Exploiting Semantics and Syntax for Service Specification and Signature Matching: The S^5 Web Service Matchmaker

by

Syed Farrukh Mehdi

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

at

Dalhousie University
Halifax, Nova Scotia
November 2011

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 "**Exploiting Semantics and Syntax for Service Specification and Signature Matching: the S^5 Web Service Matchmaker**" by **Syed Farrukh Mehdi** in partial fulfilment of the requirements for the degree of Master of Computer Science.

Dated: November 25, 2011

Supervisor: _____

Readers: _____

_____

DALHOUSIE UNIVERSITY

DATE: November 25, 2011

AUTHOR: Syed Farrukh Mehdi

TITLE: Exploiting Semantics and Syntax for Service Specification and Signature Matching: The S^5 Web Service Matchmaker

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc CONVOCATION: May YEAR: 2012

_____

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

Web services are independent software systems designed to offer machine-to-machine interactions over the WWW to achieve well-described operations. The description of a web service entails (a) a syntactic component detailing the service's operations and data structures in terms of the Web Services Description Language (WSDL), and (b) a semantic component that offers a semantic description, in terms of an ontology, of the services' data and operations. Typically, service providers expose their services to the public by providing brief descriptions of the service's operations; the challenge is to discover the right service based on rather sparse service descriptions in response to a specific service request.

In this thesis, we present a hybrid semantic web service discovery framework that offer semantic web service discovery at both the signature and specification levels of a web service, whilst exploiting logical and non-logical service matching methods. For signature level service matching, we have developed two techniques: (i) logical similarity measures applied to the services' input/output concepts; and (b) non-logical matching based on a Structure Preserving Semantic Matching algorithm. For specification level service matching, we have applied a unique short sentence matching approach on the textual-description of a web service. The cumulative similarly measures determine the overall similarity of a services' description with the service request. We evaluated the performance of our S^5 Web Service Matchmaker using the OWLS-TC dataset, and furthermore compared its performance with the OWLS-MX discovery model. Our results indicate that S^5 Web Service Matchmaker offers an improved web service matching performance with a significant increase in recall and subtle improvements in precision.

List of Abbreviations Used

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IC	Information Content
IR	Information Retrieval
NGD	Normalized Google Distance
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
OWLS-MX	Hybrid Semantic Web Service Matchmaker
OWLS-TC	OWL-S Service Retrieval Test Collection
RDF	Resource Description Framework
RDFS	RDF Schema
SOAP	Simple Object Access Protocol
SPSM	Structure Preserving Semantic Matching
TF/IDF	Term Frequency / Inverse Document Frequency
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WSDL	Web Service Description Language
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

Acknowledgement

I would like to express the deepest sense of gratitude to my supervisor, Dr. Raza Abidi for his excellent supervision, patient guidance, encouragement and advice through the course of my research. Without his sincere help and support, I would not have been able to bring forth the quality of research and writing presented in this thesis. His wisdom and patience guided me on my way through the doors of science and made my research fruitful. I would also like to extend my thanks to my committee members, Dr. Denis Riordan and Dr. Karen Jin, for their support.

I would also like to take this opportunity to express my profound gratitude to my beloved parents, whose love and support made my study possible and fruitful. A very special thanks goes out to my fiancée, sisters and especially to my friend Ali Daniyal, for their moral support, encouragement, patience and editing assistance during my study.

I also recognize that this research would not have been possible without the financial assistance of CANARIE, Canada. This research is part of the Platform for Knowledge Management (POKM) Project, supported by a research grant from CANARIE, Canada. As the author, I fully acknowledge the funding support provided by CANARIE for this research project.

