasekaran

Submitted
in partial fulfilment of the requirements
for the degree of Master Of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2012

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "TRADEOFFS IN GOOGLE DISTANCE ONLY VS A WORDNET HYBRID FOR QOS-ENABLED WEB SERVICE COMPOSITION" by Dhivya Veerasekaran in partial fulfilment of the requirements for the degree of Master of Computer Science.

Dated: July 17th, 2012

Co-Supervisors: _____

_____

Readers: _____

_____

DALHOUSIE UNIVERSITY

DATE: July17, 2012

AUTHOR: Dhivya Veerasekaran

TITLE: TRADEOFFS IN GOOGLE DISTANCE ONLY VS A WORDNET HYBRID FOR QOS-ENABLED WEB SERVICE COMPOSITION

DEPARTMENT: Faculty of Computer Science

DEGREE: MCSc CONVOCATION: October YEAR: 2012

_____

Signature of Author

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis proposes a hybrid approach in using Google Distance and WordNet together in a new method, called the Domain Independent Quality of Service (DIQOS) method for QoS-enabled web services discovery. Comparisons, using delay, recall, and precision metrics, between this hybrid approach and an earlier lightweight Google Distance-only based approach for web services discovery are provided. Further, our performance evaluation demonstrates as of yet undocumented trade-offs between Google distance, Google-WordNet distance, and WordNet distance approaches for similarity matching in the web services discovery phase. The impact of all approaches on QoS-enabled web service composition is described for representative web transactions in the travel domain**.** Findings include that the recall of signature matching increases by 5 to 15% for WordNet-assisted Google Distance DIQOS approach over the pure Google-distance DIQOS variant. WordNet-assisted Google Distance also shows 20-40 % increases in recall for signature matching compared to the WordNet only approach. Also, bigram-based, short sentence, and WordNet-based vector optimizations in specification similarity matching show an average of 25% increase in recall over a previous competitive method called the Flexible Ontology-Independent QOS-enabled method (FOIQOS). Our WordNet-assisted Google Distance method shows 34% increase in recall compared to FOIQOS. Our approaches produce 11% lower precision than FOIQOS, but FOIQOS is speedier as delays are 5 to 15 % lower.

LIST OF ABBREVIATIONS USED

BPEL	Business Process Management Language
BPEL4WS	Business process Execution Language for Web Service
DIQOS	Domain Independent Quality Of Service
FOIQOS	Flexible Ontology Independent Quality Of Service
HTTP	Hypertext Transfer Protocol
NGD	Normalized Google Distance
POS	Parts of Speech
QoS	Quality of Service
SOAP	Simple Object Access Protocol
TF-IDF	Term Frequency / Inverse Document Frequency
TTP	Trusted Third Parties
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
URBE	UDDI Registry By Example
W3C	World Wide Web Consortium
WSCI	Web service Choreography Interface
WSDL	Web Service Description Language
WSLA	Web Service Level Agreement
XML	eXtensible Markup Language

ACKNOWLEDGEMENTS

This thesis work would not have been possible without the support of many people.

It is difficult to overstate my gratitude to my supervisor, Dr Dawn Jutla, who supplied me with the thesis topic and several technical ideas. With her knowledge, enthusiasm, inspiration and efforts to explain things clearly and simply, she offered me invaluable assistance, support and primary thesis guidance. She patiently read through several iterations of my thesis write-up and provided valuable feedback. I would also like to thank her former Master student at Saint Mary's, Mr. Rui Ding, whose thesis work and code I extended.

I would like to offer my heart-felt gratitude to my Dalhousie co-supervisor, Dr. Peter Bodorik without whose assistance this study would not been possible.

I would also like to extend my thanks to my committee members, Dr. Vlado Keselj and Dr. Dirk Arnold for their advice, time, and support. I would also like to thank Dr. Denis Riordan for performing the service of Chair of my thesis defence.

I would also like to take this opportunity to express my profound gratitude to my beloved parents and my husband Mr. Arjun Nagarasan, whose love and support made my study possible and fruitful.

I am indebted to Rajesh Franklin Anbarasu for his invaluable assistance during the entire duration of my thesis work. I wish to thank my roommates for all the support, camaraderie, entertainment and caring they provided.

CHAPTER 1 : INTRODUCTION

The rapid proliferation of Web Service providers and cloud computing over the Web is triggering much research in Web Service discovery and composition. Several investigations have been performed in ontology-dependent and ontology-independent Web Service discovery (Maximilien, 2004; Ding & Jutla, 2011). While pure statistical approaches (Ding & Jutla, 2011) can provide ontology-independence and speed, they do not focus on semantic matching. The pure ontology approaches (Maximilien, 2004; Oundhakar et al, 2005; Sanchez, 2008) are also problematic when ontologies are badly maintained or unavailable in a domain, or the high costs of maintenance exceed their benefits. In this thesis work, a hybrid approach is used to capture both ontology independence and a deeper focus on semantics.

Popular Web resources, such as Google Search (Vitanyi and Cilibrasi, 2007) and WordNet (Miller, 1995), are used individually in previous Web Services discovery models, and offer advantages to Web Services discovery. For example, WordNet, a lexical database is widely used in similarity matching methods for service discovery (Wang et al, 2003; Li et al, 2006; Ma et sl, 2008; Chen et al, 2010). The usage of Google Distance as a similarity matching method for Web Services discovery first appeared in independent Web Service discovery proposals, in the work of Ding and Jutla (2011), followed by Yang et al (2011). In our research work, the Google Distance method and a WordNet-based distance measure are used in lockstep within a novel strategy for Web Services discovery purposes, and examined together in the context of selecting the best QoS-enabled dynamic Web Services composition.

Similarity matching is employed to match service requests to candidate services in the discovery phase of dynamic Web Services composition. Similarity matching itself supports three levels of matching: lexical, semantic, and operational. The levels of matching are normally performed in that order. However, the shortcomings of many similarity matching approaches in service discovery, at all levels, include their use of non-representative methods. For example, some methods use "long-text methods", such as TF-IDF, for processing short-sentence WSDL service descriptions (Segev, 2009). Other approaches may use high cost or poorly maintained support ontologies with missing relationships that eliminate partially matched services prematurely. Indeed, ontologies are often absent or poorly maintained in many domains, including business and public policy domains. Automated reasoning over badly maintained domain ontologies is not useful, creates new problems, and certainly incurs overhead in terms of delay. This thesis intends to ameliorate the above issues in a new approach for Web Services discovery.

WordNet may be converted into a well-maintained general ontology that spans vocabulary across all domains. However, domains that have specialized vocabulary, for example the health domain, are not be fully covered by WordNet. Intuitively, Google Distance's use of the Web search space provides compensatory coverage for words that are not in the WordNet lexical database. Further, Google Distance is useful for service discovery in domains where ontologies do not exist or that lack a well-defined ontology.

Google Distance also possesses certain problems in finding similarity between two semantic words (e.g. car boot and car trunk), which do not usually appear together in

Web pages. Therefore WordNet-assisted Google Distance may provide a more valid similarity measure.

In this research work we compare our new hybrid approach, called the Domain Independent QoS-enabled (DIQOS) approach with a lightweight ontology-independent Google-Distance only approach, named the Flexible Ontology-Independent QoS-enabled (FOIQOS) approach (Ding & Jutla, 2011), for automatically discovering and selecting Web Services for dynamic composition. Both approaches incorporate QoS parameters to meet predefined application-level QoS objectives.

1.1 RESEARCH PROBLEM

The key problems with ontology-dependent approach in Web Service composition are poor and high cost maintenance of the ontologies, and unavailability of ontologies in many domains. Dynamic Web Service composition over a badly maintained ontology leads to various issues such as overhead delay and inefficient composition solution. In comparison, ontology-independent approach executes with great speed but it lacks in capturing semantic meaning of service specification in Web Service discovery.

1.2 RESEARCH OBJECTIVES

The objectives of the thesis are as follows.

1) Create a novel variant of the ontology-independent Google distance method for similarity matching, used within the Flexible Ontology-Independent Quality of Services (FOIQOS) method (Ding and Jutla, 2011), for Web Services discovery by:

a) adding more semantics to a discovery solution via exploring the hybridization of the Google-distance solution with WordNet.

b) adding more appropriate methods in similarity matching to leverage known characteristics of Web Services.

2) Evaluate the impact of the resulting new service discovery method on QoS-enabled best composition.

3) Identify the qualitative and quantitative trade-offs of the Google Distance-only based FOIQOS method (Ding and Jutla, 2011) and this thesis' new method for QoS-enabled services discovery and composition.

1.3 THESIS ORGANIZATION

The thesis organization is as follows. Chapter 2 provides background and literature review for web services discovery and composition. Chapter 3 presents the proposal for a new approach for Web Services discovery. Chapter 4 describes the implementation environment and step-by-step procedure for implementing the proposed methods. Chapter 5 presents the performance evaluation of the proposed approach by comparing it with a predecessor ontology-independent Google distance approach. Finally, conclusions and future work are described in Chapter 6.

CHAPTER 2 : BACKGROUND AND LITERATURE REVIEW

2.1 WEB SERVICE - OVERVIEW

The W3C defines a Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network" (W3C, 2004). Web Services are used as software building blocks in a variety of service-oriented application domains such as business application integration and business information management. The functionality of a business task can be accomplished either by a Web Service or composites of Web Services. Web Services are loosely coupled, distributed and independent software entities, which are easily maintained and simple to access. They are described using standards (WSDL) that provide interoperability between different applications from different sources without consuming much time (Cerami, 2002). Web Services are independent of operating system, programming language, or software application being used.

Business organizations are creating better applications by reusing their own Web Services and other Web Services that are published and discovered over the Internet or Intranet by using a stack of standards (Heather, 2001) such as SOAP, XML, UDDI, and WSDL. SOAP (Simple Object Access Protocol) (Mendelsohn et al., 2000), is used to transfer data; XML (eXtended Markup Language) (Bray et al, 2008), represents data in a structured manner using customized tags; WSDL (Web Service Description Language) (Meredith et al, 2001) is used to describe the service information; UDDI (Universal Description, Discovery, and Integration) (Bellwood et al, 2002) acts as a repository, used by the service providers to publish their services and aids consumer in discovering the

expected service to accomplish their tasks. Due to the extensive growth of Web Services in the repository, selecting the best Web Service becomes difficult and challenging.

2.2 WEB SERVICE MODEL

The Web Service model (Ran, 2003) is composed of an interaction between service providers, service consumers, and service registries. It involves functionalities such as publish, find and bind operations. The service providers offer various Web Services (i.e. the business logic or functions that are published over the public registries) to aid their customers and partners who could invoke it on demand. The service consumer requests the Web Service by providing the requirements to the registry. A service registry is the repository of Web Services, in which service discovery is carried out to find the relevant Web Service for users' request and it sends its response (service description) to the consumer. The service consumer obtains the binding information from the service description, which aids to initiate and invoke the services from the providers' end. Figure 2.1 (Ran, 2003) illustrates the service roles and their service functionalities involved during the interactions.

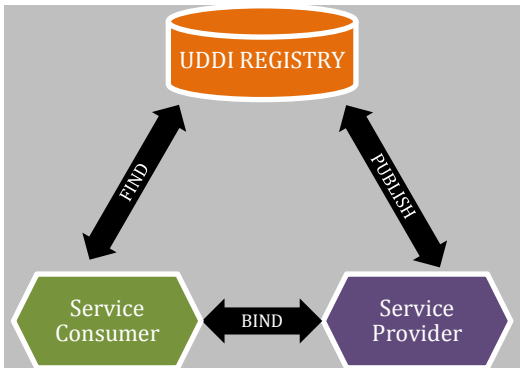

Figure 2.1 : Web Service model

2.2.1 Web Service Process

Publish: Service providers publish their service information i.e. service description, service name, input and output parameters by sending these as a WSDL file to the UDDI registry so that a consumer can find it. The UDDI registry helps to discover Web Services requested by the service consumer.

Find: Service consumers describe requirements to locate the service providers from the UDDI registry. The service requirements are provided as service profile, WSDL, or service template.

Bind: The registry provides WSDLs of the relevant service providers to service consumer. The service consumer initiates or invokes the service providers at run-time using binding information in the service description. The service providers and consumers will communicate with each other using SOAP message, which is an XML-based protocol for Web Service exchange information.

2.2.2 Web Services Stack and Technologies

The Web Service processes are performed using protocol standards represented in each level of Web Service stack. Figure 2.2 (Heather, 2001), shows the conceptual Web Services stack, in which each layer is stacked on one another. The right side vertical bar represents requirements that must be addressed at each level on the stack. The left side annotation represents the standard technologies applied on the stack.

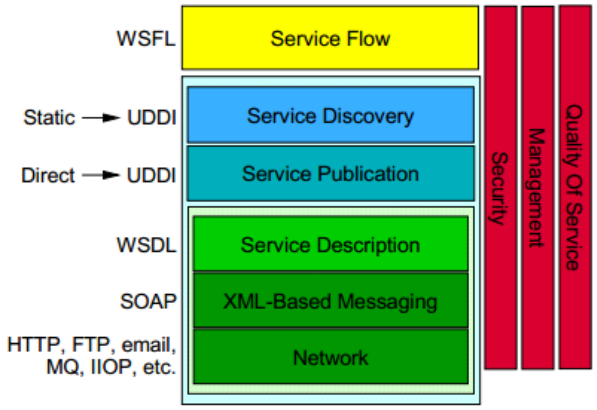

Figure 2.2 : Web Service conceptual stack

XML is the markup language used to represent data in a structural format (Bray, et al., 2008). It can be readable by both human and machine. It is a flexible and a simple data format that helps in exchange of data over the Internet. It is used for almost all the Web Service technologies such as SOAP, WSDL, and UDDI, in which XML is used to represent data. XML is represented in the form of tags, which can be created by the users. It does not have predefined tags unlike HTML. Users create their own tags for their task. A sample XML document is shown below:

```
<? xml version="1.0" encoding="UTF-8"? >
<Customer id=1>
 <Contact>
        <Name>John</Name>
        <PhoneNo >3045678933</PhoneNo>
</Contact>
</Customer>
```

The network is used as the foundation of the Web Services stack. The standard network protocols such as HTTP, FTP, MQ, email, IIOP, etc. can be used for Web Service communication over the Internet.

The next level protocol in the stack, SOAP, is used to exchange structural information in the implementation of Web Services over computer networks (Mendelsohn et al., 2000). SOAP is platform independent and helps exchange of messages among different applications. It relies on XML for a data messaging format and network protocols for message negotiation and transmission. A sample skeleton of SOAP message is shown in the example below:

```
<?xml version="1.0"?>
<soap:Envelope xmlns: soap="http://www.w3.org/2001/12/soap-envelope" >
 <soap:Header>
</soap:Header>

<soap: Body>
<-- Fault element is optional, used only if a fault occurs in web service.   -->
        <soap: Fault>   </soap: Fault>
</soap:Body>
</soap:Envelope>
```

Envelope: It is the root element of the SOAP message. It is mandatory in SOAP message data format, which contains header and body. It indicates the start and end point of the message, so that receivers can differentiate between the messages.

Header: It is an optional part of the SOAP message, which can also have multiple headers in the message part. It offers a framework for specifying application-level requirement.

Body: It is a mandatory element, which is defined within envelope, and must follow any header. It specifies the application-defined XML data that is exchanged in the SOAP message.

Fault: It is defined only once within the body element. When an error occurs during processing, the fault message, which is predefined in the SOAP message, is popped out as a response to the sender.

The service description layer is a stack of description documents. WSDL is an XML based protocol for exchanging information in decentralized and distributed environment (Meredith et al., 2001). It is the standard format for describing the Web Services. It describes how to access the service and what operations it would perform. It can be created manually by XML editors or automatically by using tools like Java2WSDL from the existing service interfaces. Table 2.1 (Meredith et al, 2001) illustrates the seven major elements in the WSDL element.

Table 2.1 : WSDL document structure

WSDL Element	Definition
Types	It contains typed definitions that are used in the information
Message	It represents the data communicated. The data may contain one or
Operation	It specifies the operation performed by the service
Port Type	It defines set of operation supported by one or more endpoints.
Binding	It specifies the concrete protocol and data format specification for
Port	It is an address, combination of binding and network address. It
Service	It aggregates the sets of related endpoints that are used in the

The service publication and service discovery are the next level of layers on the stack; it uses the UDDI registry, a standard mechanism for publishing and discovering Web Services. The UDDI has three components similar to the real-world White pages, Yellow pages, and Green pages. Usually, the service or business is classified under the taxonomies provided by the Yellow pages (Bellwood et al, 2002). The UDDI can also be customized by defining the data structure for representing service description information

in XML that provides a web based user interface to publish and query business information (Ran, 2003). The service providers make their service available to the public by publishing them in the public/private registries like UDDI. The consumer can get access to the published service by querying UDDI with its requirements. The discovery is done either at a consumer agent or at a provider agent. The service descriptions are published using various mechanisms such as direct publish (providers send service description directly to consumers through email or FTP), HTTPS mechanism (through URL) and UDDI registries (Heather, 2001). The services discovered depend on the services published by the providers. It can either be at design time i.e. searching for Web Service descriptions by the type of interface or at runtime depending on the qualities of service advertised.

2.3 WEB SERVICE DISCOVERY

Web Service discovery is defined as "the act of locating a machine-processable description of a Web Service that may have been previously unknown and that meets certain functional criteria" (Booth et al, 2004). The Web Service provider publishes their services e.g. stockExchange, and the service consumer uses that service. The Web Service discovery is the act of finding suitable services for the requested task. There are various discovery mechanisms involved such as registries, index, catalogues and peer-to-peer (P2P) solutions. Among these techniques, registries, index and peer-to-peer solutions are discussed below.

Registry

Registries (Booth et al, 2004) are the authorized and centralized repositories where Web Services are published. The registry owner has the power to decide who can publish and update the service information. The registry provides authentication to the service providers so that the other parties cannot modify their service. UDDI is an example of a registry, which is public as well as private by hosted organizations. Some examples of businesses that hosted public UDDIs are Microsoft, IBM and SAP, but they discontinued the practice a couple of years ago. Moreover, organizations are creating their own private UDDI and they are not willing to make them Public. XMethods, WebServiceList, and WebServiceX are some of Web Service collection repositories available public but information provided is not authorized and updated.

Index

An index (Booth et al, 2004) is a reference to the collection of information published by the service provider. Service provider exposes service description and functionality of the Web Service on the Web. The index can be created by anyone and any interested index owners can invoke the service information published by the providers without their knowledge. The information contained in an index refers to the authoritative information, which can be verified before use. Google search is an example of the index approach.

P2P

P2P (Booth et al, 2004) is a decentralized approach in which Web Services are discovered dynamically. Unlike a centralized repository, Web Services are distributed as nodes in the network. A Web Service query is propagated within the network peers until the requirement matches with one of those peer Web Services specifications. It can be a better approach than a registry since it does not need a centralized registry, which could cause congestion and failure. But in another aspect of performance, it costs more because the information is spread in the network, which increases the network traffic (Booth et al, 2004). If peers are unavailable, performance can also suffer.

2.3.1 UDDI Registry

The UDDI registry (Bellwood et al, 2002) is the repository where the Web Service providers publish Web Services. When the Web Service consumer requests the service, discovery mechanism is used to select the relevant service among the services stored in the registry. In order to minimize the effort of searching for services in the collection across various domains, the UDDI registry is extended to support categories, which allow Web Services to be published in the specific domain. The Web Services are categorized using tmodel template. For example, airlines service providers can publish their service in the 'Travel' category and a consumer can query that specific category and look for the required service among collections. The UDDI registry can store information such as business description, services provided by the business, description about the services, and description of technical interfaces. It takes four core data structures such as business entity, business service, business template, and business tModel. The UDDI data structure and their relationship are shown in the Figure 2.3 (Bellwood et al, 2002).

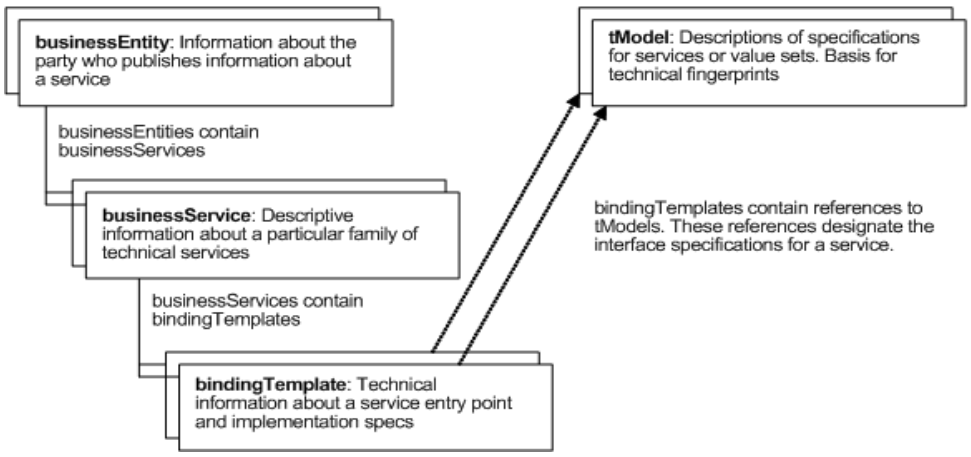

Figure 2.3 : UDDI data structure

businessEntity: It provides information about the provider who publishes the Web Service. The information such as contact address, business identifier, industry category and list of services are stored in the UDDI registry.

businessService: It describes the details of an individual Web Service in the business entity. It takes information such as, what type of Web Service, how to bind it and what kind of taxonomical category it belongs to.

bindingTemplate: It describes the technical information about the Web Service. A single business service can have multiple binding templates; each represents an actual implementation of the Web Service.

tModels: It is the way of describing the business service and binding templates in the UDDI registry. Any abstract concept can be registered within UDDI as tModels.

2.3.2 Qos Requirement for Web Service and Storage in UDDI Registry

QoS (Quality of Service) is defined as "a combination of several qualities or properties of a service" (Menasce, 2002). QoS are the non-functional attributes used to qualify the Web Services requested by the user. It is used to refine the relevant Web Services, with properties similar to the functional attributes. A QoS parameter is considered as an important attribute to both service providers and service consumers. There are several parameters involved in measuring the quality of the Web Service such as reliability, cost, time, performance, availability and capacity. In this thesis, we use the first three parameters i.e. reliability, cost, and time, for measuring quality of service.