Chapter 1. Introduction

1.1 Introduction

Web service [1] is a single independent program that can be published, searched and invoked on the web. Web service embodies computing in a distributed environment—web service operates in an environment in which users, computing applications, and data sources are distributed. It provides necessary communication and interaction protocols for these distributed elements to be dynamically interconnected to perform a particular task. Traditionally, client—server applications are the main components in a distributed environment; a client application initiates an activity, whereas the server application processes the activity request and sends the response back to the client. Taking this idea to a web-centric environment demands a middleware that will not only discover the server applications but will also maintain communication with the client application. Here, web services provide the necessary distributed middleware technology between the client and server applications. It is similar to the old technologies like CORBA [42] or JAVA RMI [41] but with more technological advantages. For instance, the CORBA architecture supports the communication between two objects running on client and server applications. Java RMI supports object function calls between two java-based applications. Both Java RMI and CORBA are object-oriented distributed mechanisms which require a connection oriented communication protocol. On the other hand, web services are XML-based communication protocols that are developed to support WWW-

services via a universally supported HTTP protocol. Representing web service description in an XML format makes it simple to understand by humans as well as machines and interoperable between different incompatible applications. The basic idea of web services in a distributed environment is described as:

1. The web service provider describes the format of request and response for its service.
2. A client application makes a request using a provider's defined format over the network.
3. Web service receives a client request, processes it and generates a response.

Nowadays, web services are employed by major computing organizations to provide a range of functionalities. For example, Microsoft .Net provides the passport service to its developer. Passport allows users to access multiple web pages after a single sign-on. Microsoft's software developer's kit allows developers to access the passport service to incorporate an authentication process into their web-based applications. Hotmail is one of the running examples of passport web service. Similarly, Google provides MAP API web services that allow any map application to request map data using its interfaces. These web services take URL parameters as an HTTP request from clients and return map data in the HTTP request back to the client applications. Likewise, Amazon web services are a collection of web service interfaces that collectively provides a cloud computing platform over HTTP [43]. Amazon provides free online access to all exposed web services. For example, Amazon Simple Storage Service (Amazon S3) is one of the web services that provide online storage of data. It provides a WSDL interface that would be invoked by a web service consumer in order to store and retrieve data anywhere, anytime.

Web service architecture consists of three main entities, as shown in Figure 1.1: Service provider, Service requester and Service broker. *Service provider* is the one who creates and publishes the web service on any web service registry, like Universal Description, Discovery, and Integration (UDDI) [2]. *Service requester* is the consumer of the web service who searches for a service in a registry and invokes the service. *Service broker* is the middle layer between *service provider* and *service requester*, which allows the web service provider to publish their services and helps the consumer to search and then use the relevant services.

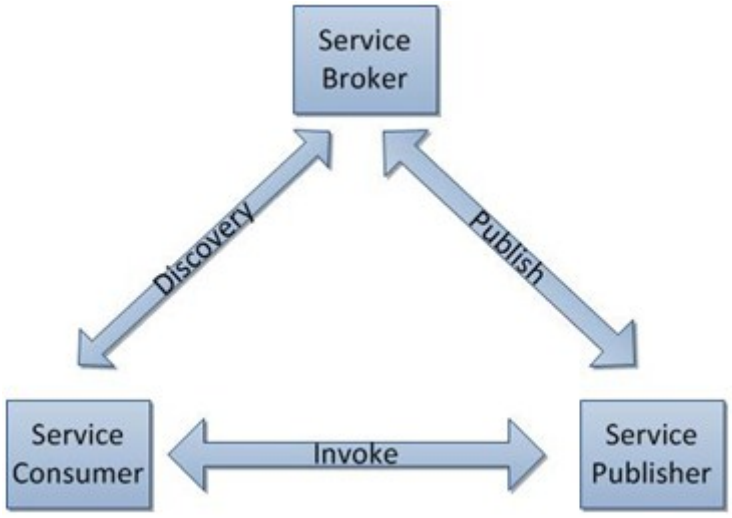

Figure 1.1: Web service data flow model

The web service lifecycle comprises three stages starting from service publishing, service discovery and then service invocation.

Service Publishing: Web service publishing is the process of making a service available for use at the web registry server. Web services are specified in terms of an XML-based language known as WSDL. The web service publishing process requires the service provider to advertise services by providing maximum information that facilitates the service consumer in search. Detailed description of web services results in effective publishing and discovering.