The current UDDI registries are extended to support non-functional requirements by categorizing tModels for the quality of service (QoS). Earlier tModels are categorized to store technical description of the Web Services. The tModels contain information such as key, name, URL pointing to the location where the actual concept represented by the tModel can be found and optional description information. Blum (2004) proposed tModels, for the quality of service that assists to provide QoS information on the binding templates. The following example illustrates how the bindingTemplate refers to the tModel with QoS information.

```
<tModel tModelKey= "mycompany.com: FlightBookingService:
PrimaryBinding:QoSInformation">
        <name>QoS Information for Stock Quote Service</name>
        <overviewDoc>
                <overviewURL>http://<URL describing schema of QoS attributes>
                </overviewURL>
        </overviewDoc>
        <categoryBag>
                <keyedReference tModelKey="uddi:uddi.org:QoS:ResponseTime"
                keyName="Average ResponseTime" keyValue="63" />
```

```
            <keyedReference tModelKey="uddi:uddi.org:QoS:Throughput"
            keyName="Average Throughput" keyValue=">10Mbps" />
            <keyedReference tModelKey="uddi:uddi.org:QoS:Reliability"
            keyName="Average Reliability"   keyValue="50" />
        </categoryBag>
</tModel>
```

The tModel contains 'category Bags' to store the QoS metrics. The QoS metrics are represented by name and value pairs, which are 'keyedReference', a general data structure. The tModelkey in each 'keyedReference' acts as a namespace. It also assists in providing a unique naming scheme.

2.3.3 Research in QoS-based Web Service Discovery

Dong et al (2004) uses the Woogle, a customized Web Service search engine to find the relevant Web Services for user requirements. It collects Web Services from various sites such as WebserviceList, XMethods, BindingPoint, and RemoteMethod, and clusters the parameter names in the collection of the Web Services into semantically meaningful concepts. An agglomerative clustering algorithm is used to classify service parameter names into concepts, which is helpful in identifying two Web Services operations with similar functionalities. Dong et al (2004) performs the Web Service matching based on its operation name and input/output parameters but Web Service specification matching is not implemented in their algorithm. Web Service specifications, such as service name and descriptions, provides brief indications about the functionality of the Web Service which helps in improving Web Service discovery (Medhi, 2011).

Wang and Stroulia (2005) compute the similarity between two Web Services using a semantic structural matching algorithm. They use a semantic information retrieval

approach (Miller, 1995; Faloutsos & Oard, 1995) to compute similarity between advertised and requested Web Service descriptions. Then they apply a structural similarity algorithm on the WSDLs to compute the signature similarity to filter irrelevant Web Services. The authors consider only functional attributes in the service discovery, but the service consumer needs to trust the retrieved service in terms of QoS. The non-functional attributes such as time, cost, and reliability are helpful in finding the reliability of the Web Service to satisfy the users' demands. But the importance of QoS-enabled service discovery is ignored in Wang and Stroulia (2005).

Xu et al (2006) proposes a reputation-enhanced Web Service discovery with QoS. Earlier discovery on UDDI was keyword-based matching, which was performed during design time on the functional attributes. Xu et al (2006) extends the UDDI data structure to support QoS along with functional requirements. The research work is mainly based on the maintenance of non-functional requirement (QoS) and dynamic discovery Web Services based on the QoS and reputation score computed from the feedback of the customer about the service at runtime.

Plebani and Pernici (2009) propose "UDDI Registry By Example" (URBE) a novel approach to compare the similarities between Web Service interfaces. The description of the Web Service interfaces such as operations' name, input and output parameters are retrieved from the WSDL to evaluate their similarity. The URBE approach considers service interface descriptions as a bag of isolated terms and applies WordNet to find the semantic distance between two terms. Finally, similarity of two services is calculated using a bipartite graph.

Yang et al (2011) computes the normalized Google Distance between concepts, based on semantic description of Web Service names, and input and output parameters to compare similarity between advertised service and requested service. The proposed algorithm on the Google Distance showed an improvement in the recall and precision values of Web Service discovery and composition over a classic OWL-S/UDDI algorithm in a distance education domain using 150 Web Services for course information, 150 Web Services for education affairs information, and 200 Web Services for a question database. The authors did not measure for best QoS-enabled Web Services composition, or quantify the specific impact on QoS-enabled services discovery that would be expected with selection mechanisms that exhibit higher recall and precision.

Ding and Jutla (2011) design a QoS-enabled ontology-independent approach using Google Distance to dynamically discover and compose Web Services. They use lexical, semantic and operational similarity matching algorithms to discover the relevant service. But the name and descriptions of the Web Services are not matched semantically at the discovery phase. Rather, the statistical-based Google Distance approach is used to identify the relevant service but it may have some loss of relevant Web Services as well. Consequently, it can fail to produce the best composition result. Ding and Jutla (2011) did not use recall and precision measures in their experiments.

2.4 WEB SERVICE COMPOSITION

In a large business organization, in many instances a single Web Service cannot satisfy the requirements of the users' requests. A Web Service needs to communicate with other Web Services to produce the required output. Web Service Composition provides an open, standard approach for connecting different Web Services, to create a

high-level business process. The standards such as BPEL4WS (Business process Execution Language for Web Service) (Andrew et al, 2003), WSCI (Web service Choreography Interface) (Arkin et al, 2001) and BPEL (Business Process Management Language) (Dumas et al, 2002) are primary standards supporting the Web Service composition.

Web Service composition is the process of composing one or more Web Services to achieve desired functionality. It can be categorized into manual or automatic composition. In manual composition, developers identify composite Web Services and carry out composition manually. But in automatic composition, a software agent selects the required services and performs composition using predefined algorithms. There are two types of Web Service compositions: static composition and dynamic composition. In static composition, the Web Services are fixed during design time. This type of composition is more suitable, if the service providers or partners involved in the process are predetermined and not likely to change. On the other hand, in dynamic composition the Web Services are decided at run time. It provides more flexibility for modifying and adapting the operation of the software at run time. There is no binding to any particular service provider.

2.4.1 Research in Dynamic Web Service Composition

Ponnekanti and Fox (2002) develop a toolset "SWORD" which allows developers to compose existing Web Services to new composite Web Services. In SWORD, the Web Services are represented in the form of rules, which contain preconditions, and post conditions. SWORD checks whether the service is capable of producing a particular output for the given inputs. In this method, the input and output are represented in a

"world-model" (Entity relationship based) and submitted to SWORD. The rule-based SWORD system is used to perform the Web Service composition and plan generation automatically with the provided input and output states.

Casati et al (2000), develops a system "eFlow" which provides a platform for specifying, enacting and managing composite services. The eFlow technique follows a static workflow generation methodology, which uses a graph to execute the order of the Web Services in the composition. The graph contains service, decision, and event nodes, which are created manually at design time. Web Service Composition is accomplished at runtime by executing the service node.

Polymorphic Process Model (PPM) (Schuster et al, 2000) uses a methodology that combines both static and dynamic Web Service composition. The static composition is supported by multi-enterprise process. It consists of abstract sub processes, which manages the functional descriptions but do not contain implementation. They are implemented by Web Services and binding is performed at runtime. The dynamic composition is supported by service-based process. The approach is modeled by a state machine, which assists in providing possible transition states for services and invocation.

Li and Xiang (2010) propose a new algorithm for Web Service composition, based on global QoS optimizing and Multi-Objective Chaos Ant Colony Optimization (MOCACO). MOCACO is used to select the services that satisfy QoS user constraints. The author conducts Web Service composition simulation on MOGA, MOACO and on their proposed model MOCACO to compare the solution quality and performance. The results show that MOCACO is more efficient than MOGA and MOACO in terms of

execution time and best composition solution. Many approaches (Jorge et al, 2004; Liu et al, 2004; Wan et al, 2007) were based on QoS local optimization or mono-objective and did not compare global QoS constraints in the Web Service selection. But MOCACO (Li & Xiang, 2010) is used to select services and optimize the different global QoS parameters to satisfy the user constraints.

Ding and Jutla (2011), propose Flexible Ontology-Independent and QoS-enabled dynamic Web Services composition using Google Distance (FOIQOS). They evaluate the efficiency of their proposed approach with the feature-based model (E-Workflow) (Caroso & Sheth, 2003) and the AI-based approach (MOACO), by calculating the total execution time and best composition solution. The result proves that FOIQOS outperforms significantly than ontology-dependent and heuristic-based methods in achieving increased accuracy and reduced overhead.

CHAPTER 3 : DIQOS - DOMAIN-INDEPENDENT QUALITY OF SERVICE ENABLED WEB SERVICE DISCOVERY

In this thesis, we propose a novel approach, DIQOS (Domain-Independent Quality of Service) for dynamic Web Service discovery and composition. This chapter discusses requirements, architecture and methods of our proposed DIQOS approach. The following sub-sections describe various optimized methods for Quality of Service (QoS)-enabled Web Service discovery and composition.

3.1 OVERVIEW

3.1.1 Requirements

Our proposed approach is designed to satisfy the following specific requirements for Web Service discovery and composition.

- Ontology-independence
- Universality through domain-independence
- Efficient in terms of time, recall and precision
- Optimized discovery process
- Low cost

3.1.2 Contribution

In this research, we propose and evaluate a novel hybrid approach to use Google Distance and WordNet together in the semantic similarity matching stage of Web Service discovery.

Normalized Google Distance (NGD) is a novel approach proposed by Cilibrasi and (2007) to calculate distance between pairs of words/terms to compare the similarity of two words. The World Wide Web (WWW) is a huge collection of documents that acts as a loosely unstructured or semi-structured database, which is mostly written in the natural language. The WWW provides automatic semantic concepts of the terms. Search engines such as Google, Yahoo, and Bing assist in retrieving search results through the Web. Google Distance is calculated by page count i.e. how often the term occurs on the Web. The page count score is calculated based on term information distance and Kolmogorov complexity. The search keywords with the same or similar meanings in a natural language sense tend to be closest in the units of Google distance i.e. higher similarity, while words with dissimilar or different meanings tend to fall farther apart i.e. have lower similarity. If the distance measure between two words is zero, or close to zero, it represents that they have the same meaning. When the distance is one or close to one, the words have different meanings.

WordNet (Miller, 1995) is a lexical resource for Natural Language Processing, in which nouns, verbs, adjectives and adverbs are grouped and organized into synonym sets (synset), each representing distinct lexical concepts. It provides the advantage of using both a dictionary and thesaurus. Each synset is linked to each other with the number of semantic relations such as hypernym, hyponym, holonym, and meronym. It is a rich database showing semantic relationship among words. The latest version of WordNet is 3.1, which was released in June 2011. Princeton University has published statistical data for WordNet 3.0 and it is not available for latest version WordNet 3.1. WordNet 3.0 statistics reported that the database contains 155,287 words organized in 117,659 synsets

for a total of 206,941 word-sense pairs (Wikipedia, 2012). The total size of WordNet 3.0 is about 12 megabytes, presented in the compressed form for ease of use. It is used for various purposes by researchers and specific domains such as computational linguistics, automatic text analysis and classification, and word sense disambiguation.

Though WordNet possesses all English words in its database, it does not contain certain domain-oriented words. For example, in the health domain, words such as 'Weaver', 'Waardenburg', and 'Wagner' are not contained in WordNet. This problem can be overcome by Google Distance approach, which can also calculate similarity for domain related words using the Web. But there are several instances where Google Distance may not find the similarity for words such "cuisine" and "cafeteria". The words that are used as near synonyms to each other, in some contexts, sometimes will not appear together in Google search pages. This problem can be overcome by using WordNet, when the Google Distance between two semantically equivalent words falls below a threshold value. Both approaches are complementary to each other. So, in our proposed approach we may use both WordNet and Google distance for finding the similarity between service signatures, when the data set under examination has a lot of synonyms or near-synonyms (e.g. in the travel domain). We also give a choice of using Google Distance only for fast signature matching for contexts when the data set has relatively fewer synonyms (e.g. security domain vs. travel domain).

The proposed hybrid approach does not rely on any ontology. As well, it can be used across all domains with low cost. In this thesis, we compare the performance of our approach with a pure ontology-independent approach, called FOIQOS, using standard delay, recall, and precision measures.

3.1.3 DIQOS Architecture

The traditional Web Services discovery model is constructed with its common three roles: service provider, service consumer and UDDI registry. As shown in Figure 3.1, our proposed architecture is in between the service consumer and UDDI registry. It automates the Web Services discovery and composition process when a consumer requests services from the registry. Our architecture accomplishes this task by using domain independent and QoS-enabled approaches for Dynamic Web Service discovery and automatic composition as shown in Figure 3.1. The broker is a software agent, which helps to automate the discovery and composition process on user demand. It performs functions such as input loading, preprocessing, discovering services and composing of services to accomplish the given workflow. It reads two kinds of inputs from the users: (1) One input which provides the workflow of the entire task along with the description for each service referred to as a service template, and (2) The user's actual input to execute and compose the workflow. For service discovery, our agent reads the requested service template and searches for the best candidate services using the DIQOS approach (Figure 3.1). Finally, the software agent reads the original input and follows the workflow to compose the discovered Web Services together and send the response to the requested user. The detailed methods of the service discovery are shown in Figure 3.2 that explains the different components and techniques we adopted to improve the discovery process.

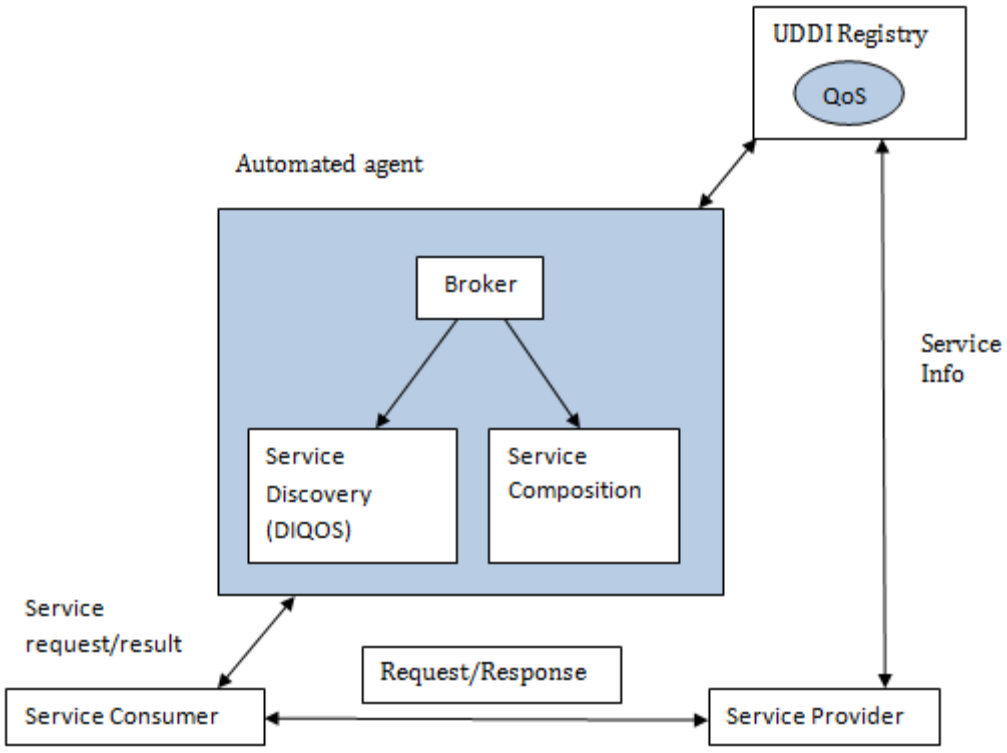

Figure 3.1 : DIQOS architecture

The proposed DIQOS approach for service discovery is broken down into three service components: specification similarity matching, signature similarity matching, and operational similarity matching.

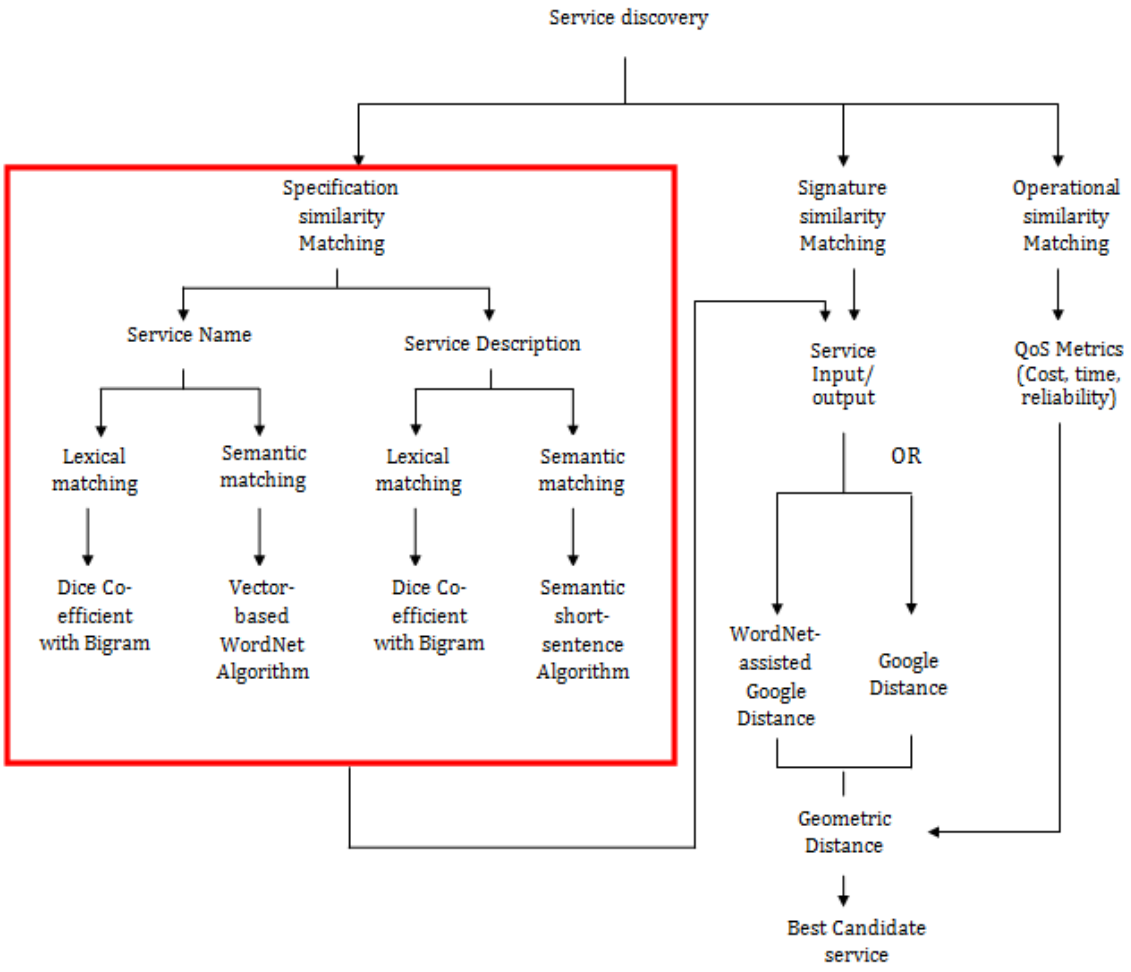

Figure 3.2 : DIQOS service discovery algorithm

The specifications of a Web Service such as name and description are considered to be very important. They contain brief explanations about the functionality of the service. WordNet is used to find the semantic similarity between two service specifications. However, words in short forms, abbreviations and misspellings are not meaningful and such words cannot be measured using WordNet. So, in order to overcome this drawback of WordNet, a lexical algorithm, Dice co-efficient with bigrams, is used in addition to using WordNet to calculate the similarity score. The similarity score

for two services' names and descriptions is calculated by taking the average of both lexical and semantic algorithms.

The signature of a Web Service refers to input and output parameters of the Web Service. The signature helps to identify the service that matches with its input/output concepts. We use NGD to extract the meaning of two concepts i.e. semantic description of a service's input and output parameters. The NGD value can be calculated using several search engines such as Bing, Google and Yahoo. We chose the Bing API in contrast to other search engine because the availability of service is free with no cost, whereas the Google API is free but the number of queries is limited to 100 per day. So, in this research, we use the Bing API, open source software to compute NGD for Web Service signatures. The primary reason for adopting Google Distance is to reduce the reliance on domain specific ontologies, to increase the system performance and to reduce the complexity of system design. The NGD is lightweight, less expensive and faster than ontology-dependent approaches. Google Distance uses the Web search space, which provides compensatory coverage of words that are not in the WordNet lexical database. However, two different words that are synonym to each other, may not always occur together in the Google pages. Consequently, Google Distance approach results in a much smaller similarity value. So, depending on the data set, we use either WordNet along with Google Distance or Google distance alone to bring out best similarity score.

Both specification and signature components together play a vital role in the discovery process. So, in this research, we propose an approach, which uses a specification matching algorithm in the first stage to filter out the services using a threshold value. In the next stage a signature matching algorithm is applied to find the

related services that match both semantically and functionally. To maintain standard in filtering irrelevant services, system threshold value 0.5 is used (Ding and Jutla, 2011). In service matching, the similarities score less than the system threshold value is considered as irrelevant and ignored.

Finally, an operational matching algorithm is used to find the best candidate service among the retrieved ones that satisfy user requested QoS metrics. We considered three major QoS metrics such as cost, time, and reliability. The main aim of the QoS metrics score is to find the best candidate service for the requested service template. A private UDDI is created to support QoS-enabled Web Service discovery. It helps service providers to publish their services along with their QoS parameters in the private UDDI. We have followed WSLA (Web Service Level Agreement) to obtain those values from the service providers and Trusted Third Parties (TTP). In this current work, the operational similarity algorithm has been adopted from Ding and Jutla's research work (2011), which efficiently chose the services when the operational similarity value is less than or equal to a system-subjective threshold value.

3.2 WEB SERVICE DISCOVERY

This following elaborates on the overview that was provided in the preceding section.

3.2.1 Optimized Data Preprocessing

The data preprocessing is essentially the first stage in knowledge discovery. It helps to remove unwanted and inflected words during filtering process. We use improved stemming algorithm in the data-preprocessing step. The usual stemming process removes

the inflection present in the words (Stroulia, 2005; Li, 2006; Zhang, 2006; Ding & Jutla, 2011). It is inappropriate for our matching process because the stemming process performs the operation on a word without knowing the context of the word (Wikipedia: Lemmatization, 2012). In this research work, the meaning of the words must be maintained even after the preprocessing step. So we use WordNet (Miller, 1995) to stem the service name and descriptions.

3.2.2 Specification Similarity Matching

Specification similarity includes matching of service names and service description published in the UDDI with the requested specifications. In earlier research (Cardoso, 2003, Ding and Jutla, 2011), the specification matching is performed lexically using edit distance and q-gram, but these methods do not help to identify the strings semantically i.e. when two different strings pose the same meaning. Li et al (2006) proposed a hybrid solution to compare two sentences using corpus-based and knowledge-based measures. They used semantic sentence similarity and word-order similarity to obtain overall similarity between two sentences. They tokenized the given sentences into a vector of strings and applied cosine sentence similarity measure implemented by WordNet (knowledge-based) and their proposed word similarity measures (corpus-based) to find semantic similarity. They also used word-order similarity, which possesses pseudo-syntactic behavior in the sentence similarity. This approach helps to compare two different words using WordNet and compute their similarity value using a corpus-based measure. However, Li's approach is not an efficient implementation for the sentence similarity, though they considered word-order similarity and semantic similarity. The word-order information is considered as pseudo-syntactic, which doesn't produce high-

level syntactic information from the sentence (Oliva, 2011). To overcome the problem, Oliva et al (2011) proposed SyMSS (syntax-based measure for short-text semantic similarity) technique to find the similarity between two sentences. They parse the sentence using Johansson and Nugues parser (2008) to get the syntactic information i.e. position of a word in the sentence, and then apply a lexical database to get the semantic information for comparing two sentences. They also calculate the similarity of two sentences efficiently by considering their lexical and semantic structures, but they do not consider other constraints such as spelling errors, improper sentence handling, and short-forms. To overcome the discussed issues, we implemented more appropriate methods in the similarity algorithms for specification matching.

Our proposed methods perform both lexical and semantic matching for Web Service name and description. For semantic matching of service names, we propose use of a Vector-based semantic algorithm using WordNet to measure the similarity score of two service names. For semantic matching of service descriptions, we use a semantic short-sentence matching algorithm to measure the similarity score of two service descriptions. However, semantic matching alone could not measure the similarity of two words because certain words such as “ttravel”, and “phno” are not present in WordNet. It could obtain a similarity score of 0 even if the words are referring to ‘travel’ and ‘phone number’. So, in order to find the similarity score for words with spelling errors and short-forms, a lexical matching algorithm Dice-coefficient with bigram is used along with semantic algorithms for specification matching.

3.2.2.1 Service Name Matching

Service names are short concepts to represent what the Web Service does. It usually contains one or two words about the service. For service name matching, we used both lexical and semantic matching. Lexical matching will overcome various issues like names represented in short-forms and spelling mistakes. Semantic matching can bring out the similarity between two different words with same meaning.

i) Lexical Matching

In lexical matching, we used Dice co-efficient with the bigram method to compare two service names.

Bigram: A bigram (Collins, 1996) is a sequence of two adjacent elements/letters in an input string of tokens. For example, when dividing the string "travel" into 2-gram, it gets divided into {tr,ra,av,ve,el}.

Dice Coefficient: Dice Coefficient similarity scoring is used to find the similarity between two strings. It is calculated by the ratio of the number of n-grams shared by two strings to the total number of n-grams in both strings (Kondrak et al, 2003). The similarity measures between the strings X and Y are calculated by the Formula (1) given below:

$$SIM_{Dice}(X, Y)=\frac{2\times |n-grams(X) \cap n-grams(Y)|}{|n-grams(X)| + |n-grams(Y)|} \quad \ldots\ldots \quad \text{Formula (1)}$$

where,

n-grams (X) = unique bigrams in string X

n-grams (Y) = unique bigrams in string Y

The Dice coefficient similarity value is in the range of zero to one.

The main reason to choose a string-matching algorithm as part of our proposal is to compare different kind of strings with spelling errors, short forms, and abbreviations. For example; strings such as ”erors” “mp3”,”ctrl” cannot be identified using a semantic lexical database such as WordNet. Using a string-matching algorithm can help to find similarities as they can find patterns shared by two strings in common regardless of their meanings.

Dice co-efficient with bigrams is a better indicator of similarity and correlation between two strings than other majorly used string similarity calculation technique such as Jaccard coefficient and cosine measures (Smadja, 1994). The service names may contain only a few characters in length. It is better to split the words into a smaller size to get similarity. For example, when a string “travel” is compared with a string “trevel”, the dice coefficient with bigram can calculate the similarity value as 0.6 but with tri-gram and four-gram approaches the values are 0.25 and 0 respectively as shown in Table 3.1, thereby producing lower similarity than our system’s threshold value. So, we used bigram to break the short Web Service specifications into fragments.

Table 3.1 : Comparison of various ngram

	Travel (S1)	**Trevel (S2)**	**Similarity value Between S1 and S2 using formula (1)**
Bigram	tr, ra, av, ve, el	tr, re, ev, ve, el	0.6
Trigram	tra, rav, ave, vel	tre, rev, eve, vel	0.25
Four gram	trav, rave, avel	trev, reve, evel	0

ii) Semantic Matching

For semantic matching, we use the vector-based WordNet algorithm. It uses cosine similarity using the Lin measure to compute the similarity between two service names.

Vector-based WordNet Algorithm

Lin measure: Lin (1998) describes a method to compute the semantic relatedness of word senses using the information content of the concepts in WordNet. The information content is derived from the sense-tagged corpus called SemCor[1]. The Lin measure is defined as "the ratio of the information content of the lowest common subsumer[2] (lcs) to the information content of each of the concepts" (Lin et al, 1998). The Lin measure similarity is calculated by the Formula (2) given below:

$$\text{Sim}_{\text{LIN}}(X, Y) = \frac{2 \times IC(lcs(X,Y))}{IC(X) + IC(Y)} \quad \ldots\ldots \text{ Formula (2)}$$

where, X and Y are the two service concepts of service names; IC (x) is the information content of x; lcs is the lowest common subsume of X and Y.

Cosine similarity: The semantic similarities between two strings are calculated based on the cosine similarity measures. The Formula (3) for the cosine similarity is given below (Li, 2006)

$$\text{SIM}_{\text{Cosine}}(X, Y) = \frac{A.B}{\|A\| \, \|B\|} \quad \ldots\ldots \text{ Formula (3)}$$

[1] SemCor corpus is a sense-tagged corpora created at Princeton University by the WordNet Project research team. It is a subset of the English Brown corpus, containing 360,000 words.
[2] The synset at the meeting point of two climbing paths of a word hierarchy is called subsume

where X, Y are the two service names; A, B are the tokens of the service names X and Y respectively.

To perform similarity matching, two service names should be tokenized and collected as 'bag of words'. Two vectors are created from those 'bag of words' using WordNet and the Lin measure as shown in Formula (2). The overall similarities between two service names are computed using the cosine similarity (Formula (3)).

For example: On performing semantic similarity between service names such as S1: "search hotel" and S2: "find cuisine", the algorithm tokenizes the strings and collect strings as bag of words V (search, hotel, find, cuisine). Two vectors (v1 and v2) must be created by using WordNet and the Lin measure between the bag of words (V) and service name tokens (S1, S2) as

V: (search, hotel, find, cuisine)

S1: (search, hotel) S2: (find, cuisine)

v1= {1, 1, 0.85, 0.7} v2= {0.85,0.7,1,1}

With the calculated two vectors, we can measure the semantic similarity of S1 and S2 by calculating the cosine product between v1 and v2 using Formula (3).

Overall, service name matching is calculated by taking the average of lexical and semantic methods using Formula (4).

$$SIM_{ServiceName}(X, Y) = \frac{SIM_{Dice}(X,Y) + SIM_{Cosine}(X,Y)}{2} \quad \ldots\ldots \text{Formula (4)}$$

To the best of our knowledge the application of the algorithm given below is new to the Web Services discovery. The following Figure 3.3 shows our proposed algorithm for performing lexical and semantic matching on service name:

Step 1: Textual description such as Service name of the candidate service is extracted from the private UDDI registry published by the service providers.
Step 2: Preprocesses the service names of candidate service and requested service using proposed improved stemming.
Step 3: Calculate bigram for the two service names and apply Dice co-efficient string similarity measure using formula (1) to obtain the lexical similarity value.
Step 4: Calculate semantic similarity using vector-based WordNet powered algorithm. Tokenize the service name into vector of strings and calculate the semantic similarity between two service names using cosine similarity using formula (3).
Step 5: Compute overall service name matching using the average of that of step (3) and step (4) using formula (4).

Figure 3.3 : Algorithm for lexical and semantic of service name

3.2.2.2 Service Description Matching

We assume that the service descriptions are meaningful sentences specifying the functionality of the service. But there can be an instance in which the developer might misspell the service or use short-forms and abbreviations in the specifications. To consider such instances, we included both lexical and semantic algorithms to identify the most appropriate service for the requested one. We followed the same lexical algorithm (Dice co-efficient with bigrams) used for service name matching. We proposed a method for exploiting structure as well as semantic of two service descriptions matching. The spirit of the semantic algorithm is taken from Oliva et al (2011) who applied it to

domains other than the Web Services discovery. For example, two sentences “This service receives city and returns shop “and “This service receives shop and returns city “. Both sentences have same set of words but syntactically the words are placed in different locations, which also alter the meaning of the sentence totally. In DIQOS we parse the sentence using POS algorithm (syntactic) and applied WordNet (semantic) along with the Lin measure to calculate similarity.

The following steps are used to compute service description matching:

Step 1: Textual description of the candidate Web Service is extracted from the private UDDI registry.

Step 2: Apply the Stanford parser (Marneffe & Manning, 2008) to parse the Web Service textual description syntactically. The syntactic structure is represented in the typed dependency form, which is provided by the parser to show the grammatical relationship between words in a sentence. Typed dependencies show how each word in a sentence is dependent on another word. It represents the relationship in the form of triplets such as name of the relation, governor, and dependent. For example, the Stanford parser can parse a sentence “This service receives city and returns shops” as shown in Figure 3.4. In Figure 3.4, the typed dependency between pairs of words in a sentence are, “the subject of ‘receives’ is ‘service’”, “subject of ‘returns’ is ‘service’”, “object of ‘receives’ is ‘city’” and “object of ‘returns’ is ‘shops’”. We consider only the “subject”, “verb” and “object” that are present in the sentence for comparison. We classify the typed dependencies in the Penn Treebank part-of-speech for “subject”, “verb” and ”object” as shown below in Table 3.2

Table 3.2: Grammatical relations

SENTENCE STRUCTURE	GRAMMATICAL RELATION (PENN TREEBANK)
Subject	csubj, csubjpass, nn, nsubj, nsubjpass,
Verb	dobj
Object	dobj, iobj, pobj

where

- obj is direct object
- iobj is indirect object
- pobj is object of a preposition
- csubj is clausal subject
- csubjpass is clausal passive subject
- nn is noun compound modifier
- nsubj is nominal subject
- nsubjpass is passive nominal subject
- num is numeric modifier
- number is element of compound number
- xsubj is controlling subject

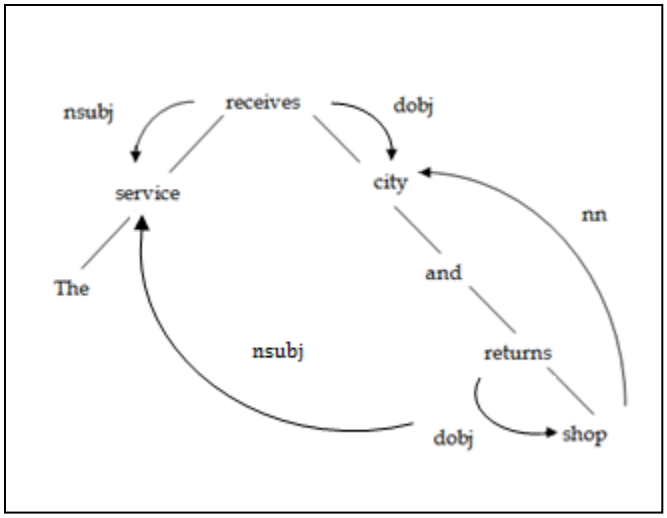

Figure 3.4 : Parsing and typed dependency of a sentence

Step 3: After extracting the syntactic structure from both sentences, the Lin similarity is applied for comparing the head and its dependents. In the semantic measure, we check for the head similarity value. If the head similarity value is close to 1 then finding the semantic similarity for dependency is continued; if the head similarity value is 0, then the dependency similarity computed is ignored. For example, consider two service descriptions, "This service receives shop" and "This service returns shop", that have different meanings. In these "receives, shop" and "returns, shop" are objects. On performing semantic similarity algorithm between heads ('receives', 'returns') similarity value is 0 and dependent ('shop', 'shop') similarity value is 1. As per the condition i.e. if the semantic similarity value between heads is 0 then the dependency would be ignored, which will achieve our desired computation. Comparison of two sentences "R: This service receives street and returns city" and "C: This service gets avenue and delivers location of town" is shown in the Table 3.3. Using Stanford parser, the requested service description (R) is parsed and retrieved grammatical relations as Nsubj (receives, service),

Nsubj (returns, service), Nn (city, street), dobj (receives, street), and dobj (return, city). Similarly, candidate service description (C) is also parsed and retrieved grammatical relations as Nsubj (receives, service), Nsubj (delivers, service), dobj (gets, avenue) and dobj (delivers, town). The algorithm groups the obtained relations from R and C into subject, verb and object. Each grammatical relation in each group of R is compared to each grammatical relation in each group in C. The subject of R {Nsubj (receives, service), Nsubj(returns, service), Nn (city, street)} is compared with C {Nsubj (gets, service), Nsubj(delivers, service)}. For example, Nsubj (receives, service) of R is compared with Nsubj (gets, service) and Nsubj (delivers, service) of C using the Lin similarity measure i.e. heads of subject (receives, gets) using the Lin similarity measure gives score 0.5 and the dependent of subject (service, service) using the Lin similarity measure gives score 1. The average of head and dependent gives 0.75 as similarity score for relations Nsubj. The overall similarity for the relation Nsubj (receives, service) is calculated by taking maximum similarity value obtain among the relations. Similarly, Nsubj (returns, service) of R is compared with Nsubj (gets, service) and Nsubj (delivers, service) of C using the Lin similarity measure (Formula (2)) and gets a similarity score as 0.7. And Nn (city, street), of R is compared with Nsubj (gets, service) and Nsubj (delivers, service) of C using the Lin similarity measure (Formula (2)) and gets a similarity score as 0.14. The total similarity score for subject group is calculated by taking the average of similarity scores of all relations. Similarly, total similarity score for verb and object group are calculated. The overall semantic similarity is applied on the dependencies obtained from the parser using Formula (5).

$$SIM_{Sem}(R, C) = \frac{SIM_{Subject}(R,C) + SIM_{Verb}(R,C) + SIM_{Object}(R,C)}{3} \quad \ldots\ldots \text{ Formula (5)}$$

Table 3.3 : Service description similarity

Grammatical relations	Requested Service(R)	Candidate Service(C)	Semantic Similarity between R and C (Lin measure)	
Subject	Nsubj (receives,service)	Nsubj (gets, service) Nsubj (delivers, service)	0.75 0	max (0.75,0)* = 0.75
	Nsubj (returns, service)	Nsubj (gets, service) Nsubj (delivers, service)	0 0.7	max (0, 0.7) = 0.7
	Nn (city, street)	Nsubj (gets, service) Nsubj (delivers, service)	0.14 0.12	max (0.14,0.12) = 0.14
	$SIM_{Subject}$			$= \frac{0.75+0.7+0.14}{3}$ = 0.53
Verb	Dobj (receives, street)	Dobj (gets, avenue) Dobj(delivers, town)	0.75 0.02	max (0.75,0.02) = 0.75
	Dobj (returns, city)	Dobj (gets, avenue) Dobj(delivers, town)	0 0.92	max (0,0.92) =0.92
	SIM_{Verb}			$=\frac{0.75+0.92}{2}$ = 0.83
Object	Dobj (receives, street)	Dobj (gets, avenue) Dobj(delivers, town)	0.75 0.02	max (0.75,0.02) = 0.75
	Dobj (returns, city)	Dobj (gets, avenue) Dobj (delivers, town	0 0.92	max (0,0.92) =0.92
	SIM_{Object}			$=\frac{0.75+0.92}{2}$ =0.83
SIM_{Sem}(R, C) **Using Formula (5)**			$\frac{0.53+0.83+0.83}{3}$ = 0.73	

* max(x,y) function provides maximum value among x and y

Step 4: The overall service description similarity is calculated by Formula (6) taking the average of lexical and semantic similarity calculations as above.

$$SIM_{ServiceDescription}(R, C) = \frac{SIM_{Dice}(R,C) + SIM_{Sem}(R,C)}{2} \quad \ldots\ldots \text{ Formula (6)}$$

The overall similarity for the specification is computed by using Cardoso et al (2003) syntactic function. The overall specification similarity is calculated as follows:

$$SIM_{total} = \frac{\omega 1\ SIM_{ServiceName}\ (R,C) + \omega 2\ SIM_{ServiceDescription}(R,C)}{\omega 1 + \omega 2} \quad \ldots\ldots \text{ Formula (7)}$$

where,

$\omega 1, and\ \omega 2$ are two weights given to service name and service description

$SIM_{ServiceName}$ (R, C) and $SIM_{ServiceDescription}$ (R, C) are the similarity value calculated by above formulas.

It uses two weights $\omega 1$ and $\omega 2$ that indicate the degree of confidence that a designer has for the service name and service description.

3.2.3 Signature Similarity Matching

The signature similarity matching of Web Service such as I/O parameters of requested services(R) are semantically matched with the candidate services(C) using the methods proposed in Ding & Jutla (2011). The semantic similarity on the signature of Web Service contains different meanings due to Web Service integration: 1) semantic

similarity between two inputs/outputs of R and C, which is a straightforward method of matching; 2) semantic similarity between outputs of R and inputs of C i.e. C may be the next candidate service in the requested workflow' 3) semantic similarity of the outputs of C and input of R and 4) semantic similarity between outputs of A and C and inputs of R, The different workflow requested by the users is given below in the Figure 3.5 (Ding & Jutla, 2011):

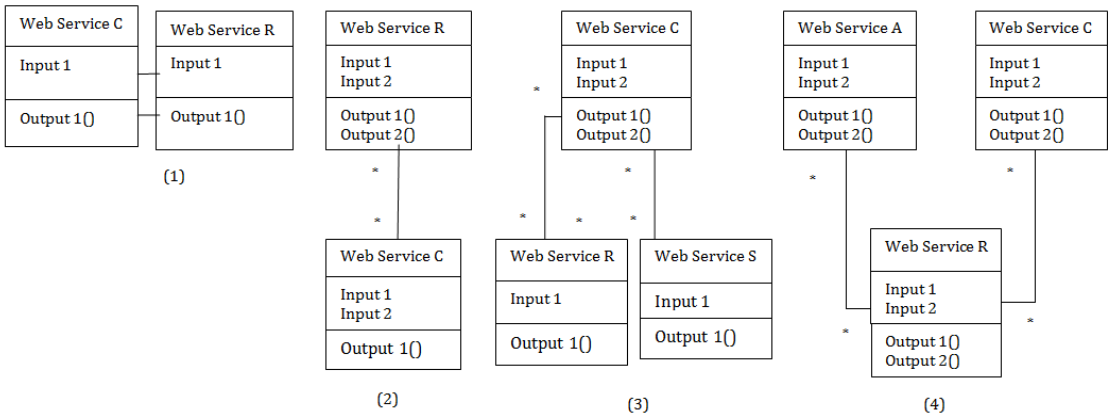

Figure 3.5 : Compose services

The semantic similarity for service signatures can be calculated using lexical databases such as WordNet, HowNet, and information retrieval approaches using Bing (Dong, 1999).

3.2.3.1 Semantic Similarity Matching using Google Distance and WordNet

Evangelista (2009) modified the Normalized Google distance to achieve the desired similarity as per above. That is the Google distance between the two same words must be 0 and two independent words must be 1. But the NGD possesses experimental value 0.7 for independent words, which is not accurate to satisfy the condition (Evangelista, 2009). So, in order to achieve the desired value of unity, Evangelista

recalibrated the NGD (Formula (8)) by dividing it by 0.7 as given below. In thesis, we use Formula (9) to calculate Google Distance.

$$\text{NGD (x, y)} = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{logM - \min\{\log f(x), \log f(y)\}} \quad \ldots\ldots \quad \text{Formula (8)}$$

Where,

f(x) = number of page hit for word x

f(y) = number of page hit for word y

f(x,y) = number of page hit for both x and y

$$\text{NGD (X, Y)} = \frac{NGD\ (x,y)}{0.7} \quad \ldots\ldots \quad \text{Formula (9)}$$

In this thesis, service input/output parameters are compared semantically using proposed WordNet-assisted Google Distance. Initially, the parameters are matched using Google Distance and similarity score is calculated. If the similarity score is less than the system threshold value then the WordNet is used to calculate the similarity score. If WordNet produces same or lower value than Google Distance similarity score then ignore the Web Service. If WordNet produces higher similarity score than threshold then continue the discovery process with the selected candidate service.

For example, consider two keywords such as “Hire” and “Rent”, were the distance between them is calculated using NGD. The Google search for the word “Hire” returns 93,200,000 hits and for the word “Rent” it returns 1,550,000,000 hits. By giving

both words together as “Hire Rent” the Google search return, 135,000,000 hits. The similarity score between two words is 0.83, which is calculated using Formula (9).

If the NGD value is less than the threshold level, then the WordNet must be invoked to calculate the similarity between services parameters. In the above example, similarity score is 0.83, which is higher than threshold value. So, WordNet invocation is not required and moved to next step of evaluation.

Consider another example, two service parameters “shop name” and “Boutique name”, with similarity score calculated by the Google Distance is 0.41, which is less than the threshold value even though ‘shop name’ and ‘boutique name’ are meaningfully equivalent. At this point WordNet method is invoked and similarities between words are calculated using Formula (3). The similarity score calculated is 0.7, which is greater than threshold value and consider as a relevant service. If the similarity score is less than the threshold value, service will be pruned as an irrelevant service.

3.2.4 Operational Similarity Matching

3.2.4.1 QoS Requirement for Web Service

The QOS requirement for Web Service refers to the quality aspects of a Web Service. QoS metrics helps the discovering process to identify reliable Web Services for users. Even though, specification and signature of requested and candidate service match, it is necessary to look for their QoS parameters match to determine the best composition for the requested workflow.

There are several QoS metrics available and in this thesis, we are using Response Time (T), Cost (C), and Reliability (R) to perform operation similarity matching.

Response Time (T) is the time interval from when the service is requested and delivered to the user. It includes the total processing time of the Web Service.

Cost (C) is the cost required by the customer to pay for the execution of the service. It is considered as an important parameter because certain Web Services cannot be accessed without paying for it and also these Web Services are costly. But, the users usually prefer to use those Web Services that are cheap.

Reliability (R) is the measurement of the services that correctly serve the users' requests. In this work, we have used this as the function of failure rate, using formula (10) below

Reliability(R) = 1- Failure rate …… Formula (10)

where,

Failure rate is the amount of time the service fails for total number of requests.

On performing the service composition, retrieving the Web Service with QoS values better than user requested values would bring the out best Web Service composition.