Service Discovery is a process of searching for a web service that satisfies certain functional requirements as requested by a service consumer. Universal Description, Discovery and Integration (UDDI) are registry servers that provide a mechanism to publish and discover web services to its users. Service discovery involves searching web service registries for particular services based on desired functionalities provided by web service consumers. Many businesses like Google, Amazon, and eBay now make their web services publically available. The challenge is finding the right service from these service registries. Web service clients need to search all the available web services in the registry and select the appropriate service that meets the desired functionalities. For example, if a client is searching for a credit card authentication web service in a registry that contains thousands of web services, it is not possible to manually search for a service. Rather, a search for a web service is usually conducted by intelligent discovery agents that take the desired functionalities as a query. These agents match consumer's desired functionalities with the designated functional descriptions of services to find a match between the service's functionality and the client's needs. Web service discovery agents take the user's requirements in terms of category and functional parameters to

search for compatible services in the service registry. The overall service discovery process is explained in Figure 1.2.

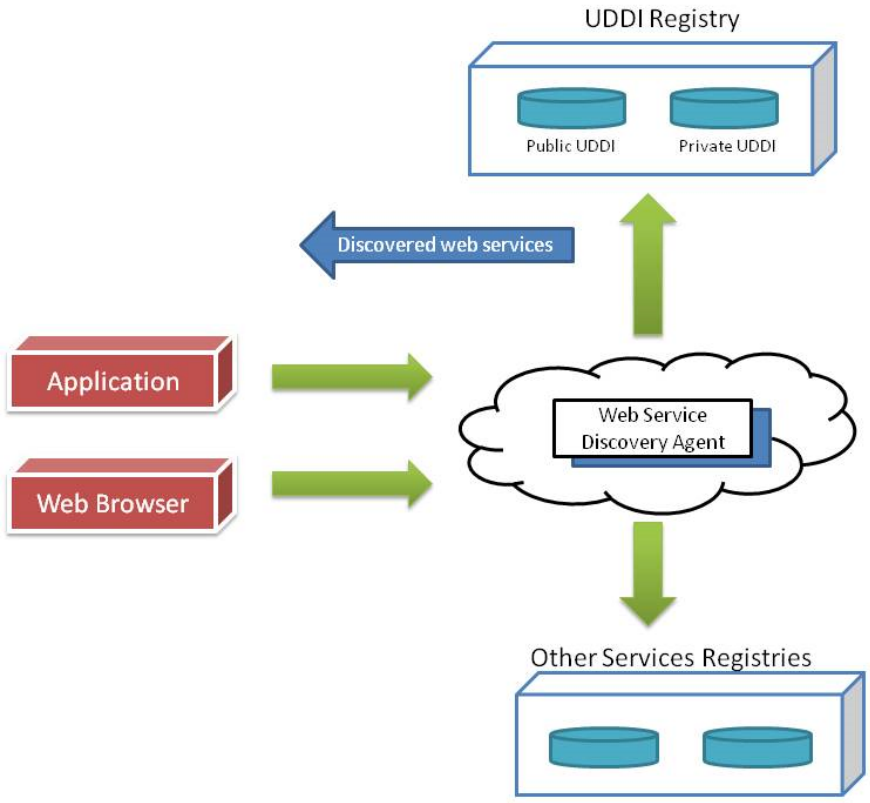

Figure 1.2: Web service discovery model

Web service discovery agents serve as a mediator between the web service client and service registries. A range of web service discovery algorithms and matchmaking techniques are used to facilitate the service discovery processes. Once the right service is discovered, the service client invokes the most suitable web service.

Service Invocation involves the exchange of messages between the service consumer and the service provider. The consumer requests and the provider responses are SOAP

messages. Once the client retrieves the description of the desired web service, he invokes the operations by providing the desired input parameters. The procedure of invocation of a web service by client is shown in Figure 1.3.

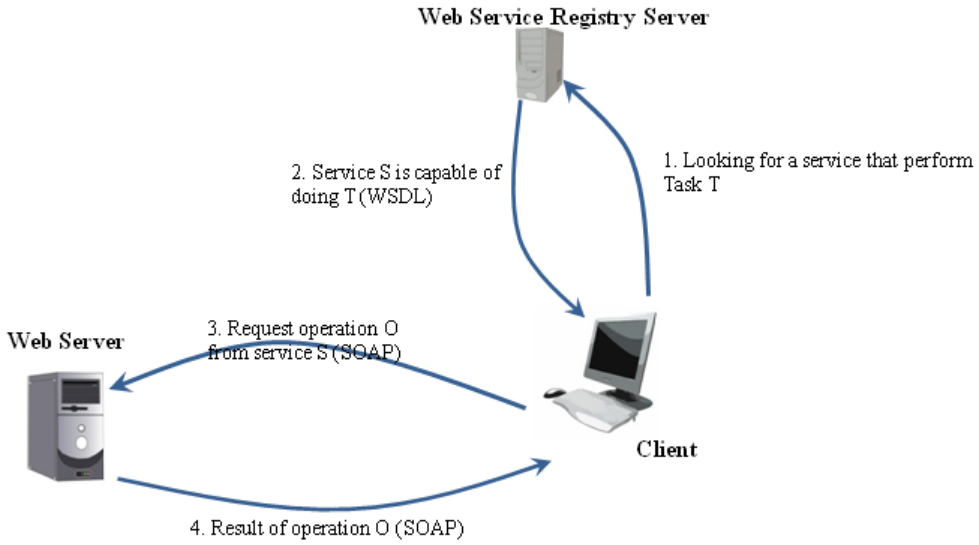

Figure 1.3: Web service invocation model

1. As we have mentioned, first a client needs to know which web service meets his requirements. A client sends his requirements to a registry server to discover relevant web services.
2. Discovering web services provides a client with the description of relevant web service(s). This description would be in a specific language (such as WSDL) which describes the service's operations. For example, a client is interested in finding a web service that provides credit card authentication. The discovery process may return a description of the web service 'CreditCardInfoService'

which has an operation, ‘authenticateCreditCard’ that takes credit card number and expiry date as parameters.

3. Once the relevant web service is discovered and the client receives the description of the right web service, the client invokes the web service to achieve its functionality. Service invocation is done by sending a SOAP request message to the service. For example, a client will send the SOAP request to a server asking for the authentication of a credit card by providing the credit card number and expiry date.
4. The server receives the SOAP request; executes the requested operation and returns the result in terms of a SOAP message. For example, in this case, it will authenticate the credit card number and return either yes/no or any error message.

1.2. Service Discovery Overview

Service discovery involves matching a service request with the description of all the available web services. This is largely pursued by examining some intrinsic characteristics of a web service, such as its inputs, outputs, textual description, and service name. A service description is categorized into two components: signatures and specifications. Web service signatures are the input/output parameters that directly correspond to function’s parameters. Web service signature does not describe the behavior of a web service. On the other hand, web service specifications are the behavioral descriptions of a web service that are explicitly defined by a web service publisher, such as textual description, category, and business. There are two main

approaches for service discovery: (a) Matching approaches involving the analysis of the service's descriptions (i.e. inputs/outputs, parameters' structures, textual-descriptions, and names) are called the non-logical matching; (b) Deductive approaches using logical concepts, rules and ontologies are called the logical matching.

In past researches for service discovery [9, 10, 11, 18, 19, 20], different non-logical similarity techniques have been applied to the signatures and specifications of a web service—the techniques aimed to determine the similarities/differences between a pair of web services' parameters, operations, textual-descriptions, and names. The non-logical technique that is widely used for the web services discovery models is keyword-based matching. Web service discovery agents (or matchmakers) perform keyword-based syntactic matching between the user query and WSDL documents and compare information, such as web service name, input/output parameter name, and messages structure. They treated web service description as a document and considered the user request as a keyword query; and searched for these keywords in the description of the available web services using syntactic methods.