3.2.4.2 Computing Operational Similarity

Cardoso and Sheth (2003) computed the operational similarity of the requested service (R) and candidate service (C) using the geometric distance method. The

operational similarity range from 0 to 1 and it depends on R's and C's QoS parameters. They assumed that if the computed value was close to 1, it indicated that R and C were expected to match based on their close range of metric value and if the value was close to 0, it showed that R and C carry widely different performance values. The operational similarity on QoS metrics is computed as per the Formula (11) below:

T= $QoSdimD\ (R, C, Time)$

C=$QoSdimD\ (R, C, Cost)$

R=$QoSdimD\ (R, C, Reliability)$

$$\text{Operational}_{\text{similarity}}(R, C) = \sqrt[3]{T \times C \times R} \quad \text{…… Formula (11)}$$

where,

QoSdimD (R, C, dim) represents the distance of each QoS parameters presents in R and C, where dim is a QoS parameters. This function can be calculated by the Formula (12)

$$\text{QoSdimD}(R,C,\text{dim}) = \sqrt[3]{dcd_{min}(R, C, \text{dim}) * dcd_{avg}(R, C, dim) * dcd_{max}(R, C, dim)}$$

……. Formula (12)

where,

dcd_{min} (R,C,dim) represents the minimum values of QoS parameters in R and C, where dim is a QoS parameter. Similarly, dcd_{avg}, dcd_{max} were calculated where avg, max represents average and maximum values of a QoS parameters in R and C. This function

can be calculated by the formula (13)

$$dcd_{min}\,(R, C, dim) = 1 - \frac{|\min(C.\dim)| - |\min(R.\dim)|}{\min(R.\dim)} \quad \ldots\ldots \text{ Formula (13)}$$

This approach calculates operational similarity based on the assumption that the highest similarity value helps in selecting the best candidate service. But with QoS, the assumption is inaccurate because the users request with the longest processing time T1 and preferred processing time T2. Cardoso's algorithm will select the service, which satisfies the condition that time T to be in between T1 and T2. But if any service is available with T less than T2, it is avoided. According to Time parameter, the users always prefer shorter time, which fails in this approach.

To overcome the issues, Ding & Jutla (2011) used an improvement which assumed that QoS parameters in triple set (min, avg, max) of candidate service must be less than the acceptable value provided by the requested service. The operational similarity is calculated using the formula (14)

Substitute dim with QoS parameters such as time, cost and reliability to get the QoS similarity scores:

$$\{\min(C.\dim < R.\dim)\} \,\&\, \{avg(C.\dim < R.\dim)\} \,\&\, \{\max(C.\dim < R.\dim)\}$$

$$dcd_{min}\,(R,C,dim) = 1 - \frac{R.\dim - \min(C.\dim)}{R.dim}$$

$$dcd_{avg}\,(R,C,dim) = 1 - \frac{R.\dim - avg(C.\dim)}{R.dim}$$

$$dcd_{max}\,(R,C,dim) = 1 - \frac{R.\dim - \max(C.\dim)}{R.dim}$$

QoS dim D (R,C,dim)=$\sqrt[3]{dcd_{min}\,(R,C,\text{dim}) \times dcd_{avg}(R,C,dim) \times\ dcd_{max}\,(R,C,\text{dim})}$

…… Formula (14)

where,

dcd_{min}, dcd_{avg}, dcd_{max} are the distance used to calculate minimum, average and maximum of QoS values between R and C.

In this research work, we adopted Formula (14) from Ding and Jutla's (2011) work to calculate operational similarity value. If the similarity value is less than the threshold value then the service is selected as best candidate service.

3.2.4.3 Approaches to Obtain QoS Values

The general approach to obtain the QoS values is to use the values from the service provider or to get it from the trusted third parties. Here the system compares the two values of the QoS from the service provider and the trusted third party and checks it against various conditions within the system and decides to publish the service. Since it compares the values with that of the third party's value, it is considered to be reliable.

Initially a negotiated agreement, called the SLA (service level agreement), is defined between the service provider and the customer. It defines the level of service that a customer requires or can expect. It defines how the customer receives and not how the service provider delivers. It is technically interpreted as a set of SLO (Service level Objective) and the SLO contains the QoS measurements that are combined to produce the SLO achievement value.

WSLA (Keller & Ludwig, 2003) define the obligations of the parties involved. The service providers guarantee to provide services such as availability, response time and throughput for Web Services. WSLAs may include third parties such as management service providers that monitors and manages the measurement of metrics, supervision of guarantees or even the management of deviations of service guarantees.

This dissertation uses WSLA to describe the QoS parameter values obtained from the service provider and trusted third party. These values are used to calculate the reliability of the Web Service.

3.2.4.4 Approaches to Store QoS Value in System Design

- *Extension to UDDI Data Structure*

The existing UDDI can be extended to support the Web Service QoS metrics by using tmodels. It helps the service providers to publish their descriptions along with QoS, which assists in discovering the best candidate service for best composition. Figure 3.6 shows the tmodel representation to store the specification and QoS parameters of the Web Service.

```
<tModel tModelKey="uuid:1C620754-09E4-4930-AA19-709C62E52166">
  <name>uudi-org:qosInfo</name>
  <description xml:lang="en">Quality of Service Information</description>

  <overviewDoc>
    <description xml:lang="en"></description>
    <overviewURL>http://www.uddi.org/specification.html</overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
      keyName="uddi-org:types" keyValue="categorization"
      tModelKey="uddi:1D3FCD00-0DA6-4479-ADFE-B10950C74F62"/>
  </categoryBag>
</tModel>
```

Figure 3.6 : Sample snapshot of UDDI data structure

- *Extension to SOAP*

The existing SOAP message format can be extended to support the Web Service QoS metrics. It is used to inquire and publish the parameters of the Web Services. The functionality such as find business, find service, find tmodel, save business, save service, and save tmodel can be accomplished by the SOAP message. The sample functionality for finding the desired service is given below in Figure 3.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<envelop xmlns="http://schemas.xmlsoap.org/soap/envelop">
  <body>
    <serviceList generic="1.0" xmlns="urn:uddi-org:api"
        operator="rental.com/car/services/uddi" truncated="false">
      <serviceInfos>
        <serviceInfo
          serviceKey="F4D95E5D-F37A-4BDC-8E95-BFD45671A0F"
          businessKey="8CF8C1E3-4C39-43E1-8B6B-0137ED6BE63F">
          <name>car reservation</name>
          <qosInfo>
            <Time> 30 </Time>
            <Cost> 30 </Cost>
            <Reliability> 0.65 </Reliability>
          </qosInfo>
        </serviceInfo>
      </serviceInfos>
    </serviceList>
  </body>
</envelop>
```

Figure 3.7 : Sample snapshot of SOAP message

In this thesis, we used UDDI data structure approach to publish and retrieve the candidate Web Services from the registry.

CHAPTER 4 : IMPLEMENTATION

In this chapter, the implementation of the DIQOS service discovery and composition proposal is discussed. The following tasks are used to implement the proposed approach.

- Set up the development and testing environment.
- Create Private UDDI registry.
- Publish Web Services by storing semantic descriptions, WSDL and QoS parameters about the Web Service in the UDDI registry.
- Create a sample Workflow and Service templates instances as input.
- Inquire and discover the best candidate Web Services in the registry for each Service templates instance using Service discovery algorithm discussed in Chapter 3.
- Dynamically compose the discovered services together based on the workflow and output the result.

In this thesis, we compare DIQOS with FOIQOS based on the total execution time and relevant Web Services that are retrieved for the requested Workflow. The total composition execution time is calculated by varying the number of Web Services in the dataset from 10 to 100, adding 10 each time. F-Measure is calculated by using recall and precision (Wikipedia: F-Measure, 2012). We execute seven different queries; belonging to a business domain (Travel), and measured the level of performance in DIQOS and FOIQOS approaches. We also evaluate the best composition solution by calculating the

number of times a Web Service composition is achieved for each query. For our comparison of results, we create and publish 100 Web Services in a UDDI registry.

4.1 TOOLS USED

We use the following tools for implementing DIQOS approach.

- Java EE SDK 6 from Oracle Corporation, as a development language and run-time environment.
- JUDDI version 2.0.1 from Apache Software Foundation is Java implementation of UDDI version 2.0 specification and it is used as a private UDDI registry.
- MySQL 5.0.92 community Database server as the database server in support of the UDDI registry.
- Tomcat 6.0.16 application server from Apache Software foundation. It contains software components such as UDDI registry, Web Services and compose agent.
- UDDI4J version 2.0.5 from IBM and HP. It provides a Java API to interact with a UDDI registry.
- Eclipse SDK version 3.7.2 from the Eclipse Foundation. It is an open source Java development platform.
- Apache Axis2 version 1.6.1 from Apache Software Foundation. It is an implementation of the SOAP (Simple Object Access Protocol).
- WordNet 2.1 from Princeton University for semantic word matching (WordNet 3.1 API not available).
- Stanford Parser Version 2.0.1 is used to find grammatical structure of the sentence.

Our approach uses WordNet resources to perform preprocessing and service discovery process. Both approaches executes on an Intel Core 2 Duo CPU @ 3GHZ with 3GB memory. A screenshot of the Apache Tomcat application server manager is shown in the Figure 4.1. It shows that Web Services, UDDI registry and broker are deployed in the Apache Tomcat Application server.

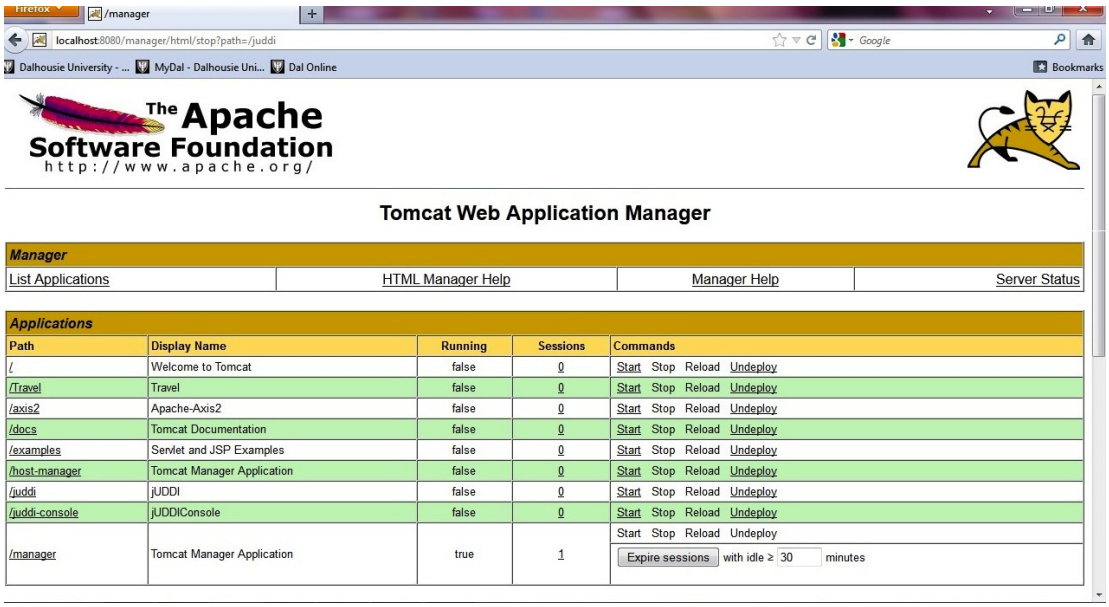

Figure 4.1 : List of applications deployed on the Apache Tomcat server

4.2 PERFORMANCE EVALUATION METRICS

This thesis evaluates the performance of the DIQOS and FOIQOS service discovery in the aspects of total execution time, recall, precision and F-Measure, in retrieving relevant Web Services for the requested service templates. We also evaluate the performance of DIQOS and FOIQOS composition in terms of the algorithms choosing the best candidate services and composing it for workflow. The best candidate

services are chosen based on the smaller QoS values calculated from the retrieved candidate Web Services.

The total execution time is defined as the time taken by the broker to discover the Web Services and to compose them automatically. We measure the performance of both approaches at each stage of their methodologies. The total execution time is calculated by summing three stages: Before Matching, Matching and Composition

The total time taken during 'before Matching Stage' includes sum of time for loading and parsing input, time for searching UDDI, time for loading WordNet and time for initializing the search engine API. In 'Web Service matching stage', the time taken is calculated by adding text preprocessing time, lexical and semantic matching of Web Service Specification, semantic matching of I/O parameters and QoS matching. Finally, 'Web Service composition' takes time in composing the retrieved services together.

We use standard evaluation measures, recall and precision to measure the performance of both DIQOS and FOIQOS approaches.

4.3 UDDI REGISTRY AND QOS INFORMATION

The current JUDDI standard supports QoS information of Web Service to store in the registry. The tModels are used to store QoS metrics advertised by the service providers. When the service providers publish the service, a tmodel is created and registered with the service deployment. They can update their QoS metrics by using their respective tmodel key.

In this thesis, we use Web Service explorer, a UDDI browser supported in Eclipse IDE to create a business and to publish the Web Services in the registry. To inquire the

Web services from the registry, we use UDDI4J, a Java implementation API in the service discovery model.

4.3.1 Publish Web Service in the UDDI Registry

To publish a Web Service in the UDDI registry, service providers require registering with the UDDI registry with UserId and Password. In order to publish the service entities, service providers need to create a business entity and save it in the registry. The following steps illustrate the procedure for publishing the business services in the registry.

Step 1: Get an authorization token by using user id and password registered at the UDDI registry

Step 2: Create a business entity to represent the service provider. Next, create service entities to represent the services that are to be published.

Step 3: Create a tModel to represent the QoS information for the service, save it in the UDDI registry. The tModels are referenced to the binding templates, which contain technical implementation about a service.

Step 4: Save the business entity in the UDDI registry, receive a business key and a list of service keys assigned by the UDDI registry

The semantic information and QoS of Web Service are published in the UDDI registry without modifying the existing UDDI standards. We use customized categories to store the information published by the service providers. The tModel is used to represent the customized category, which acts as a standard or a contract between service

providers and service consumers. In this thesis, we create a tModel "Travel" as a customized category containing eleven key name-descriptions for a Web Service. The key name such as input, output, QoS time min, QoS time avg, QoS time max, QoS cost min, QoS cost avg, QoS cost max, QoS reliability min, QoS reliability avg, and QoS reliability max. Thus, we store the functional parameter (input and output) and the non-functional parameter (QoS metrics) in the UDDI registry. Figure 4.2 shows the sample screenshot of functional and non-functional parameter storage in the registry.

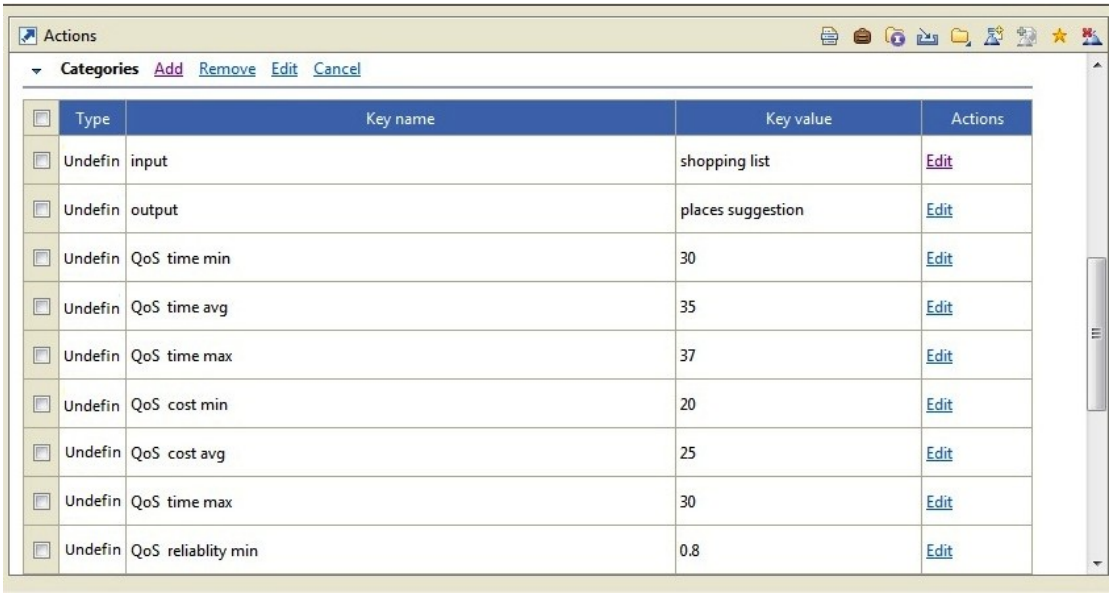

Figure 4.2 : Sample screenshot of functional and non-functional parameter storage in the registry

In this thesis, we create and publish, 100 Web Services in the private UDDI registry as shown in the Figure 4.3.

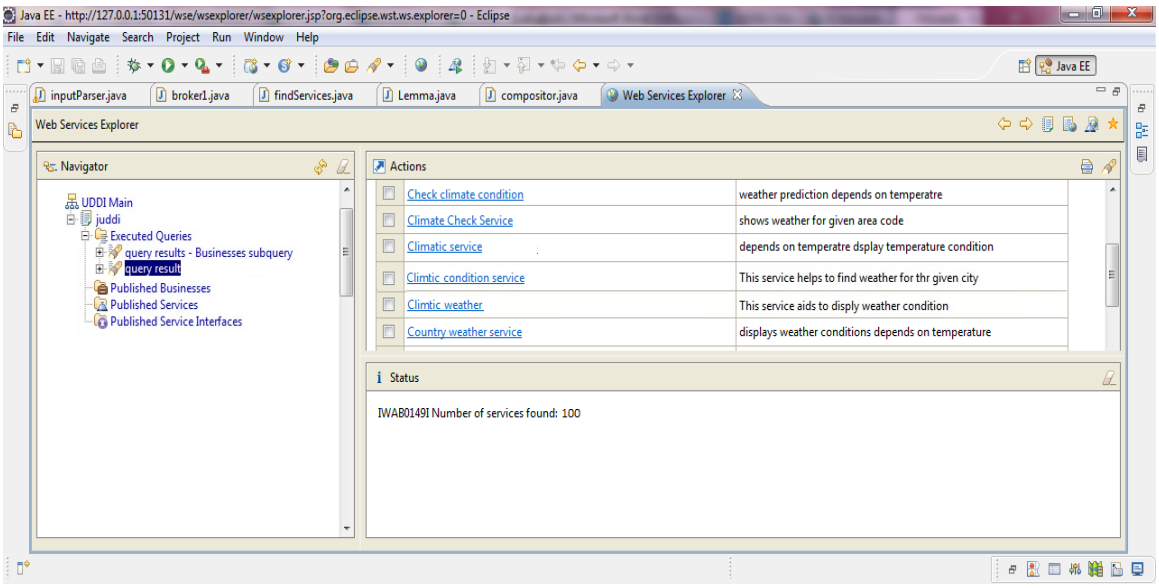

Figure 4.3 : 100 Web Services published to the private UDDI registry

4.4 IMPLEMENTATION OF SERVICE DISCOVERY METHODS

4.4.1 Find Web Service in Registry

In this thesis, we created a tModel "Travel", which act as an interface or category. We assume that the service providers on travel domain publish their services using this tModel. So, in our service discovery process, we search the UDDI registry with the tModel key to get the businesses that use "Travel" tModel to publish their services. Then, we get all the Web Services published by those businesses and perform data processing and service matching algorithm such as specification matching, signature matching and QoS matching to find the best candidate Web Service for the requested input. The pseudo code for finding the advertised Web Services from the UDDI registry is given below.

Construct a UDDI Proxy object
Set UDDI registry query URL

Construct a Find Quantifier
Set Find Quantifier to 'Case Sensitive Match'
Construct a Vector
Add the Quantifier into the Vector

Construct a tModel Bag
Add tModel Key into tModel Bag

Search UDDI registry by tModel Bag, Find Qualifier Vector and Maximum number of entries

Get Vector of Businesses

For (Each Business)
{
    Search UDDI registry by Business Key and Find Qualifier Vector
    Get Vector of Services list

    For (Each Service)
    {

    Get Service details by Service Key such as Service Name, Service Description, Service WSDL URL, Service Input/ Output parameter, and Service QoS parameters such as time, cost, and reliability.
    }

}

Store the advertised Web Services information to the list

```
<?xml version="1.0" encoding="UTF-8"?>
   <process id="1" type="sequence" >
  <process id="2" type="parallel" >
 <ST id = "1">
      <Name>find car dealers</Name>
      <Description>provde country code to fnd car dealers in your
location</Description>
     <Input>area-code</Input>
      <Output>dealers-name</Output>
      <QoS>
           <Time>20</Time>
           <Cost>35</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name>find insuance company</Name>
      <Description>get zipcode to view insurance providers</Description>
      <Input>zipcode</Input>
      <Output>insurance-company-name</Output>
      <QoS>
           <Time>30</Time>
           <Cost>45</Cost>
           <Reliability>0.6</Reliability>
      </QoS>
   </ST>
 </process>
<ST id = "3">
      <Name>rental car form</Name>
      <Description>displays both car delaers and insurance compay
available</Description>
      <Input>dealers,insurancecompany</Input>
      <Output>rental form</Output>
      <QoS>
           <Time>30</Time>
           <Cost>45</Cost>
           <Reliability>0.6</Reliability>
      </QoS>
   </ST>
</process>
```

Figure 4.4 : Sample input file

4.4.2 Load and Parse the Input

Our proposed automated agent receives Workflow and Web Service templates as input, which is represented in the XML format. The input file is parsed and the Web Service template instances are stored to the list. Figure 4.4 shows the sample Workflow input file. The broker loads the input file and parses it using the pseudo code shown below and gets the list of requested Web Service templates.