Syntactic matching is a necessary condition in order to discover and invoke a web service but it does not guarantee that there is a semantic agreement between the available service functionality and the user requirements. It may be noted that the WSDL documents do not entail any semantic description of the service; however the presence and the use of the semantic descriptions of web services can assist in improved web service discovery. For this purpose, a lot of efforts are made to semantically enhance the description of a

web service, which leads to the association of external knowledge to web service descriptions. This external knowledge is represented in the form of ontologies. The basic idea behind the association of ontologies to web services description is to annotate the service parameters using concepts from the associated domain ontology and apply logical inferences on service input/output parameters. [33]. Semantic Markup for Web Services (OWL-S) [3] offers ontology of web services that consists of classes and properties for describing web services. This ontological description of web services facilitates the automation of publishing, discovering, and invocation of services [3]. Three main parts of the service ontology are: Service Profile, Service Model and Service Grounding.

- *Service Profile* is used for publishing and discovering web services.
- *Service Model* gives the descriptive information about the operations of web services.
- *Service Grounding* tells how to communicate and invoke a web service via messages.

Many intelligent semantic web discovery models have been developed based on OWL-S description of web service [4, 6, 13, and 44]. Typically, semantic web service discovery models exploit the information available in the service profile by performing logical matching on the signatures (i.e. input/output concepts) and textual similarity matching on specifications (i.e. textual descriptions and web service name). Logical matching is also called ‘service-profile IO-matching” or “service-signature matching” as these logical filters are applied on the web service input/output parameter concepts. All previously developed matchmakers follow a logical approach in which the degree of logical

similarity (i.e. concepts subsumption) between a pair of web services is computed according to the logical-based filter definitions. To provide the consumers with accurate but less number of relevant web services, coarse-grained logical matching is applied at the web service level instead of fine-grained logical matching at individual parameters level; i.e. all pair of I/O concepts should have the same degree of subsumption relations to determine the degree of semantic relationship between two web services. As a result, only those web services are selected that have the exact, general or specific degree of relation for all pair of input/output parameters. Any web service that slightly deviates from the specified criteria would not be retrieved as relevant. It is also claimed that the quality of discovery models can be improved by integrating logic-based approach with syntactic matching [4]. In order to improve the quality of discovery models, service specifications (i.e. textual-description and service name) described by service profile are modeled with non-logic-based techniques, such as content-based information retrieval, to exploit the semantics of service' content using relative frequencies of terms. A vector space model is used to compute specification similarity between a pair of web services; which involves converting a textual description and web service names into a vector of terms, and applying string similarity measures to compute their similarities/differences.

1.3. Problem Description

The problem with the WSDL-based matchmakers is their syntactic matching. Matchmaking succeeds if and only if the query syntactically matches the web service description. The same words with different meanings and the same meaning with

different words both lower the precision and recall of discovery models [8]. For example, if a user is searching for a term “car”, then any web service that contains “vehicle” would not necessarily be retrieved by the syntactic matchmakers due to lack of semantics. Additionally, in OWL-S-based matchmakers, the way researchers have applied logical filters it does not consider any web service that shows partial similarity with the user request. They focused only on those relevant web services that fully satisfied the consumers’ needs. Those retrieved web services either showed exact or fully-subsumed degree of similarity with user request. For example, if some of the web service parameters show EXACT filter and other parameters show SUBSUME filter, then these web services would not ever be retrieved by matchmakers. This approach does not consider partially matched web services. Partially matched web services are those services that show different degrees of logical similarity for each of its parameters. Hence, there is very limited choice for a consumer to select another web service if a selected service fails. Other drawback of this approach is that it does not provide consumers with alternative options in the process of developing a composite web service. This approach always guarantees high precision with poor recall. The key issues that we have figured-out from the previously developed matchmakers are:

- Logical matching at web service level (instead of individual input/output parameter level) tends to improve the precision at the cost of low recall. As a result, a consumer has very limited set of web services for selection.
- Syntactic approaches that are applied on the signatures and specifications of web services, such as input/output parameters, textual-description and service name, also tend to lowered the recall; resulting in a list of few

relevant services for selection. Applying long-text approaches, such as TF/IDF, word co-occurrence, on the web service textual-description of few words is not efficient way to exploits semantics of a web service.