```
Construct a DocumentBuilderFactory object
Construct a DocumentBuilder object
Parse the input XML file and store it in document object

Get Document root element

Get a Node list of Service Templates

For (Each Node in the list)
 {

        Get Service Template element
          Parse the Service Template element values: id, Name, Description, Input, Output,
          Time, Cost, and Reliability.
         Add the Service Template instance to the list
}
```

4.4.3 Proposed Service Matching Algorithms

Our proposed service matching algorithm compares each service template requested by the users with the advertised Web Services in the UDDI registry. We obtain the advertised Web Services from the UDDI registry as discussed in the section 4.4.1 (Find Web Service in registry), which are stored as a list of advertised Web Service instance. We obtain the service template instances list from the requested workflow as discussed in 4.4.2 section (Load and parse the input). This section discusses about the proposed DIQOS approach, which is used to compare collection of advertised Web service with the requested service template instances.

We preprocess the service names using proposed stemming algorithm, before applying lexical and semantic matching algorithm. The stemming is performed by using WordNet and part-of-speech (POS). For lexical matching, Dice co-efficient with bi-gram is used and for semantic matching, the service names are tokenized into string arrays and applied WordNet on those bag-of-words. The semantic similarities between two words are calculated based on the proposed cosine similarity measure. The overall service name similarity is calculated by taking average of both lexical and semantic algorithms. The following pseudo code explains the implementation of improved stemming and semantic service name matching algorithm.

Improved Stemming

$$
\begin{aligned}
&\textit{Improved Stemming (Service name}_{\textit{Requested}}\textit{, Service name }_{\textit{Advertised}}\textit{)}\\
&\{\\
&\textit{Request\_Vector}_{\textit{Org}} \quad \textit{(Tokenized Service name}_{\textit{Requested}}\textit{)}\\
&\textit{Advertised\_Vector}_{\textit{Org}}\ \textit{(Tokenized Service name}_{\textit{Advertised}}\textit{)}\\
&\textit{Request\_Vector}_{\textit{Stemmed}} \leftarrow \textit{WordNet\_Stemming (Request\_Vector}_{\textit{Org}}\textit{)}\\
&\textit{Advertised\_Vector}_{\textit{Stemmed}} \leftarrow \textit{WordNet\_Stemming (Advertised\_Vector}_{\textit{Org}}\textit{)}\\
&\}
\end{aligned}
$$

Semantic service name matching Algorithm

$$
\begin{aligned}
&\textit{ServiceNameMatching}_{\textit{Semantic}}\ \textit{(Service name}_{\textit{Requested}}\textit{, Service name}_{\textit{Advertised}}\textit{)}\\
&\{\\
&\quad \textit{Improved Stemming (Service name}_{\textit{Requested}}\textit{, Service name }_{\textit{Advertised)}}\\
&\quad \textit{Semantic}_{\textit{Name}} \leftarrow \textit{CosineSimilaritySemantic (Request\_Vector}_{\textit{Stemmed}}\textit{,}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \textit{Advertised\_Vector}_{\textit{Stemmed}}\textit{)}\\
&\}
\end{aligned}
$$

We assume that the service description of requested and advertised Web Services is in proper sentence. So, we implement the semantic matching algorithm for description by matching two words in the descriptions holding same POS. We parse the two descriptions using Stanford parser API. After extracting the tagged dependency, group the tags into Subject, Verb and Object and remove all other tags. The similarity is calculated by computing semantic similarity between two same tags present in both descriptions using Lin similarity measure.

Semantic service description matching

$ServiceDescriptionMatching_{Semantic}$ $(Service\ Description_{Requested}, Service\ Description_{Advertised})$
{
$Service\ Description_{Requested}\_Tagged \leftarrow tagged\ dependency\ (Service\ Description_{Requested})$
$Service\ Description_{Advertised}\_Tagged \leftarrow tagged\ dependency\ (Service\ Description_{Advertised})$

For Each $(ReqTag \in Service\ Description_{Requested}\_Tagged)$
    For Each $(AdvTag \in Service\ Description_{Advertised}\_Tagged)$
        If (ReqTag & AdvTag are in same group)
        {
            $Semantic_{Description} \leftarrow Similarity + WordNet_{LinMeasure}\ (ReqTag, AdvTag)$
        }
}

The Input/output parameters of the Web Services are computed using WordNet-assisted Google Distance. The pseudo code for calculating the WordNet-assisted Google Distance is given below. For the Google Distance only version, the last ' if ' statement is removed.

```
Signature matching_Semantic (I/O_Requested, I/O_Advertised)
{
  SimilarityScore_NGD= GoogleDistance ( I/O_Requested, I/O_Advertised )
        If(SimilarityScoreNGD <threshold)
        {
        SimilarityScore_WordNet= WordNet_Semantic (I/O_Requested, I/O_Advertised)
        }
}
```

The operational similarities are computed using formula as discussed in chapter 3. The overall Web Service discovery algorithm is implemented in three levels: specification matching, signature matching and operational matching. We use threshold value 0.5 (0-1 scale) to filter out the unwanted services among the advertised services. For the better performance, we design the system to accept the service and move to next level of computation only when the similarity score is higher than the assigned system threshold value. The following is the pseudo code for overall service matching process.

Overall Service Matching Process

```
Construct a WordNet object, Stemmer object, bi-gram object, Parser object,
Google Distance object, QoS Distance object

Set Threshold
For Each (Requested Service Template)
{
     Apply improved Stemming to Requested Service Template's Service name
     For Each (Advertised Service)
     {
     Apply improved Stemming to Advertised Service's Service name

// Lexical Matching of Service names and description
Calculate dice-coefficient with bi-gram for Requested Service Template's and
Advertised's Service specification

//Semantic Matching of Service names and description
 Calculate semantic similarity using semantic name matching and semantic
description matching algorithm
```

```
        //Overall Service specification similarity
         Similarity_Specification = (Lexical _Name + Semantic_Description)/2

        Get the specification similarity score
        If (specification similarity score > threshold){
        /*Dataset with lots of synonyms*/
           Calculate WordNet-assisted Google Distance for Requested Service Template's
           and Advertised's Service I/O parameter

        /*Dataset with fewer synonyms*/
        Calculate Google Distance
                Get the signature similarity score
         If (Signature similarity score > threshold)
         {
              Calculate QoS Distance by using geometric distance
               If (QoS > 0 and QoS < best QoS value)
               {
                     Save the current Advertised Service instance as the best one.
               }
         }
        }
    }
}
```

4.5 IMPLEMENTATION OF DYNAMIC WEB SERVICE COMPOSITION

The dynamic Web Service composition takes Web Service request, original input, and discovered best candidate service object for each service templates in the requested Workflow. The Web Service request is an XML file contains service descriptions and workflow of the requested query. The sample queries are provided in the appendix A. In this thesis, we considered composition with two or three services. In the input file, workflow is represented using "process id" and "type". The process id can be either of type 'sequence' or 'parallel'. There are two types of workflow: 1) 'sequence' using two Web Services and 2) 'parallel' and 'sequence' using three Web Services. The service descriptions are referred to as service template (ST) and given unique number 'id' which helps to compose the services with workflow. The service template contains information

such as 'Name', 'Description', 'Input', 'Output' and 'QoS'. Each 'QoS' contain 'non-functional parameters such as Time', 'Cost', and 'Reliability'.

When composing a service, process type is used to determine whether a service is input to other services or whether it requires inputs from the other services for processing. If the process is of type 'sequence', the service requires an input from the other processed service. If the process is of type 'parallel', the service needs to use the original input at parallel and store the outputs.

The requested workflow service templates are stored as document object as discussed in 4.4.2 section (Load and parse the input). We implement the Web Service composition algorithm in the broker, to run the workflow with the input and execute the discovered Web Services dynamically and composing them together to produce output. The algorithm checks for each node object and their process type before processing their respective discovered Web Service with inputs. The pseudo code for dynamic Web Service composition is implemented as follows

```
If (Node type == ELEMENT_NODE) {
  Get all attribute of the Node
For Each Attributes {
  If (Node name == "ST" and Attribute name == "id")
{
      Get the discovered best candidate Web Service for this ST
       Invoke the Web Service with the input parameters
   }
  Else if (Node name =="process" and Attribute name =="type") {
      Get the process type whether it is sequence or parallel
  }
  For Each child node {
      If (process type == "sequence' and child node name == "ST") {
      Store the output, which is used as input for next service template
      Recursive call the function to process next child node
       }
```

```
        Else if (process type == "parallel" and child node name == "ST") {
        Recursive call the function to process next child node
        Add outputs from the previous child nodes
        }
        Else {
    Recursive call the function to process next child node
     }
  }
}}
```

4.5.1 Dynamic Invocation of Web Service

Web Service objects are discovered during the runtime using our service discovery methodology. The dynamic invocation of that Web Services can be achieved by WSDL file. The discovered service objects contain WSDL URL, which helps to get to the location of WSDL file for each Web Service object. Our designed algorithm then parses the WSDL file and processing it by passing input parameters. A WSDL document is a XML format file, used to describe a Web Service. It is represented using several elements classified into three major categories as type, operation, and binding. The WSDL file can be created either manually or the automated tools like JAVA2WSDL.

In this thesis, we create WSDLs using automated tool and deployed it in the Apache Tomcat using Apache AXIS framework. Apache Axis is implemented in SOAP, which provides extensive support to processing of WSDL document. The following pseudo code explains the implementation of dynamic invocation Web Service process.

// Parsing and obtaining elements from WSDL
Construct a WSDL parser object
Parse the WSDL File
Obtain all elements entries from WSDL file
// Obtain required elements from WSDL file
Get the service entry for the input service name
Get the service object
Get all the ports from the service object
Get the end point reference value
// Service call
Construct a client service object
Create a new service call
Set the end point reference value and operation name for the created call
Get the binding entry for this service and this port
Get the parameters contained in the binding entry
Assign input values to the input parameters
Invoke the service and assign the outputs to the output parameters

CHAPTER 5 : EVALUATION

We conducted experiments on ontology-independent (FOIQOS) and domain-independent (DIQOS) approaches to evaluate their performance in the aspects of overhead delays, recall, precision and F-Measure.

5.1 EXPERIMENTS

We created a test data set consisting of 100 simple web services that lend themselves to composition for tasks in the 'Travel' domain. The data set is constructed in the manner of having distinct inputs, outputs, semantic descriptions, WSDL descriptions and QoS values. The published Web Service descriptions are provided in Appendix B. Table 5.1 and Table 5.2 show the seven representative queries or transactions, such as flight booking, hotel search, email notification, weather prediction, rental car, shopping, and login verification and their respective relevant services in the data collection. The complete descriptions of the queries are provided in Appendix A.

To test the performance of the algorithm, we queries include different types of service descriptions and I/O parameters. For semantic matching of service specification, our dataset includes service name and description with different words possessing the same meaning. For signature matching of service parameters, semantic I/O parameters, where some synonym words would (or would not) occur together in the Google pages, are also tested. Our experiments measure efficiency of the selection process by using standard evaluation measures: delay, recall and precision.

Table 5.1 : Query type and relevant candidate services

Q.no	Query	Total no. of relevant service
1	Rental car	12
2	Hotel search	25
3	E-mail notification	18
4	Weather prediction	14
5	Flight booking	11
6	Shopping	12
7	Login verification	8

Table 5.2 : Sample Web Service description for seven queries

Q.No	Service Name	Service Input's	Service Output's	QoS Parameters		
				Cost	Time	Reliability
1	Find dealer	Area code	Dealer name	35	20	0.5
	Find insurance company	ZIP	Insurance name	45	30	0.6
	Rental form	Dealer, insurance company	Rental form	45	30	0.6
2	Search Hotel	Current location	Restaurant list	35	30	0.5
	Cuisine address	Restaurant name	Hotel address	70	60	0.6
3	Travel package	Travel deal	User name	45	40	0.5
	Email notification	Email ID	Offer notification	50	55	0.4
4	Weather forecast	Search city	Temperature	66	55	0.2
	Climate condition	Weather	Prediction	60	30	0.6
5	Search flight	Travel request	Airlines list	35	20	0.5
	Compare flights	Flights names	Best deal	45	30	0.5
6	Shopping	Shopping plan	Stores list	50	45	04
	Location finder	Mall name	Shop spot	55	35	0.6
7	Password check	User Authentication ID	Verification status	35	40	0.4
	Authentication confirmation	Login message	Login confirmation	45	30	0.6

Further, experiments produce a key outcome measure: best QoS-enabled web services composition. We compared FOIQOS and DIQOS approaches by calculating standard evaluation measures of overhead delay, recall and precision.

Each approach is split into different stages: Before Matching, Google Initialization, Isolated Google Distance, Matching Stage, and Composition. The overhead delay examination is performed at each stage of the algorithm. For each approach, we varied the number of Web Services instances in the UDDI, from 10 to 100 by incrementing 10 each time. The recall and precision measures are calculated by varying the queries belonging to the Travel domain. We used standard evaluation criteria to measure the efficiency of our approach against FOIQOS and the computation of recall and precision are given as follows.

Recall denotes the ability of the system to retrieve all the relevant items (Wikipedia: Recall, 2012). It is calculated by

$$\text{Recall} = \frac{total\ number\ of\ relevant\ items\ retrieved}{total\ number\ of\ relevant\ items\ in\ collection}$$

Precision is the measurement of the ability of the system to retrieve only relevant items (Wikipedia: Precision, 2012). It is calculated by

$$\text{Precision} = \frac{total\ number\ of\ relevant\ items\ retrieved}{total\ number\ of\ items\ retrieved}$$

F-Measure, which is the weighted harmonic mean of recall and precision, calculates the trade-off between precision and recall (Manning et al, 2008). It is computed by:

$$\text{F-Measure} = \frac{2*Recall*Precision}{(Recall+Precision)}$$

A typical raw result of an experiment is shown in the Figure 5.1. We provided the detailed execution time for each stage in the methodology: Before Matching (loading and preprocessing the inputs, UDDI search, loading WordNet, initializing the search engine API), Service Matching, and Composition. We also presented the best matching service objects for each Web Services instance in the requested Workflow. Each Service object is represented with the following parameters: Service name, Service description, Input/output parameters, and QoS metrics. The dynamic composition with the discovered Web Services is also summarized in the result.

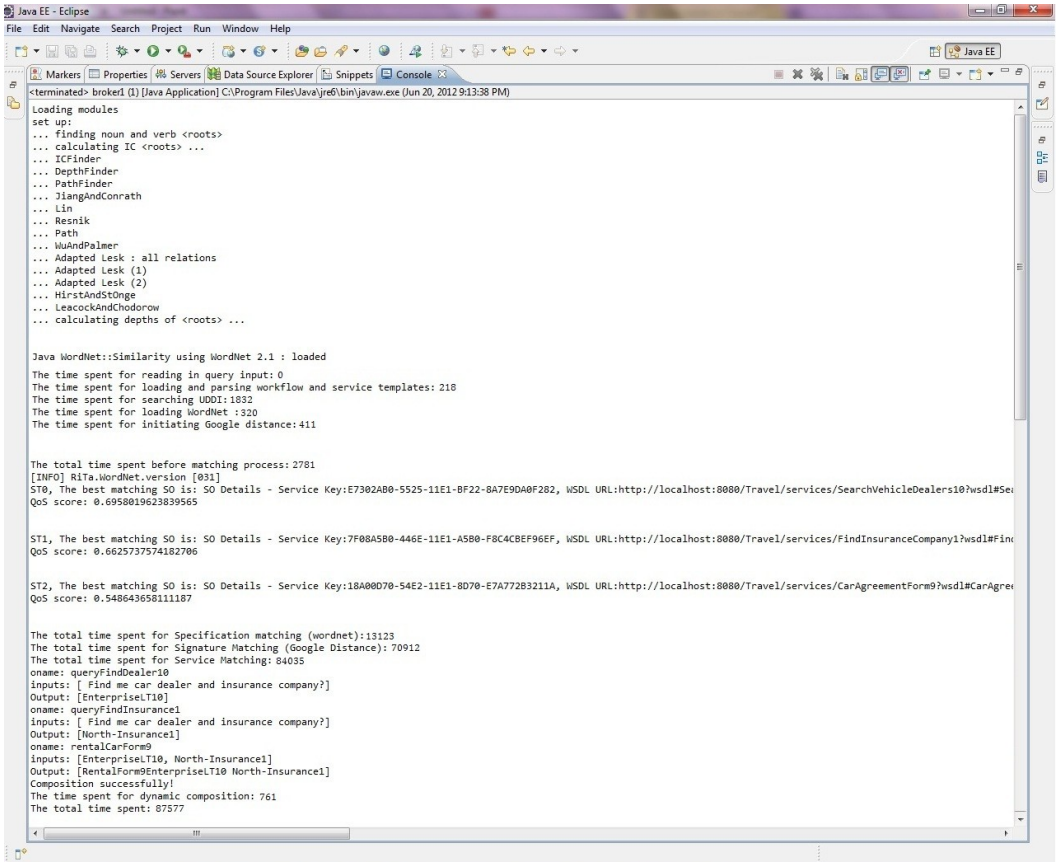

Figure 5.1: Sample output of DIQOS

5.2 RESULTS: AN EXAMINATION OF THE TRADEOFFS

We examine the qualitative and quantitative tradeoffs of the web services discovery methods next.

5.2.1 Overhead Delay Examination

Figure 5.2 shows the total execution time taken for both the approaches by increasing the number of candidate Web Services in UDDI by ten for each execution. It shows that FOIQOS performs much better in execution time than our approach because it does not rely on the semantic resources for service selection. In our approach, we used WordNet in preprocessing and service selection stages to retrieve most of the relevant service that matches partially/fully meet the requirements of the requested user specification. The total execution time of DIQOS takes approximately 15% more time than FOIQOS in execution.

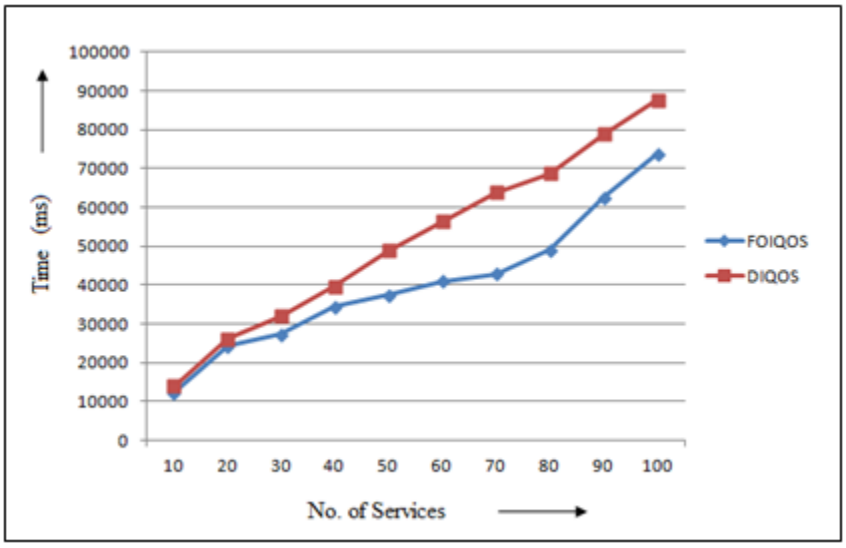

Figure 5.2 : Comparison of total execution time in FOIQOS and DIQOS

It is notable that when the number of Web Service instances is small, both approaches take almost the same execution time to produce the output. But when the numbers of Web Service instances are increased in UDDI, there are larger variations in the total execution time. The difference then is about 15% more in DIQOS as compared to FOIQOS. The reason behind the difference in total execution time is mainly due to the inclusion of the semantic lexical database WordNet in the before matching and matching stages. The "before-matching" stage includes search engine API initialization**,** WordNet initialization (DIQOS), input loading, and UDDI search activities. As previously discussed, both approaches rely on Google distance to calculate signature similarities. They need to invoke the Bing server to get the similarity scores between advertised and requested services signatures. Due to the server availability issues and network traffic, the similarity scores are not constant and some variations in execution time are expected. In the matching stage within FOIQOS and DIQOS, the execution time depends on preprocessing the input and advertised services, specification matching, signature matching, and QoS matching.

Table 5.3 : Execution time on each phase (FOIQOS)

No. Web Services	Before Matching (incl . Loading Input, UDDI search & Search Engine Init) (ms)	Isolation Google Distance (GD) (ms)	Matching stage (incl. GD) (ms)	Composition (ms)	Total execution time (ms)
10	1171	9439	10402	764	12337
20	1297	20480	22155	767	24219
30	1392	22089	25178	763	27333
40	1512	28781	32065	763	34340
50	1653	31219	35081	765	37499
60	1771	31869	38362	764	40897
70	2022	36416	40053	766	42841
80	2137	37475	46136	764	49037
90	2276	52780	59484	765	62525
100	2461	68341	70574	764	73799

Table 5.4 : Execution time on each phase (DIQOS)

No. of Web Services	Before Matching (incl. Search Engine Init, WordNet Init, Loading Input &UDDI search) (ms)	Isolated Google Distance (GD) (ms)	Word Net Matching &Ngram &Operational matching (WMNGOP) (ms)	Matching stage (incl. GD & WMNGOP) (ms)	Composition (ms)	Total Execution time (ms)
10	1513	10523	1251	11774	765	14052
20	1601	21233	2523	23756	763	26120
30	1727	25563	3892	29455	767	31949
40	1836	30145	5023	37168	761	39765
50	1961	33763	6311	40074	768	42803
60	2095	45921	7501	53422	762	56279
70	2345	51611	9100	60711	765	63821
80	2474	55034	10412	65446	769	68689
90	2594	63712	11723	75435	763	78792
100	2781	70912	13123	84035	761	87577

Tables 5.3 and Table 5.4 illustrate the detailed execution times for each stage in the discovery and composition process of FOIQOS and DIQOS respectively. The entire process is divided into three stages: before matching, matching stage, and composition. Here, the number of candidate services for discovery is increased from 10 to 100 (increasing 10 Web Service instances on each iteration).

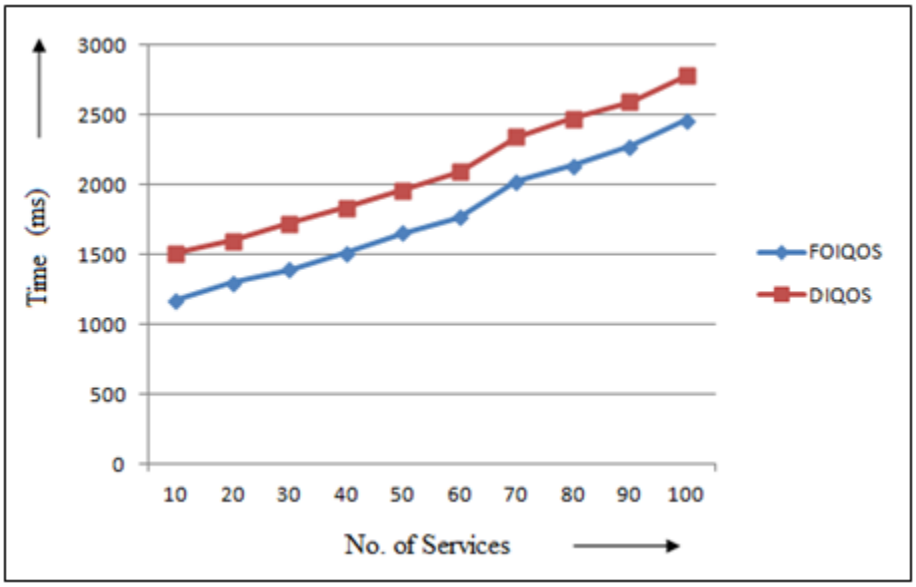

Figure 5.3 : Comparison of before matching stage in FOIQOS and DIQOS

In the before matching stage, in FOIQOS, the total execution depends on the loading and parsing of the inputs, UDDI search and search engine initialization. Whereas DIQOS uses WordNet for improved service matching and the total execution time depends on the loading and parsing of the inputs, UDDI search, WordNet loading and search engine initialization. The loading and parsing of input takes about 200-220 milliseconds, WordNet loading takes 310-340 milliseconds, and search engine initialization takes about 410-440 milliseconds. The time taken for UDDI search relies on number of Web Services instances in UDDI. The time consumption for UDDI search

(10-100 Web Services) is in the range of 500-1900 milliseconds. In the before matching stage, the major time is spent for searching UDDI when the number of services increase significantly. Even search engine initialization takes a while in this stage; it almost remains the same after each execution. Figure 5.3 illustrates the comparison of execution time before matching stage in FOIQOS and DIQOS. It shows that the DIQOS consumes more time in before matching stage than FOIQOS. The reason for higher execution time in this stage in DIQOS is due to loading of WordNet for service matching as the rest of the "before matching" stage is identical in both FOIQOS and DIQOS (Figure 5.3).

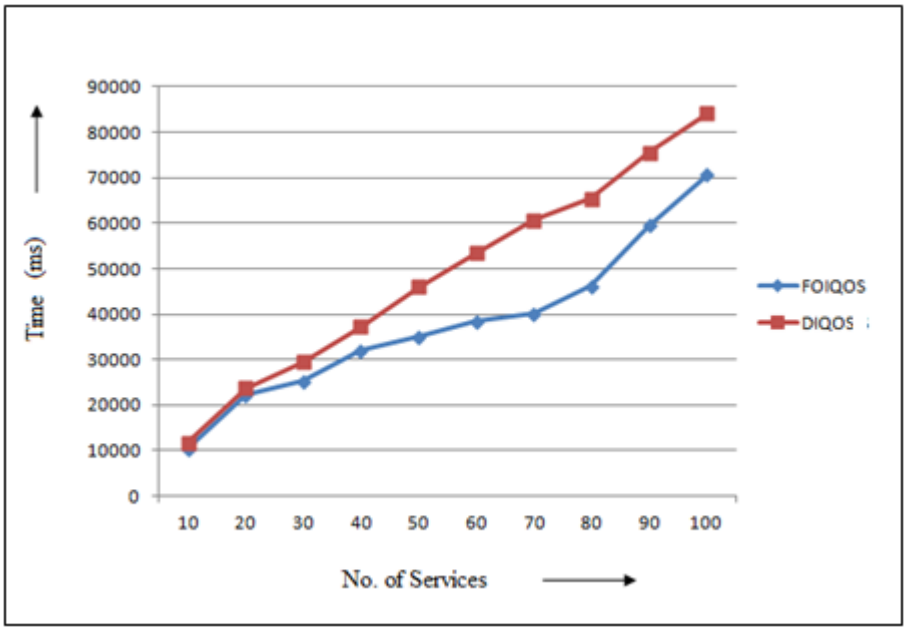

Figure 5.4 : Comparison of matching stage in FOIQOS and DIQOS

In the matching stage of DIQOS, the execution time depends on preprocessing the input and advertised services, specification matching, signature matching, and QoS matching. Instead of using a regular stemming algorithm, we used our proposed improved stemming algorithm to perform the effective selection mechanisms. We applied

parts-of speech concept using WordNet prior to finding the root word. It helps to maintain the meaning of the word lexically.

After preprocessing service input, specification matching is performed by a lexical algorithm (Dice co-efficient with bigram) and semantic algorithm (using WordNet). The signature matching is performed by using the Google Distance algorithm in finding semantic similarities between input and output parameters of the Web Services. Finally, the QoS similarity is calculated to find the best candidate services for the requested service. In both approaches, the matching stage consumes more time compared to the "before matching" and "composition" stages in the algorithm.

Figure 5.4 illustrates the comparison of execution time in the matching stages in FOIQOS and DIQOS. It shows how much more time DIQOS consumes than FOIQOS in both specification and signature matching stages. In specification level matching, as DIQOS accesses the WordNet database for semantic matching, it consumes more time than FOIQOS. FOIQOS performs service matching by using a string matching algorithm and retrieves the services which match lexically. But DIQOS performs service matching by using both a string matching algorithm and semantic matching algorithm (WordNet) and retrieves a larger number of relevant services than FOIQOS. Consequently, due to processing a larger numbers of retrieved relevant services from the specification matching stage, DIQOS takes more time than FOIQOS to accomplish signature level matching using Google Distance. In signature level matching, inclusion of Google Distance also consumes more time. Thus, tables 5.2 and 5.3 show that while performing isolated Google Distance, DIQOS takes (5-15 %) more time than FOIQOS. It also

illustrates that Google Distance calculation consumes the maximum time in the matching stages in both FOIQOS and DIQOS.

The final stage of our approach is the dynamic Web Service composition for the requested workflow input. After discovering the best candidate services for the requested input, the composition algorithm will parse the WSDL for each discovered Web Service instance and invoke Web Services based on the input workflow. Finally, the algorithm composes the result and presents it to the requesting user as output.

5.2.2 Recall and Precision

The evaluation is carried out in specification and signature level matching process to find out the performance of the algorithms. The first stage is specification level filtering (i.e. matching name and description of services), which is considered to be important in the service discovery process. Although two services may provide the same textual description, they do not need to rely on the same number of input/output parameters necessarily. So, signature level filtering (i.e. matching input and output parameters) is used as a second stage to filter irrelevant services missed in the first stage. In this thesis, we have evaluated the recall and precision of the signature matching using Google Distance only and WordNet-assisted Google Distance. We considered different queries in the travel domain to evaluate the performance of our algorithms in terms of recall and precision.

5.2.2.1 Specification matching recall and precision

Table 5.5 : Comparison of various threshold values at specification matching

Q. No	Threshold 0.4			Threshold 0.5			Threshold 0.6		
	Recall %	Precision %	F-Measure %	Recall %	Precision %	F-Measure %	Recall %	Precision %	F-Measure %
1	92	68	78	92	79	85	75	82	78
2	94	65	77	92	72	81	83	76	79
3	92	71	80	84	80	82	80	83	81
4	93	66	77	93	74	83	91	76	83
5	100	55	71	100	64	78	96	68	80
6	92	62	74	92	75	83	89	79	84
7	89	66	76	89	73	85	85	78	81

Table 5.5 shows recall, precision and F-Measure calculated for seven queries with the threshold values 0.4, 0.5 and 0.6. The overall F-Measure for threshold 0.4, 0.5 and 0.6 is calculated by taking the average of the respective F-Measures scores of individual experiments. The average F-Measure for threshold values of 0.4, 0.5, and 0.6 are 76, 82 and 81, respectively. The matching process with the threshold value 0.5 produces a higher F-Measure value when compared to those of threshold values 0.4 and 0.6. The increase in the F-Measure indicates a better discerning ability (Budanitsky et al, 2001, Pinnis, 2012). So, in the specification matching we used 0.5 as a standard measure to prune irrelevant services and to retrieve more relevant services.

Table 5.6 : Specification level recall and precision of DIQOS & FOIQOS

Q.No	DIQOS (Google Only Signature Matching)				FOIQOS			
	Service retrieved	Recall %	Precision%	F-Measure %	Service retrieved	Recall %	Precision%	F-Measure %
1	14	92	79	85	6	50	100	67
2	32	92	72	81	20	68	85	76
3	20	84	80	82	15	68	87	76
4	19	93	74	83	12	67	83	74
5	11	100	64	78	4	37	100	54
6	16	92	75	83	11	77	91	83
7	11	89	73	85	7	88	100	92

Table 5.6 shows the recall and precision values calculated at specification level matching by DIQOS and FOIQOS approaches. DIQOS outperforms FOIQOS in producing high recall for all queries but precision is low. DIQOS retrieves possible number of relevant services by performing both lexical and semantic matching algorithms, which lead to increase of recall value. On the other hand, it also retrieves some irrelevant services, which reduce the precision value. Rather, FOIQOS approach follows only lexical matching algorithm in the specification matching stage, which causes poor recall but high precision. DIQOS (Google only) shows 8% increase in F-Measure compared to FOIQOS. The sample result for a “Rental car” query at specification level filtering stage of both approaches is displayed in the Table 5.7.

Table 5.7 : Sample output at specification level

Requested Work Flow	Relevant Service	Retrieved services (DIQOS)	Retrieved services (FOIQOS)
Search Vehicle dealers Search insurance company Car agreement form	Rental car form Find insurance company Explore company insurance Car agreement Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Contrat form	Rental car form Find insurance company Explore company insurance Car agreement Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Search cheap flight Compare flight Find Flight deal	Serch vehicle dealer Find insurance company Car agreement Vehicle dealer Search vehicle daler Search insurance comp
Precision		79	100
Recall		92	50

For the requested Vehicle booking workflow, relevant services in the test collection, such as “Explore insurance Company”, “Hire vehicle form”, “Rental car form”, “Contrat form” and “Car dealer”, are not retrieved using FOIQOS but DIQOS

matches these possible services. But DIQOS missed “Contrat form” Web Service even though it was considered as a relevant service in the test collection. The requested Web Service name was ‘Car agreement form’ but the available service name was ‘Contrat form’. When we applied our proposed service name matching algorithms on this Web Service specification, it failed to find the match because the word ‘Contract’ was misspelled as ‘ Contrat’; the semantic algorithm using WordNet was unable to find the match because ‘Contrat’ was meaningless; and the lexical algorithm failed too because of minimum bigrams in common. DIQOS also retrieved some other Web services such as ‘Search cheap flight’, ‘Compare flight’, and ‘Find flight deal’, which are considered to be more relevant in their specification. But comparatively, our approach retrieves partial/fully satisfied user requirement services, but FOIQOS retrieves only some of the services that match strictly with the requested service. In the vehicle booking workflow, FOIQOS produces 20% higher precision than DIQOS but lower recall. The FOIQOS approach retrieved only 50% of the relevant services from the total relevant collection. But DIQOS matching algorithm shows better recall, producing almost all the relevant services from the test collection. Figures 5.5 and 5.6 illustrates that our proposed approach shows better recall performance than FOIQOS at specification level matching.

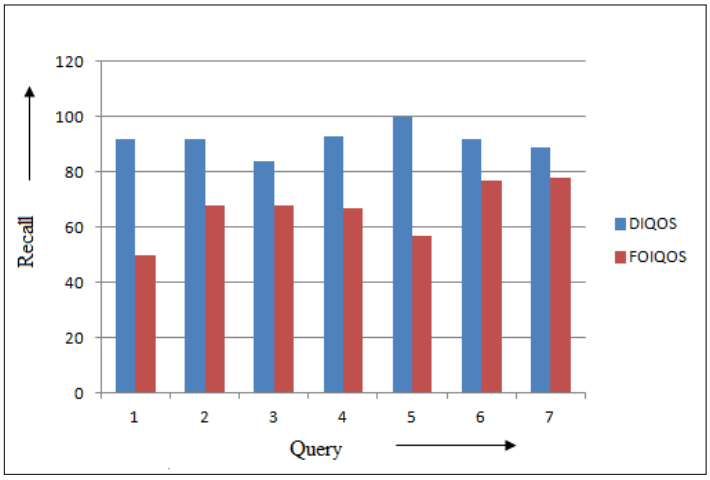

Figure 5.5 : Recall at specification matching

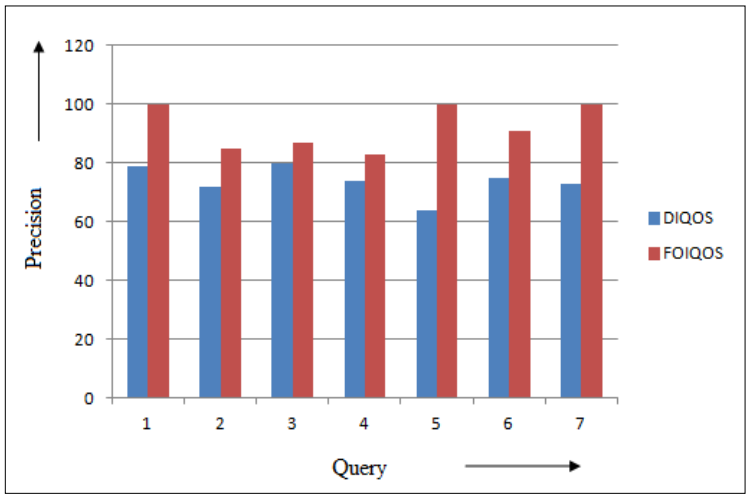

Figure 5.6 : Precision at specification matching

It is clear from our results that recall and precision measures are query-dependent in the web services discovery task. In all cases, FOIQOS demonstrates higher precision than DIQOS. But for the flight booking transaction, there is a wider gap (36%) in precision than for other queries. The difference in precision for the email notification transaction of two approaches is negligible. The results show that the WordNet addition in DIQOS is responsible for a better coverage of relevant services.

FOIQOS is strict in that it performs its selection process on the specification such as names and descriptions using “string matching” algorithms (edit distance and q-gram). These algorithms help to find matches lexically but fail to match service descriptions using semantic concepts.

For example, consider a user request for a service “customer login verification”. When FOIQOS processes the request, it finds services such as “customer id verification”, “Cust login verify”, “login verification”, but it does not find services such as “user Id confirmation”, “passenger authentication” and “authenticate user Login”. Because, FOIQOS fails to find semantic matches, even though “authentication”, “confirmation”, “verification” are synonyms. This limits the selection of Web Services available for the consumer with possibly superior QoS and, thus, it produces lower recall. In the above example, though FOIQOS fail to match the services semantically, it does not retrieve any irrelevant data. Consequently, it demonstrates higher precision.

To increase the recall, our approach uses WordNet-assisted service selection strategy along with string matching algorithm. When DIQOS processes a request, it finds services that match both lexically and semantically.

Since WordNet contains networks of meaningful words, it cannot find matches for words such as “ID”, “cust”, “pwd”. But rather it finds match of words such as “authentication “, “confirmation“ and “verification”. So, our approach uses both “String matching” and WordNet based algorithms to achieve our goal.

Due to a wider selection strategy in the specification level filtering, Web Services that match semantically with their description may also contain different I/O parameters.

So, the precision of DIQOS was less than that for FOIQOS. Our subsequent filtering method will weed out the unwanted services selected in the previous phase.

5.2.2.2 Signature Matching Recall and Precision

Table 5.8 : Comparison of various threshold values at signature matching

Q.No	Threshold 0.4			Threshold 0.5			Threshold 0.6		
	Recall %	Precision %	F-Measure %	Recall %	Precision %	F-Measure %	Recall %	Precision %	F-Measure %
1	92	72	81	92	92	92	75	94	83
2	94	75	83	92	86	88	83	87	85
3	92	72	81	84	80	81	80	83	81
4	93	69	79	93	82	87	91	86	88
5	10	68	81	100	78	87	96	80	87
6	92	72	81	92	86	88	89	89	89
7	89	70	78	89	80	84	85	83	84

Table 5.8 shows recall, precision and F-Measure calculated for seven queries with the threshold values 0.4, 0.5 and 0.6. The overall F-Measure for threshold 0.4, 0.5 and 0.6 is calculated by taking the average of the respective F-Measures scores of individual experiments. The average F-Measure for threshold values of 0.4, 0.5, and 0.6 are 81, 87 and 85, respectively. The matching process with the threshold value 0.5 produces a higher F-Measure value when compared to those of threshold values 0.4 and 0.6. The increase in the F-Measure indicates a better discerning ability (Budanitsky et al, 2001, Pinnis, 2012). So, in the signature matching we used 0.5 as a standard measure to prune irrelevant services and to retrieve more relevant services.

Table 5.9 shows the recall and precision for the signature matching of DIQOS (Google Distance only, WordNet-assisted Google Distance) and FOIQOS (Google

Distance only). By comparing the recall in Table 5.6 with Table 5.9, DIQOS (Google Distance only) and FOIQOS (Google Distance only) approaches show low recall when compared to the recall value obtained for each query in the specification matching. On the other hand, DIQOS (with WordNet-assisted Google Distance) outperforms than other with respect to recall, and also maintains the precision values obtained from the specification matching. DIQOS (WordNet-assisted Google Distance) shows 4 % and 18% higher F-Measure value than DIQOS (Google Distance only) and FOIQOS respectively.

Table 5.9 : Signature level recall and precision of DIQOS & FOIQOS

Q. No	DIQOS (Google Distance)				DIQOS (WordNet-assisted Google Distance)				FOIQOS			
	Services retrieved	Recall %	Precision %	F-Measure %	Services retrieved	Recall%	Precision%	F-Measure %	Services retrieved	Recall %	Precision%	F-Measure %
1	11	83	92	87	12	92	92	92	5	42	100	59
2	25	88	86	86	27	92	86	88	18	64	89	74
3	18	78	80	78	19	84	80	81	12	62	93	74
4	15	86	82	83	16	93	82	87	10	67	91	77
5	13	91	78	84	14	100	78	87	3	27	100	42
6	12	83	86	84	13	92	86	88	9	67	91	77
7	8	75	80	77	9	89	80	84	5	62	100	76

The results for a query ‘Vehicle Booking’ at the signature level filtering stage of both approaches are displayed in the Table 5.10. The total number of relevant services in the test collection for ‘Vehicle Booking’ is 12. DIQOS (Google Distance only) retrieves 11 services, among them 10 are relevant and 1 is irrelevant. It shows recall and precision

at 83 and 92 respectively. DIQOS (WordNet-assisted Google Distance) retrieves 12 services. Among them 11 are relevant and 1 is irrelevant. It has both recall and precision at 92. FOIQOS (Google Distance only) retrieves 5 services with all 5 being relevant and thus it shows recall and precision at 42 and 100 respectively.

Both DIQOS (Google Distance only) and DIQOS (WordNet-assisted Google Distance) approaches retrieve all relevant services in the test collection but missed a relevant service 'Contrat form' in specification matching due to low similarity score. Even though 'Contrat form' is considered as relevant service, our approach filtered it because of insufficient specification details. In signature matching, DIQOS (Google Distance only) missed another relevant service 'Car agreement' because in the I/O parameter matching between 'Lease Form' and 'Travel Document', Google Distance produces a similarity score of 0.25, which is lower than the threshold and hence removed as an irrelevant service. But adding WordNet to the Google Distance means those words are checked before filtering any as irrelevant. The DIQOS (WordNet-assisted Google Distance) approach gives a more valid similarity score for 'Car agreement' as 0.72 and thus it is able to retrieve all the relevant services.

FOIQOS (Google Distance only) uses Google Distance only and produces low similarity scores for a relevant service 'Car agreement' and removes it as irrelevant.

Table 5.10 : Sample output at signature level

Query	Relevant Services	Retrieved Services (DIQOS-Google Distance)	Retrieved Services (DIQOS WordNet-assisted Google Distance)	Retrieved Services (FOIQOS)
Search Vehicle dealers Search insurance company Car agreement form	Rental car form Find insurance company Explore company insurance Car agreement Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp	Rental car form Find insurance Company Explore company insurance Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Search cheap flight	Rental car form Find insurance Company Explore company insurance Car agreement Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Search cheap flight	Serch vehicle dealer Find insurance company Search vehicle daler Vehicle dealer Search insurance comp
Precision		92	92	100
Recall		83	92	42

DIQOS (Google Distance only) and FOIQOS (Google Distance only) use the Google Distance-only approach for signature matching and produce a lower number of relevant services compared to DIQOS (WordNet-assisted Google Distance). All three approaches work across all domains and do not rely on ontologies. Google Distance can be used to find the similarity between semantic words, short-forms words, and misspelled words using the Web. But it is possible that two synonyms words may not occur together in the Google documents. Consequently, it can reduce the similarity score between two semantic words. Following are examples of words that usually do not occur together in Web pages. Table 5.11 shows the similarity score calculated for sample input/output parameters of services using the Google Distance and the WordNet-assisted Google Distance.

Table 5.11 : Compare Google Distance and WordNet-assisted Google distance approach for signature matching