- There is no other option exists if a logical reasoning between a pair of web services fails due to missing relationship in a domain ontology; while the structural parameters of services are same.

1.4. Solution Approach

Our objective is to focus on different aspects of the logical and non-logical matching components in order to enhance the consumers' satisfaction by providing more full/partial relevant web services for selection. This is pursued by taking a hybrid approach whereby we exploit both service's input/output concepts using logical method and service's structure, textual-description, and service name using non-logical methods. The signatures and specification information of a web service is extracted from both WSDL description and OWL-S description. We argue that on their own, service discovery models based on WSDL descriptions and OWL-S description do not provide sufficient information which could help in building efficient matchmakers. For example, discovery models based on web service names only retrieve web services having similar names without considering their textual description, which could be different. Similarly, pure logic-based discovery models compare web service input/output parameters logically without taking into account non-functional information and vice versa. Although textual description provides the specification of the web service, the fact that

two web services having similar textual descriptions does not imply that they have the same input/output parameters. This shows that totally relying on the textual-description of a web service, to discover relevant web service, is not sufficient.

In this thesis we are proposing a S^5 Web Service Matchmaker which exploits semantics and syntax for service specification (i.e. textual-description) and fine-grained logical signature matching. We aim to provide the service matching at both the web service signature and web service specification levels.

For web service discovery process, S^5 Web Service Matchmaker considers the signature and specification of a web service. In signature matching, we apply the logic-based similarity measure for the input/output concepts and Structure Preserving Semantic Matching (SPSM) algorithm for parameters' structure. In specification matching, we propose short sentence semantic structure matching for the textual-description and information retrieval technique for name matching. S^5 Web Service Matchmaker always ensures that two web services are similar if and only if they have similar signatures and similar specifications. The reason we are considering the signatures and specifications of a web service in S^5 Web Service Matchmaker is because the functionality of any service is not only described by its parameters but also the specification (such as textual-description and name) which contains behavioral information. For example, two services take the same inputs i.e. 'Car' and 'Model' and return the output 'Price'. Although the signature of the two services are same but the textual-description will specify the currency of the price that each service returns. This shows that signature and specification

collectively facilitate in differentiate the relevant web services. Considering signatures of web service alone with decrease the effectiveness of discovery model and vice versa.