Input/Output parameter (R)	Input/Output parameter (C)	Similarity Score (Google Distance)	Similarity Score (WordNet-assisted Google Distance)
Cuisine name	Cafeteria name	0.41	0.65
Surrounding mood	Atmospheric	0.33	0.56
Lease form	Travel document	0.25	0.72
Air fare	Flight token	0.36	0.89
Shop location	Boutique address	0.37	0.88
Upgrade offer	Promotional offer	0.39	0.72

Google Distance-only approach is considered as the better approach for signature matching when the data set taken for examination has fewer synonyms. WordNet-assisted Google Distance is considered as the better approach for signature matching when the data set taken for examination has lots of synonyms. Due to addition of WordNet, the

total execution time will increase. For DIQOS, the choice of using Google Distance – only or WordNet-assisted Google Distance for signature matching is determined by the characteristics of the I/O parameters used in domains, specifically level of synonym intensity in a domain.

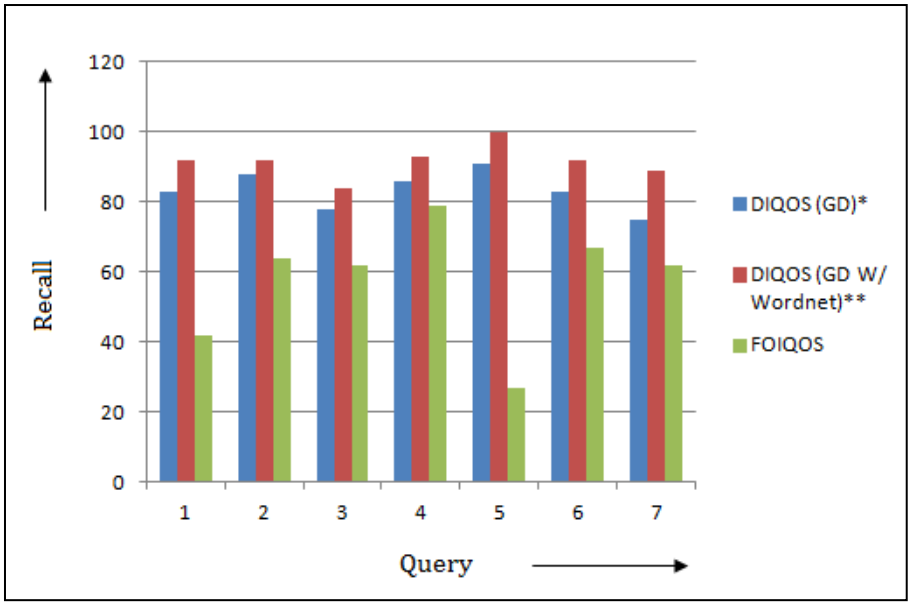

Figure 5.7 : Recall at signature matching

* DIQOS (Google Distance only)

** DIQOS (WordNet-assisted Google Distance)

Figure 5.7 shows recall of signature matching of DIQOS (Google Distance only), DIQOS (WordNet-assisted Google Distance), and FOIQOS (Google Distance only). DIQOS (WordNet-assisted Google Distance) shows (5-15 %) increase in the recall compared to DIQOS (Google Distance). DIQOS (Google Distance) shows an average of 25% increase in the recall compared to FOIQOS (Google Distance). DIQOS (WordNet-assisted Google Distance) shows 34% increase in the recall compared to FOIQOS

(Google Distance). It shows that our proposed approach retrieves all possible services, which matches both textual descriptions and input/output parameters.

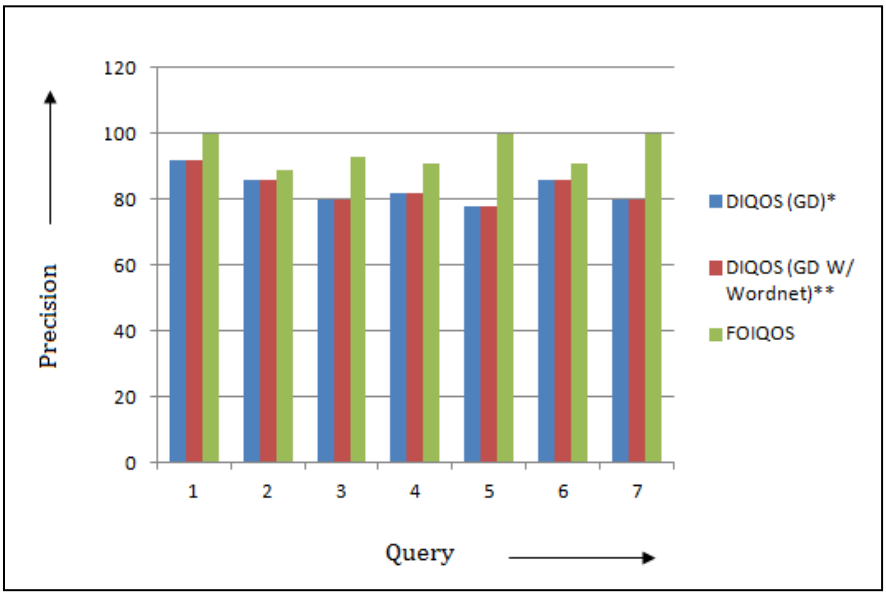

Figure 5.8 : Precision at signature matching

Figure 5.8 shows precision of signature matching of DIQOS (Google Distance only), DIQOS (WordNet-assisted Google Distance), and FOIQOS (Google Distance only). Both DIQOS (Google Distance) and DIQOS (WordNet-assisted Google Distance) maintains same precision because if the two words are not meaningfully similar, both Google Distance-only and the WordNet hybrid in DIQOS produce low similarity scores and prune the service as irrelevant. Both DIQOS approaches produce 11% lower precision than FOIQOS.

5.2.3 Finding the Best Composition Solution

We define the following strategy to find the best composition solution.

a) Candidate service selection: For the user requested workflow, Web Service instances are selected by matching the specifications and signatures of Web Services lexically and semantically with each of the service template descriptions in the workflow.

b) Best Candidate selection: Each Web Service instance in the solution must have the best QoS score among all the candidate service selection. The selected Web Service instance must have smaller QoS values to satisfy the user's request.

c) Best Composition solution: Web Services instances chosen must be relevant to requested query to obtain best composition solution. In addition to that, to accomplish the best composition solution, the Web Services chosen must have the best QoS scores. The Web Services with the best QoS score must possess the least Time value, least Cost value, and greatest Reliability value. The QoS score is the overall QoS values based on Time, Cost, and Reliability values, which is calculated by using the formulae discussed in chapter 3.

Table 5.12 : Best composition solution

Q.NO	Query	% time Best Web Service Composition is achieved		
		DIQOS (Google Distance only)	DIQOS (WordNet assisted-Google Distance	FOIQOS
1	Rental car	70%	80%	40%
2	Hotel search	90%	100%	60%
3	E-mail	80%	90%	50%
4	Weather	70%	80%	60%
5	Flight	80%	90%	40%
6	Shopping	90%	100%	80%
7	Login	90%	100%	70%

For computing the correctness of Web Service composition, seven different queries are executed by increasing the number of Web Services (adding 10 each time). The values in Table 5.12 represent the percentage of times the best composition solution was obtained for each query. Here, each query is executed 10 times by increasing 10 Web Services each time in a data set. The best Web Service composition was tracked manually. DIQOS (Google Distance only) discovers the best QoS compositions over 80% of the time, DIQOS (WordNet-assisted Google Distance) discovers the best QoS compositions over 90% of the time and FOIQOS compares at over 50% across all transactions. The results also show that the ability to obtain a best composition more often is linked to the characteristics of the words in the Web services' WSDL descriptions. For example, the shopping transaction query executed in FOIQOS and obtained the best composition solution at 80% whereas DIQOS (Google Distance only) produced 90% best composition solution and DIQOS (WordNet-assisted Google Distance) produces 100%. The 'Rental car' transaction yields a wider 40% gap between the DIQOS (WordNet-assisted Google Distance and FOIQOS (Google Distance) methods. DIQOS (WordNet-assisted Google Distance) produced the best composition solution for all the queries when compared to other approaches.

Table 5.13 : Illustration of best composition solution

Query: Flight Booking			
Retrieved services in DIQOS	QoS	Retrieved services in FOIQOS	QoS
Air Ticket Purchase	0.24	Flight Booking	0.41
Flight Booking	0.41	Flig Booking	0.45
Flig Booking	0.45		
Best Candidate selection: Air Ticket Purchase	0.24	Best Candidate selection: Flight Booking	0.41

Table 5.13 illustrates the best composition solution with a sample query "Flight Booking". As FOIQOS does not perform semantic matching in specification level, it loses service "Air Ticket Purchase" even though it matches with the requested query and it possesses lower QoS than other retrieved relevant services. On the other hand, DIQOS retrieves all the relevant services "Air Ticket Purchase", "Flight services" and "Flig services" because DIQOS performs service matching both lexically and semantically. It picks the service "Air Ticket Purchase" which possesses lesser QoS value as well as matches with the requested query. Consequently DIQOS produces a best composition solution using the best candidate selection than FOIQOS.

We used the same QoS algorithm in all the approaches. The Web Service instances discovery depends on the specification and signature level filtering. The FOIQOS algorithm does not discover all relevant services. When we apply QoS algorithm, it can find the best among those retrieved ones, which may not be the best composition result. But, our algorithm retrieves almost all relevant services that match with service description and I/O parameters. When we applied the QoS algorithm on retrieved relevant services, the desired best services are chosen most of the times for a composition.

5.2.4 WordNet-only for Signature Matching

As WordNet improves the Google Distance method in signature matching, it may be worth examining a WordNet-only implementation in the signature matching stage. Table 5.14 shows the results for a WordNet-only variant of DIQOS. Recall is much lower

for the WordNet-only variant. It may be concluded that Google Distance is a useful method for signature matching, and the hybrid variant appears to perform best for this particular data set. It is speculated that there are words in the input and output parameters of candidate web services that do not appear in WordNet but which do appear together in Web pages indexed by the Bing search engine.

In addition, in signature matching, DIQOS (WordNet only) missed relevant services such as 'Hire vehicle form', 'Search vehicle daler' and 'Find insurance Company'. For example, in 'Find insurance Company', the I/O matching between 'Insur company' and 'insurance company' produces a similarity score of zero. WordNet can obtain a valid similarity score only for meaningful words but it will produce similarity score 0 for short-forms and spelling errors. So DIQOS (WordNet only) misses relevant services, which produces low recall.

Table 5.14 : Sample output at signature level (DIQOS-WordNet)

Query	Relevant Services	Retrieved Services (DIQOS-WordNet)	Retrieved Services (DIQOS-Google Distance)	Retrieved Services (DIQOS WordNet-assisted Google Distance)	Retrieved Services (FOIQOS)
Search Vehicle dealers Search insurance company Car agreement form	Rental car form Find insurance company Explore company insurance Car agreement	Rental car form Explore company insurance Serch vehicle dealer	Rental car form Find insurance Company Explore company insurance	Rental car form Find insurance Company Explore company insurance	Serch vehicle dealer Find insurance company Search vehicle daler

	Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Contrat form	Vehicle dealer Car dealer Search insurance comp Dealer vehicle search Search cheap flight	Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Search cheap flight	Car agreement Hire vehicle form Serch vehicle dealer Vehicle dealer Search vehicle daler Car dealer Dealer vehicle search Search insurance comp Search cheap flight	Vehicle dealer Search insurance comp
Precision		88	92	92	100
Recall		58	83	92	42

Table 5.15 : Signature level recall and precision of DIQOS approaches

Query .No	DIQOS (WordNet)			DIQOS (Google Distance)			DIQOS (WordNet-assisted Google Distance)		
	Recall %	Precis-ion %	F-Measure %	Recall %	Pre cis-ion %	F-Measure %	Recall %	Precision %	F-Measure %
1	58	88	69	83	92	87	92	92	59
2	68	85	75	88	86	86	92	86	74
3	62	73	67	78	80	78	84	80	74
4	72	77	74	86	82	83	93	82	77
5	55	67	60	91	78	84	100	78	42
6	75	82	78	83	86	84	92	86	77
7	63	71	66	75	80	77	89	80	76

The following Figures 5.9 and 5.10 and Tables 5.14, 5.15 and 5.16 show a comparison of recall, precision, F-Measure and execution time for DIQOS-WordNet, DIQOS-Google Distance, DIQOS-Google Distance with WordNet, and FOIQOS. Table 5.15 shows that comparison of recall, precision and F-Measure of DIQOS variants. DIQOS (WordNet-assisted Google Distance) shows 11% increase in F-Measure value compares to WordNet only approach.

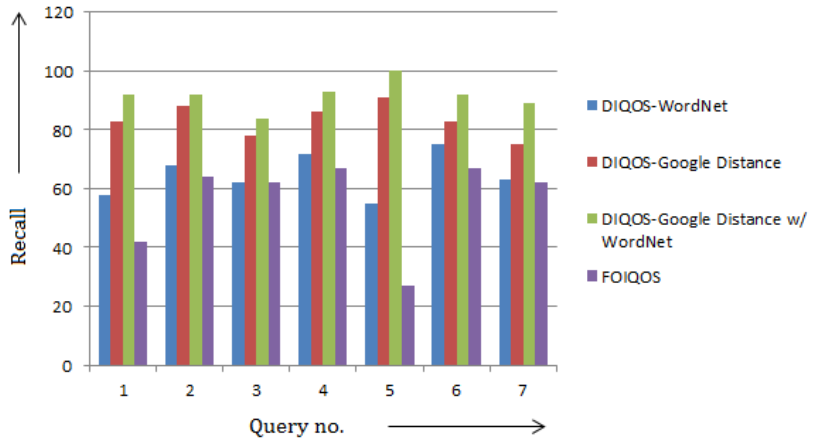

Figure 5.9 : Recall at signature matching (DIQOS-WordNet)

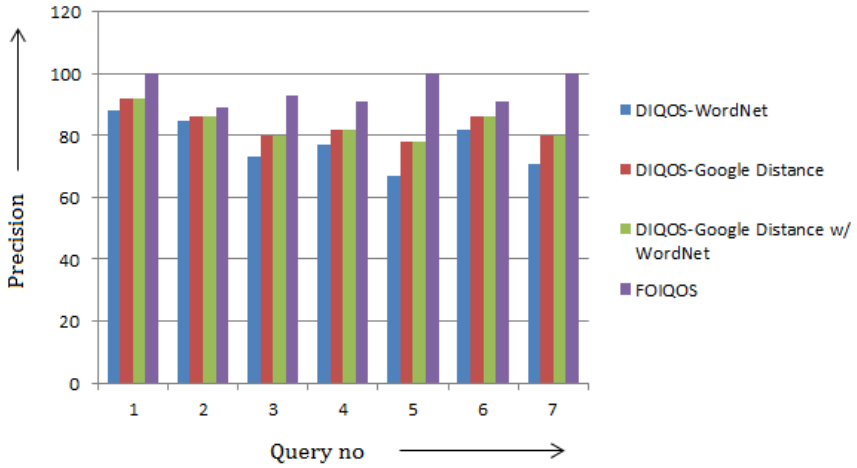

Figure 5.10 : Precision at signature matching (DIQOS-WordNet)

Table 5.16 : Execution time on each phase of DIQOS (WordNet)

No. of Web Services	Before Matching (incl.WordNet Init,Loading Input &UDDI search) (ms)	Matching stage (WordNet) (ms)	Composition (ms)	Total Execution time (ms)
10	1103	1885	765	3753
20	1181	3848	763	5792
30	1297	6256	767	8320
40	1421	8146	761	10328
50	1551	10843	768	13162
60	1665	12779	762	15206
70	1925	15200	765	17890
80	2064	17924	769	20757
90	2184	19933	763	22880
100	2341	23243	761	26345

CHAPTER 6 : SUMMARY AND CONCLUSIONS

Due to rapid growth in usage of Web Services, demand for better dynamic discovering and composition mechanisms are increasing (Tao, 2005). Many investigations contribute to the Web Service discovery and composition domains. The Web Service discovery mechanism is mainly performed on the functional properties (Input/output parameters) and non-functional properties (QoS). This thesis proposes the Domain Independent Quality of Service (DIQOS) method, an approach rooted in using WordNet and Google resources together, as well as a bigram-based method to optimize specification similarity matching. The performance evaluation exercise illustrates the tradeoffs among variants of DIQOS, including a pure WordNet-based variant, and FOIQOS, an earlier semantic-poor method for QoS-enabled Web Services Discovery.

The results show that the WordNet-assisted Google Distance shows superior recall and precision when compared to all other DIQOS variants. If speed is a heavily weighted requirement, and QoS parameter values are good enough for the identified services to compose, then FOIQOS is recommended.

To elaborate on the above, Web Service discovery is categorized into three stages: specification, signature and operational matching. The specifications of a Web Service such as name and description are considered to be very important, as they contain brief explanation about the functionality of the service. The signature of Web Service refers to input and output parameters of the Web Service, which helps to identify the service that matches with their input/output concepts. The non-functional attributes, such as time, cost

and reliability represent the quality of service parameters that identify the best candidate service that satisfies users' requested QoS metrics, among the retrieved services. The discovery process contains two logical sections: (1) Web Service matching, and (2) Web Service selection. The Web Service matching includes specification and signature matching to filter out the unwanted services instances. In this thesis, we used a specification-matching algorithm as a first step of the matching process followed by a signature-matching step. Finally, an operational matching algorithm is used to select the best matching service instances for the actual QoS-enabled composition.

Variants of the thesis' DIQOS approach (DIQOS-Google Distance-only, DIQOS-WordNet-Google Distance, and DIQOS-WordNet) are compared to FOIQOS. FOIQOS uses string matching algorithms, such as q-gram and edit distance, to perform specification matching. It helps to find the service specification that matches lexically but fails to find matches for semantic words. DIQOS improves the specification matching to find matching between service specifications lexically as well as semantically. In this thesis, WordNet is used to improve the semantic matching of service descriptions. Both lexical and semantic algorithms for service names and description matching are used. For service names matching, the vector-based WordNet algorithm is used to perform semantic matching of service names and Dice-coefficient with bigrams is used to perform lexical matching of service names. For service description matching, Dice-coefficient with bigram as well as semantic short-sentence algorithm is used to perform lexical and semantic matching of service descriptions. FOIQOS uses Google Distance only approach for semantic matching of I/O parameters and it fails to find similarity for certain I/O parameter semantic words. However DIQOS uses WordNet-assisted Google Distance to

improve the semantic matching of I/O parameters. The QoS operational matching algorithm is the same for both approaches.

Our experiments measure the efficiency of the selection process by using standard evaluation measures: delay, recall and precision. Further, experiments produce a key outcome measure: best QoS-enabled web services composition. In total execution time comparison, FOIQOS performs a little better than DIQOS because it does not have the WordNet overhead for service selection. On comparing the recall for specification matching, DIQOS retrieves higher number of relevant services compared to FOIQOS. When comparing the recall for DIQOS (Google Distance only), DIQOS (WordNet-assisted Google Distance) and FOIQOS (Google Distance), the experimental results show that the DIQOS (WordNet-assisted Google Distance) retrieves higher number of relevant services than other two approaches in signature matching because DIQOS (WordNet-assisted Google Distance) performs service matching semantically using WordNet which matches I/O parameters semantically and retains the relevant services. On the other hand, DIQOS approaches possess lower precision than FOIQOS in the specification matching, but it is better in the signature level matching. DIQOS performs composition with the relevant service instances with best QoS value at 95 % of the time whereas FOIQOS produce best composition at 74 % of the time across all the transaction.

6.1 FUTURE WORK

In this thesis, service discovery is based on the Web Service specifications published by the service providers in the UDDI registry and composition of Web Services is performed dynamically utilizing WSDL files. As a future work, we may perform WSDL structural matching by exploiting the semantics of the WSDL

description's identifiers and the structure of their operations, messages and their data types. It is possible that WSDL structural matching will enhance DIQOS to increase recall and precision, but at the expense of further delay costs.

References:

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., et al. (2003). Business Process Execution Language for Web Services Version 1.1. *Standards proposal by BEA Systems International Business Machines Corporation and Microsoft Corporation*.

Arkin, A., Askary, S., Fordin, S. Zimek, S. (2002). Web Service Choreography Interface (WSCI) 1.0. Retrieved on June 10th,2012 from http://www.w3.org/TR/wsci/

Bellwood, T., Capell, S., Clement, L., Colgrave, J., & Dovey, M. (2002). UDDI Technical White Paper. Retrieved on June 14th, 2012 from http://uddi.org/pubs/uddi_v3.htm

Blum, A. (2004, February 24). Extending UDDI with Robust Web Services Information. Retrieved on May 22nd, 2012 from http://searchsoa.techtarget.com/news/952129/Extending-UDDI-with-robust-Web-services-information

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, L., Ferris, C., & Orchard, D. (2002). Web Service Architecture. Retrieved on June 4th, 2012 from http://www.w3.org/TR/ws-arch/

Box, D., Ehnebuske, D., Kakivaya, G. Mendelsohn. (2000) N. Simple Object Access Protocol (SOAP) 1.1. Retrieved on January 4th, 2012 from http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

Bray, T., Paoli, J., Sperberg, C. M., Maler, E., & Yergeau. F. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition).Retrieved on June 21th, 2012 from http://www.w3.org/TR/REC-xml/.

Budanitsky, A., & Hirst, G. (2001). Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures, Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the

Association for Computational Linguistics, Pittsburgh. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.545

Caroso, J., & Sheth, A. (2003). Semantic E-Workflow composition. *Journal of Intelligent Information System, 21(3),*191 – 225. http://dl.acm.org/citation.cfm?id=940053

Casati, F., Ilnicki, S., Jin, L.-jie, Krishnamoorthy, V., & Shan, M.-C. (2000). Adaptive and dynamic service composition in eFlow. *CAiSE 00 Proceedings of the 12th International Conference on Advanced Information Systems Engineering* (pp. 13-31). http://dl.acm.org/citation.cfm?id=679914

Cerami, E. (2002). Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly Media. Retrieved on May 27th, 2012 from http://shop.oreilly.com/product/9780596002244.do

Chapman, S. (2006). String Similarity Metrics for Information Integration", Retrieved on March 18, 2012 from http://www.dcs.shef.ac.uk/~sam/stringmetrics.html

Chen, L., Yang, G., Wang, D., & Zhang, Y. (2010). WordNet-powered Web Services Discovery Using Kernel-Based Similarity Matching Mechanism. Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium, 64-68.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association of Computational Linguistics, Santa Cruz, CA*, *1996* (pp.184-191).

Ding, R., & Jutla, D. (2011). Flexible Ontology-Independent and QOS-enabled Dynamic Web Services Composition Using Google Distance. *Services Computing (SCC), 2011 IEEE International Conference*, 266 – 273. http://dl.acm.org/citation.cfm?id=2061970

Dong, X., Halevy, A., Madhavan, J., Nemes, E., & Zhang, J. (2004). Similarity Search for Web Services. In Proceedings of the 30th VLDB Conference, Toronto, Canada 2004 (pp. 372-383). http://dl.acm.org/citation.cfm?id=1316723

Dong, Z. & Dong, Q. (1990). Hownet. Retrieved on June 7th, 2012 from http://www.keenage.com/zhiwang/e_zhiwang_r.html

Dumas, M., Aalst, W., Hofstede, A., & Wohed, P. (2002). In: *QUT Technical report, FIT-TR-2002-05*. Queensland University of Technology, Brisbane. Retrieved on May 4th, 2012 from http://www.doc88.com/p-704872319325.html

Evangelista, A., & Kjos-Hanssen, B., (2006). Google Distance Between Words. University of Connecticut. Retrieved from http://arxiv.org/abs/0901.4180.

Faloutsos, C., & Oard, D. W. (1995). A survey of information retrieval and filtering methods. *Compare A Journal Of Comparative Education*,*8958546*(CS-TR-3514), 1-24. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=1 0.1.1.50.137 0&rep=rep1&type=pdf

Fang, Q., Peng, X., Liu, Q., & Hu, Y. (2009). A Global QoS optimizing web services selection algorithm based on MOACO for dynamic web service composition. *Information Technology and Application, 1*, 37-42.

Gilleand, M., (2006). Levenshtein Distance, in Three Flavors. Retrieved on June 4th, 2012 from http://www.merriampark.com/Id.htm

Heather, K. (2001). Web Services Conceptual Architecture. Retrieved on March 14th, 2012 from http://www.cs.uoi.gr/~pitoura/courses/ds04_gr/webt.pdf

Jiang, J. & Conrath, D. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the International Conference on Research in Computational Linguistics,Taiwan 1997*. (pp. 19-33). http://arxiv.org/abs/cmp-lg/9709008

Johansson, R., & Nugues, P. (2007). Incremental dependency parsing using online learning, In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL-2007*, (pp. 1134 - 1138)

Keller, A., & Ludwig, H. (2003). The WSLA framework: Specifying and monitoring

service level agreements for web services. *Journal of Network and Systems Management*, *11*(1), 57-81. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.5870

Kondrak, G., Marcu, D., & Knight, K. (2003). Cognates Can Improve Statistical Translation Models. In *Proceedings of HLT-NAACL 2003: Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics 2003*. (pp. 46–48).

http://dl.acm.org/citation.cfm?id=1073499

Leacock, C. & Chodorow, M. (1998). WordNet: An electronic lexical database. MIT Press, Chapter Combining local context and WordNet similarity for word sense identification, 265–283.

Li, W., & Xiang, H. (2010). A Web Service Composition Algorithm Based on Global QoS Optimizing with MOCACO. *Algorithms and Architectures for parallel processing, 6082/2010*, 218-224.

Li, Y., McLean, D. A., Bandar, Z. A., O'Shea, J., & Crockett, K. (2006). Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, *18*(8), 1138-1150.

Lin, D. (1998). An Information-Theoretic Definition of Similarity. In J. Shavlik (Ed.), *Quality* (Vol. 1, pp. 296-304).

Ludwig, H., Keller, A., Dan, A., & King, R. (2003). A Service Level Agreement Language for Dynamic Electronic Services. *Proceedings Fourth IEEE Int. Workshop on Advanced Issues of ECommerce and WebBased Information Systems WECWIS 2002*, *59*, 25-32.

Ma, J., Zhang, Y., & He, J. (2008). Web Services Discovery Based on Latent Semantic Approach, Web Services, 2008. ICWS '08. IEEE International Conference, 740-747.

Manning, C., Raghavan., P., Schutze, H. (2008). Introduction to Information Retrieval, Cambridge University Press. 2008. Retrieved on July 30 from http://nlp .stanford.edu/IR-book/

Marneffe, M., &Manning, C. (2008). Stanford typed dependencies manual, Retrieved on May 4th, 2012 from http://nlp.stanford.edu/software/stanford-dependencies.shtml

Maximilien, E.M. (2004). A framework and ontology for dynamic Web services selection, *IEEE Internet Computing,* 84-93.

Mendelsohn, N., Gudgin, M., Hadley, M., Moreau, J., Nielsen. H., Karmarkar, A, & Lafon, Y. (2000). SOAP Version 1.2 Part 1: Messaging Framework. Retrieved on January 4th, 2012 from http://www.w3.org/TR/soap12-part1/

Menasce, D. A. (2002). QoS Issues in Web Services, *IEEE Internet Computing, 6(6),* 72-75.

Meredith, G., Christensen, E., Curbera, F., & Weerawarana, S. (200). Web Services Description Language (WSDL) 1.1. Retrieved from http://www.w3.org/TR/wsdl

Miller, G. (1995). WordNet: A Lexical Database for English. Communications of the ACM, 38(11), 39-41.

Oliva, J., Serrano, J., Castillo, M., & Iglesias, A. (2011). SyMSS: A syntax-based measure for short-text semantic similarity. Data & Knowledge Engineering 70, 390–40.

Oundhakar, S., Verman, K., Sivashanmugam, K., Sheth, A., & Miller, J. (2005). Discovery of Web Services in a Multi-Ontology and Federated Registry Environment. *International Journal of Web Services Research (IJWSR), 2*(3), 1-32. doi:10.4018/jwsr.2005070101. http://www.igi-global.com/article/discovery-web-services-multi-ontology/3062

Pinnis, M. (2012). Latvian and Lithuanian Named Entity Recognition with TildeNER. In N. Calzolari, K. Choukri, T. Declerck, M. Uğur Doğan, B. Maegaard, J. Mariani,

J. Odijk, et al. (Eds.), *Proceedings of the Eight International Conference on Language Resources and Evaluation LREC12* (pp. 1258-1265). European Language Resources Association (ELRA). Retrieved from http://www.lrec-conf.org/proceedings/lrec2012/pdf/948_Paper.pdf

Ponnekanti, S.R., & Fox, A. (2002). SWORD: A developer toolkit for Web service composition. *In Proceeding of the 11th World Wide Web Conference*, (pp. 24-32).

http://www2002.org/CDROM/alternate/786/

Rajendran, T., & Balasubramanie, P. (2009). Analysis on the Study of QoS-Aware Web Services Discovery. Journal of Computing,1(1), 119-130. http://arxiv.org/abs/0912.3965

Ran, S. (2003). A Model for Web Services Discovery with QoS. *ACM SIGecom Exchanges 4(1),* 1-10.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on AI*. (pp.448-453). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5277

Sanchez, D. (2007). Learning non-taxonomic relationships from web documents for domain ontology construction. Data & Knowledge Engineering, 64 (3) : 600 DOI: 10.1016/j.datak.2007.10.001

Schuster, H., Georgakopoulos, D., Cichocki, A., & Baker, D. (2000). Modeling and composing service-based and reference process-based multi-enterprise processes. *In Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE).* http://dl.acm.org/citation.cfm?id=680058

Segev, A. (2009). Context-Based Matching and Ranking of Web Services for Composition. *IEEE Service Computing,* 3(2), 210-222.

Smadja, F., & McKeow, K. (1994). Translating Collocations for Use in Bilingual Lexicons. In *Proceedings of the workshop on Human Language Technology*, *PA, USA*, 1994 (pp. 152-156).

Stroulia, E., & Wang, Y.(2005). Structural and Semantic Matching for Assessing Web-service Similarity. *International Journal of Cooperative Information Systems*, 14(4), 407-438.

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.5200

Tao, H. (2005). An Evaluation Model for Web Services, Neural Networks and Brain, 2005. ICNN&B '05. International Conference, 1, 529-532.

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=1614669&contentType=Conference+Publications

Vitanyi, P., & Cilibrasi, R., Automatic Measuring Discovery Using Google. (2007). Retrieved from 2012 http://www.arxiv.org/abs/cs/0412098

Wang, Y., & Stroulia, E. (2003). Semantic Structure Matching for Assessing Web Service Similarity. *1st International Conference on Service Oriented Computing ICSOC03*, 194-207.

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.5200

Wu, Z., & Palmer, M. (1994). Verb Semantics and Lexical Selection. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 6. Association for Computational Linguistics.

W3C. (2012). Extensible Markup Language (XML). Retrieved on April 24th, 2012 from http://www.w3.org/XML/

W3C Working Group (2004). Web Services Architecture, W3C Working Group Note 11 February 2004. Retrieved April 30, 2012 from http://www.w3.org/TR/ws- arch

Wikipedia: F1 score. (2012, June 15). In *Wikipedia, The Free Encyclopedia*.

Retrieved 21:29, July 30, 2012, from http://en.wikipedia.org/w/index.php?title=F1_score&oldid=502498831

Wikipedia: Precision and Recall. (2012, February 28). FL: Wikimedia Foundation, Inc. Retrieved April 02, 2012, from http://www.wikipedia.org

Xu, Z., Martin, P., Powley, W., & Zulkernine, F. (2007). Reputation-Enhanced QoS-based Web Services Discovery. *IEEE Int. Conference on Web Services ICWS 2007*, 249-256. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.3368

Yang, H., Fu, P., Yin, B., Ma, M., & Tang, Y. (2011). A semantic similarity measure between web services based on Google distance. *35th IEEE Annual computer Software and Application conference,* Munich, Germany, 14-19.

Ye, L., & Zhang, B.(2006) Discovering Web Services Based on Functional Semantics", *IEEE Asia-Pacific Conference on Services Computing*, 348-355. http://dl.acm.org/citation.cfm?id=1191937

APPENDIX A – Web Service Requests (Queries)

1) Rental car

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
  <process id="2" type="parallel" >
 <ST id = "1">
      <Name> Find dealer </Name>
      <Description>It provides country code to find car dealers</Description>
     <Input> Area code </Input>
      <Output> Dealer name </Output>
      <QoS>
           <Time>20</Time>
           <Cost>35</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Find insurance company </Name>
      <Description>It gets zip code to view insurance providers</Description>
      <Input> ZIP </Input>
      <Output> Insurance name </Output>
      <QoS>
           <Time>30</Time>
           <Cost>45</Cost>
           <Reliability>0.6</Reliability>
      </QoS>
   </ST>
 </process>
<ST id = "3">
      <Name> Rental form </Name>
      <Description>It displays both car dealers and insurance company </Description>
      <Input> Dealer, insurance company </Input>
      <Output> Rental form </Output>
      <QoS>
           <Time>30</Time>
           <Cost>45</Cost>
           <Reliability>0.6</Reliability>
       </QoS>
   </ST>
</process>
```

2) Hotel search

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
 <ST id = "1">
      <Name> Search Hotel </Name>
      <Description>This service is to locate restaurant for given area code</Description>
     <Input> Current location </Input>
      <Output> Restaurant list </Output>
      <QoS>
           <Time>30</Time>
           <Cost>35</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Cuisine address </Name>
      <Description>It gets location for given hotel names </Description>
      <Input> Restaurant name </Input>
      <Output> Hotel address </Output>
      <QoS>
           <Time>60</Time>
           <Cost>70</Cost>
           <Reliability>0.6</Reliability>
      </QoS>
   </ST>
 </process>
```

3) E-mail notification

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
<ST id = "1">
      <Name> Travel package </Name>
      <Description> It offers travel package to their preferred users</Description>
     <Input> Travel deal </Input>
      <Output> User name </Output>
      <QoS>
           <Time>40</Time>
           <Cost>45</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Email notification </Name>
```

```
      <Description>This service helps to send package details to the provided ids
</Description>
      <Input> Email ID </Input>

      <Output> Offer notification </Output>
      <QoS>
            <Time>55</Time>
            <Cost>50</Cost>
            <Reliability>0.4</Reliability>
      </QoS>
   </ST>
 </process>
```

4) Weather prediction

```
<?xml version="1.0" encoding="UTF-8"?>
<process id="1" type="sequence" >
<ST id = "1">
   <Name> Weather forecast </Name>
   <Description>This service receives city and returns temperature </Description>
   <Input> Search city </Input>
   <Output> Temperature </Output>
   <QoS>
         <Time>55</Time>
         <Cost>66</Cost>
         <Reliability>0.2</Reliability>
   </QoS>
</ST>
<ST id = "2">
  <Name> Climate condition </Name>
  <Description> It receives temperature and display temperature conditions
  </Description>
  <Input> Weather </Input>
  <Output> Prediction </Output>
  <QoS>
        <Time>30</Time>
        <Cost>60</Cost>
        <Reliability>0.6</Reliability>
 </QoS>
</ST>
</process>
```

5) Flight booking

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
<ST id = "1">
      <Name> Search flight </Name>
      <Description>It gets airlines preferred for user request</Description>
     <Input> Travel request </Input>
      <Output> Airlines list </Output>
      <QoS>
           <Time>20</Time>
           <Cost>35</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Compare flights </Name>
      <Description>It compares flights and display cheap fares</Description>
      <Input> Flights names </Input>
      <Output> Best deal </Output>
      <QoS>
           <Time>30</Time>
           <Cost>45</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
 </process>
```

6) Shopping

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
<ST id = "1">
      <Name> Shopping </Name>
      <Description> It gets customer shopping plan</Description>
     <Input> Shopping plan </Input>
      <Output> Stores list </Output>
      <QoS>
           <Time>30</Time>
           <Cost>35</Cost>
           <Reliability>0.5</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Location finder </Name>
      <Description>This service suggests top places for shopping</Description>
```

```
      <Input>Mall name</Input>

      <Output> Shop spot </Output>
      <QoS>
            <Time>40</Time>
            <Cost>30</Cost>
            <Reliability>0.6</Reliability>
      </QoS>
   </ST>
 </process>
```

7) Login verification

```
<?xml version="1.0" encoding="UTF-8"?>
  <process id="1" type="sequence" >
<ST id = "1">
      <Name> Password check </Name>
      <Description> It gets users authenticate ids and check validity</Description>
     <Input> User Authentication ID </Input>
      <Output> Verification status </Output>
      <QoS>
            <Time>40</Time>
            <Cost>35</Cost>
            <Reliability>0.4</Reliability>
      </QoS>
   </ST>
<ST id = "2">
      <Name> Authentication confirmation </Name>
      <Description>This service helps to confirm the users confidentiality</Description>
      <Input> Login message </Input>

      <Output> Login confirmation </Output>
      <QoS>
            <Time>30</Time>
            <Cost>45</Cost>
            <Reliability>0.6</Reliability>
      </QoS>
   </ST>
 </process>
```

APPENDIX B – Web Service Description in UDDI Registry

Table B.1 : Web Service Description

NAME	DESCRIPTION
Search restaurant	It searches for restaurants available in given zipcode
Climatic condtion service	It depends on temperatre, dsplays temperature
Find best insurance	This service get zipcode to view the insurance
Cheap flight fares	It displays cheap flight fare
Country weather service	It displays weather conditions depends on
Weather forecaste service	This service receives the zipcode as input, prvide
Search inn	The service displays restaurent nearby given
Check Cheap Airlines	This service compares with all flights and sort out
Serch flights	It is based on user query displays available flight
Car agreement form	It assists in filling out car agreement form by ge
Travel package	This service used to find the best valid custokmner
Search restaurant	This service search cuisine available for given
Cheap Flight Search	It get airlines list interested for user requested query
Cheap flight ticket	It displays cheaps flight fares
Wether forecast servce	It receives input zipcode,provde weather condition
Restaurant address	This service helps to find restaurant address
Weather service	It provides temperature for given zip code
Shopping spot search	It gets customer shopping plan
Email notification	It uses the email ids to send their offer mesg
Search insurance suppliers	It helps to find insurance suppliers list
Shopping center finder	This service get customer shopping plan
Cheap Airlines finder	The service compares flights and dsplays best deal
Hotel finder	This service facilitates to find restaurant list for p
Message offer	The service is used to send greetings and offers
Fill automobile form	It displays both car dealers and insurance company
weather prediction	This service provides temperature for given zip
Insurance provider search	For the given zipcode and this service list insurance
Shop plan help	This service Get customer shopping plan
cheap airways	It shows different flights based on users request
Climtic Weather prediction	This service helps to find weather for thr given c
address restaurant	It finds address for given restaurant name
Purchase plan	The service get customer shopping plan
Vehicle dealers	input zip code to find vehicle suppliers in local
Insurance Companye	provides country code to display insurance dealers
Shopping plan	Get customer shopping plan
Hotel location find	It helps to find address of restaurant
Hotel explore	Displays cuisine address for given restaurant name
find cheap flights	It explore cheap airline fare by comparing with all list

Find car dealers	It request for postal code to search car dealers in
Compares flights	This serive inspect flights and show low fares
Search Fligh List	The service get airlines interested for user request
Cheap flights fares	It retrieves the cheap airlines for the requested
Login verification	It checks the username and password match
Cheap airways	This service shows different flights based on user
Explore flights	It takes user query and display available flights list
Explore find flight	The service used to find airlines available for users
Find insurance company	It takes zipcode to find insurance providers
Climtic condition service	This service aids to disply weather condition
Restaurant find	It finds best 119nsurance119 nearby
Automobile provider	This service provides coutry code to find car dealers
Search cheap flight	The service is used to find flight available for users
Password dreset	It send the new password to customer email id
Hire Car document	It displays both cars delaers and insurance
Country wether service	This service displays weather conditions depends
Forget password	It sends the message to reset password
Car agreement form	The agreement form contains information about car
Address restaurant	This service finds address for given
Find car dealers	It provides country code to find car dealers in your
Username reset	It send the user name to customer email
Hotel address	It gets location for given cuisine names
Restaurant name	This service aids to find hotel name for given
Climatic service	The climatic service receives temperatre and returns
Rental car form	It displays car dealers and 119nsurance companye
Hotel addresss Find	This service display cuisine address for given
Local weather service	This service predict weather condition for give
Password check	It sends the password for authentication
Help user shopping	The shopping service suggest top ten places for user
Find airlines	It assists to find flights for user travel request
Airlines search	It displays available flight lists for requirement
Service wather forecaste	This service receives zipcode as input and dsplay
Shoping plans	It suggests best places for shopping
Find cheap fares	The service helps to find cheapest flight fare
Flight deal service	It retrieves best deal flights for user request
Compare flights deal	This service aids to list flights and sort out best
Explore shop place	It suggests attractive places for users shopping
Cheap airways	This service shows different flights based on user
Airlines search	It displays available flights lists
Shopping assistance	This service suggest top ten places for shopping
Climate Check Service	It shows weather for given area code
Help shopping	This service suggest top ten places for shopping
Search hotel	This service find restaurant for area code
Find customer	This service helps to finds the VIP users
compare cheap airlines	It find cheapest flight for the destionation

weather forecaste servic	For the given zipcode as input, it prvide weather
Hotel explore	It facilitates to find restaurants list for postal
Help user shopping	This service suggest top ten places for shopping
Offer pack	It provides offer package info to customers
Cuisine address	This service takes us to find address for hotel
Restaurant address	This service helps to find restaurant address
Mailing service	The service assist in sending offer messages to
Search automobile dealers	It helps to search vehicle dealers
Hire vehicle form	It Displays vehicle suppliers and insurnce
Shopping plan	This service Get customer shopping plan
Insurance Company	It gets zip code to view insurance providers
explore find flight	It find flights available for users
Flights fares compare	This compares flights and display cheap fares
Hotel Location find8	It facilitates to find hotel list for postal code
Search vehicle dealers	This service search vehicle dealers for input zipcode
Check climate condition	This performs weather prediction depends on
Authentication confirm	This service sends the confirmation for user id and

Table B.2 : QoS and I/O parameters of Web Services

Web Service name	Inputs	Output	QoS		
			Time	Cost	Reliability
Search restaurant	Country code	Cuisine name	25	30	0.6
Climatic condtion service	Degree	Weather condition	26	45	0.7
Find best insurance	Zipcode	Insurance company	20	32	0.75
Cheap flight token	Airlines list	Cheap flight	34	42	0.65
Country weather service	Temperature	Climatic condition	42	55	0.8
Weather forecaste service	Area code	Temperature	42	62	0.5
Search inn	City name	Cafetaria list	15	20	0.6
Check Cheap Airlines	Airlines	Cheap flight token	15	23	0.6
Serch flights fare	Travel plan	Flight list fares	16	28	0.73
Car agreement form	Dealer name,	Aggrement	21	38	0.63

Travel package	Travel deal	user name	40	45	0.55
Search restaurant	Country id	Hotel name	18	30	0.65
Cheap Flight Search	Travel form	Flight list	25	42	0.58
Cheap flight ticket	Travel schedule	Airlines info	26	48	0.65
Wether forecast servce	Area code	Atmospheric condition	52	63	0.8
Restaurant address	Restaurant name	Address	22	32	0.6
Weather service	Search city	Predication	55	60	0.7
Shopping spot search	Shop plan	Store list	32	45	0.5
Email notification	Email id	Promotional notification	55	50	0.45
Search insurance suppliers	ZIP	Insurance name	23	45	0.68
Shopping center finder	Mall name	Shop spot	35	55	0.6
Cheap Airlines finder	Flight name	Best deal	28	42	0.55
Hotel finder	Current location	Restaurant list	50	60	0.6
Message offer	Email address	Offer message	45	35	0.6
Fill automobile form	Dealer name	Insurance	22	42	0.65
weather prediction	Climate degree	Predication	23	52	0.76
Insurance provider search	Posttal code	Insur comp	14	41	0.71
Boutique plan help	Shop schedule	Boutique list	42	32	0.54
cheap airways	Airlines list	Cheap fligh	21	34	0.56
Climtic Weather prediction	Area code	Temperature predicr	16	45	0.67
address restaurant	Hotel list	Location find	51	56	0.81
Purchase plan	Shop request	Mall list	34	35	0.72

Vehicle dealers	Zipcode	Dealer name	14	26	0.64
Insurance Companye	Country code	Company name	28	38	0.65
Shopping plan	User plan	Mall list	41	39	0.53
Hotel location find	Hotel name	Restaurant address	51	64	0.65
Hotel explore	Cuisuine name	Hotel location	21	32	0.67
find cheap flights	Flight available	Flight deal	24	41	0.56
Find car dealers	Current location	Dealer name	18	29	0.56
Compares flights	Airlines	Best deal	23	41	0.56
Search Fligh List	Travel request	Airlines available	19	30	0.67
Cheap flights fares	Flight list	Cheap flight	21	30	0.65
Login verification	User id	Login status	24	31	0.71
Cheap airways	Request plan	Cheap airways	23	39	0.76
Explore flights	User flight plan	Flight list token	10	29	0.74
Explore find flight	Flight schedule	Flight availability	19	26	0.58
Find insurance company	ZIP	Comp name	25	39	0.68
Atmospheric condition	Celsius	Wether prediction	31	45	0.71
Restaurant find	Postal code	Cusine list	21	32	0.56
Automobile provider	Current loc	Dealer name	18	34	0.65
Search cheap flight	Airlines list	Cheap flight	23	43	0.54
Password reset	Email id	New password	32	31	0.45
Hire Car document	Vehicle dealer, insure	Rental application	23	43	0.67
Country wether service	Degree temp	Weather prediction	24	41	0.56

Forget password	User id	New pwd id	34	54	0.5
Car agreement form	Dealer name, insurance	Agreement form	27	37	0.67
Address restaurant	Restaurant name	Hotel address	26	31	0.65
Find car dealers	Current area	Dealer name	16	32	0.50
Username reset	Email	New user id	34	52	0.65
Hotel address	Cusine name	Cafeteria addr	56	62	0.68
Restaurant name	Postal code	Restaurant available	20	26	0.57
Climatic service	Temperature	Climate prediction	23	45	0.73
Rental car form	Dealer, insure	Rental form	26	38	0.64
Hotel addresss Find	Hotel name	Hotel address	43	52	0.85
Local weather service	Climate degree	Surrounding mood	23	52	0.76
Password check	Pwd	Authentiction status	14	41	0.71
Help user shopping	Shop list	Places suggestion	42	32	0.54
Find airlines	Travel plan	Flight list	23	41	0.56
Airlines search	Travel request	Airline list	19	30	0.67
Service wather forecaste	Country code	Degree	21	30	0.65
Shoping plans	Shop list	Shop suggest	24	31	0.71
Find cheap fares	Airlines list	Cheap flight	23	39	0.76
Flight deal service	Flight available	Best airlines	10	29	0.74
Compare flights deal	Flight list	Cheap flight	19	26	0.58
Explore shop place	Shop plan	Explore place	25	39	0.68
Cheap airways	Airlines ilst	Cheap airways	31	45	0.71

Airlines search	Travel plan	Flight list	21	30	0.65
Shopping assistance	Shopping list	Best shop place	24	31	0.71
Climate Check Service	Area code	Temperature	22	42	0.65
Help shopping	Plan request	Shop list	23	52	0.76
Search hotel	Country code	Cusine List	14	41	0.71
Find customer	offer	Customer name	34	52	0.65
compare cheap airlines	Airlines list	Best deal	56	62	0.68
weather forecaste servic	Zip code	Climate prediction	20	26	0.57
Hotel explore	Current location	Hotel available	23	41	0.56
Help user shopping	Mall available	Location	23	52	0.76
Offer pack	Email addr	Offer message	14	41	0.71
Cuisine address	Hotel name	Hotel address	42	32	0.54
Restaurant address	Cuisine name	Cusine location	19	26	0.58
Mailing service	Customer addr	Offer message	25	39	0.68
Search automobile dealers	Current location	Auto dealer	31	45	0.71
Hire vehicle form	Auto dealer, insurance	Car aggrement	22	32	0.6
Shopping plan	Shop request	Shop location	55	60	0.7
Insurance Company	Postal code	Auto insur name	32	45	0.5
explore find flight	Costomer request	Flight available list	55	50	0.45
Flights fares compare	Flight list	Cheap fare	21	32	0.56
Hotel Location find8	Hotel name	Hotel address	18	34	0.65
Search vehicle dealers	Zipcode	Car dealer	23	43	0.54

Check climate condition	Current location	Climate prediction	16	45	0.67
Authentication confirm	User id, pwd	Authentication status	51	56	0.81