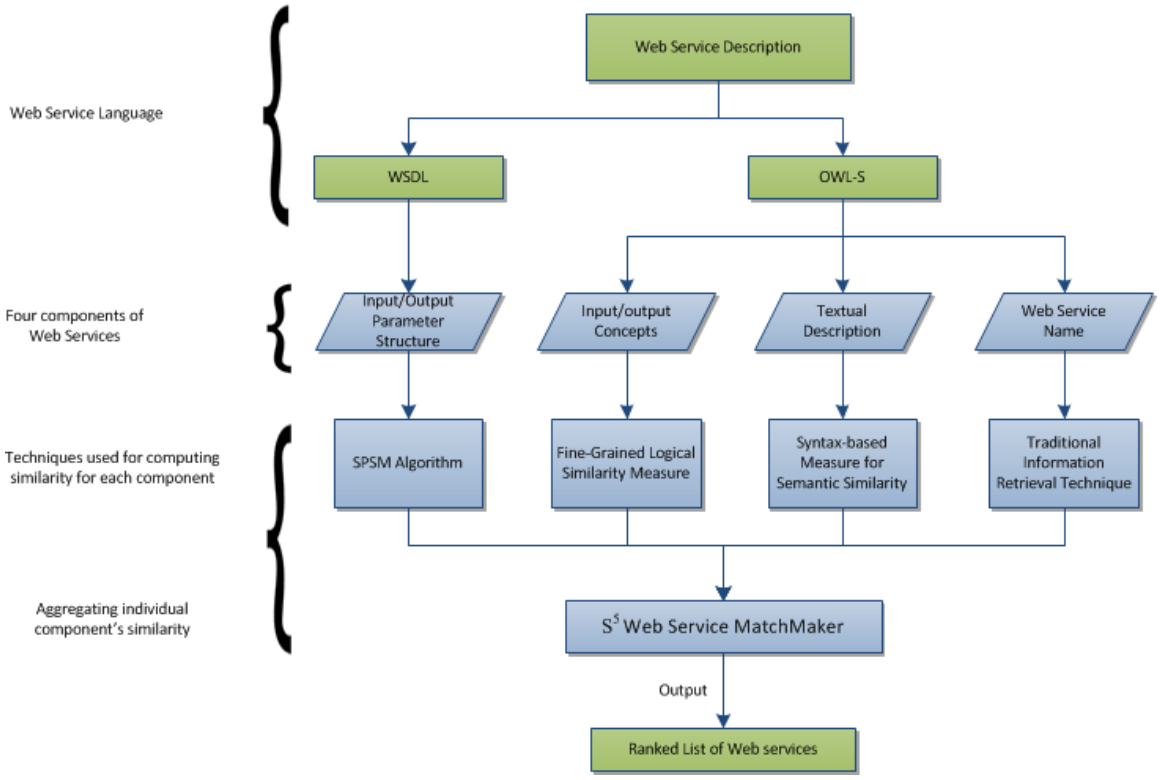

Figure 1.4: System diagram of S^5 Web Service Matchmaker

The process of our hybrid semantic web service discovery model starts from extracting the service's description (i.e. signatures and specifications) from WSDL and OWL-S documents. The individual similarities of the four different components of web services are computed simultaneously and then S^5 Web Service Matchmaker aggregates

individual component's similarity. The functionality of each of the component is described as:

- Logical signature matching is based on the deductive approach which is performed at input/output concepts level. Degree of similarity between any two concepts is computed using five logical filters, as defined earlier. Contrary to previous proposed logical approaches, we perform a fine-grained logical matching at every input/output concept level. In fine-grained logical matching, any web service that partially satisfy consumer needs would also be selected but with a low rank. For example, a web service 'S' having inputs "name", "credit card number" and "pin" would be selected as a candidate service for the requested web service 'R' that requires only "credit card number" and "pin" as inputs.
- Non-logical signature matching exploits implicit semantics, like patterns, sub-graphs, from the input/output parameters' structure. Structure matching determines similarity between two graphs by evaluating the similarity of graph nodes and edges. We use Structure Preserving Semantic Matching (SPSM) algorithm [40] that performs the semantic matching between graph nodes while preserving their structure. It matches the hierarchies of logical concepts which are organized with "is_a" relationship.
- Non-logical syntax-based measure for sentence similarity is employed on the textual-description of a web service. Note that the service textual descriptions contain important information regarding parameters and behavior of a web service, overall functionality, requirements that need to fulfill, and additional information that the service provider wants to share with service consumers. A

textual description of the web service can be exploited to increase the accuracy of web service discovery results. Traditionally, researchers [4, 6, 30, and 44] exploited the web service textual description defined in a service profile as a vector of terms and applied content-based information retrieval techniques. We argue that the symmetric and token-based string similarity measures are suitable for long-text documents but not for short sentences, such as the textual descriptions of services. In long documents, word co-occurrences is an important source of information; but in a short sentence having less number of words, word co-occurrence is not a significant factor. Yet, syntactic structural information that can be extracted from the short sentences provides more significant information and helps in calculating sentence semantic similarity. We are using syntax-based measure for short sentence similarity approach [29] to exploits both the syntax and semantics of textual description of web services.

- Non-logical service name matching involves the fine decomposition of a name into a vector of meaningful terms. For a service discovery, every service is usually identified by its name. Web service name could be a combination of words which may sometime contain important terms, reflecting web service parameters or behavior. We define our decomposing rules to split a service name to a vector of meaningful terms and then apply text similarity measures.

S^5 Web Service Matchmaker aggregates the individual similarities of all four components and retrieves only relevant web services that are above the similarity 'threshold' which is specified by the web service consumer. It is expected that there would be an increase in discovering more relevant web services, hence results in

more consumer satisfaction by providing more service selection options in case of web service failure and web service composition. The increase in relevant web service is because of three main reasons. First, the fine-grained logical matching discovers the partially matched web services to satisfy consumer needs and facilitate web service composition process. Second, if the score of logical-matching decreases, it would be compensated by parameter structure matching, resulting in an increase of discovering more relevant web services. The last main reason is due to syntax-based semantic sentence matching which helps in the refining irrelevant web services; as a result improve the effectiveness of retrieved web services.

1.5 Evaluation

The Semantic Web service discovery is a process of retrieving relevant web services based on their semantic descriptions. All the developed semantic web service discovery models are evaluated based on their retrieval performance where a model retrieves the matched services with their degree of similarity. The problem of web service discovery is treated as a special Information Retrieval (IR) problem, Hence evaluation of all semantic web service discovery models is performed using the standard IR measures (i.e. precision and recall). The test collection (i.e. OWLS-TC) used in evaluation of semantic models comprises of three components: a set of web service (the test data), a list of queries, and relevant web services for each query.

For the evaluation of S^5 Web Service Matchmaker, we have also used OWLS-TC [16], which is a standard web services retrieval test collection. This collection provides 1000 web services, described in WSDL and OWLS standards. Services are categorized into seven different domains i.e. education, medical care, food, travel, communication, economy and weapons.

To evaluate the performance of our model, we used OWLS-MX as a benchmark and used precision, recall, and accuracy as performance measures.

1. First, we evaluated the performance of pure logical component and impact of our fine-grained logical approach on web service discovery model. Our proposed logical approach successfully improved the overall accuracy of model by 2% with an increase of 43% recall.
2. Second, we evaluated the change in performance of our model by integrating both the signature and specification of a web service (i.e. logical, structural, textual-description, and service-name similarity).
3. We then compared our hybrid model with hybrid OWLS-MX, which showed an accuracy improvement by 1% with an increase of 32% recall.
4. Finally, we showed the impact of exploiting the structure of a sentence for textual-description matching. We compared the results of syntax-based textual description matching with traditional information retrieval techniques and achieved an improvement of 3% accuracy with an increase of 15% precision and 8% recall.

1.6 Workings of our Approach

S^5 Web Service Matchmaker takes web service consumer requests, extracts information and performs similarities, and returns a list of web services, as shown in Figure 1.5. The workflow for our approach is described as follows:

1. Web service discovery process is initiated by a consumer by providing an OWL-S description of a web service to be searched. We have developed a Query interface for the web service consumer to provide his request.
2. Preprocessing step involves the process of extraction of four major components of web services that are input/output concepts, their structure, textual description and web service name. These components are then sent to matchmaker for comparison.
3. After preprocessing, extracted information is sent to web service matchmaker to find similarity between the corresponding components of available web services.
    a. Logical matching would be performed between input/output concept of the query web service and all available web services using domain ontology.
    b. Structural matching between the structures of web services would be achieved by using an SPSM algorithm.
    c. Textual description matching is done using short sentence semantic matching approach.
    d. Web service names are broken into meaningful words and semantic matching is performed to compute similarity between web services.

4. S^5 Web Service Matchmaker computes the cumulative similarity of all four components and returns the list of ranked web services to the consumer. This list of services is ranked based on their similarity value in descending order. Matchmaker will also allow the consumers to select a 'Top N service' option that he is interested, where N is the number of retrieved web services.

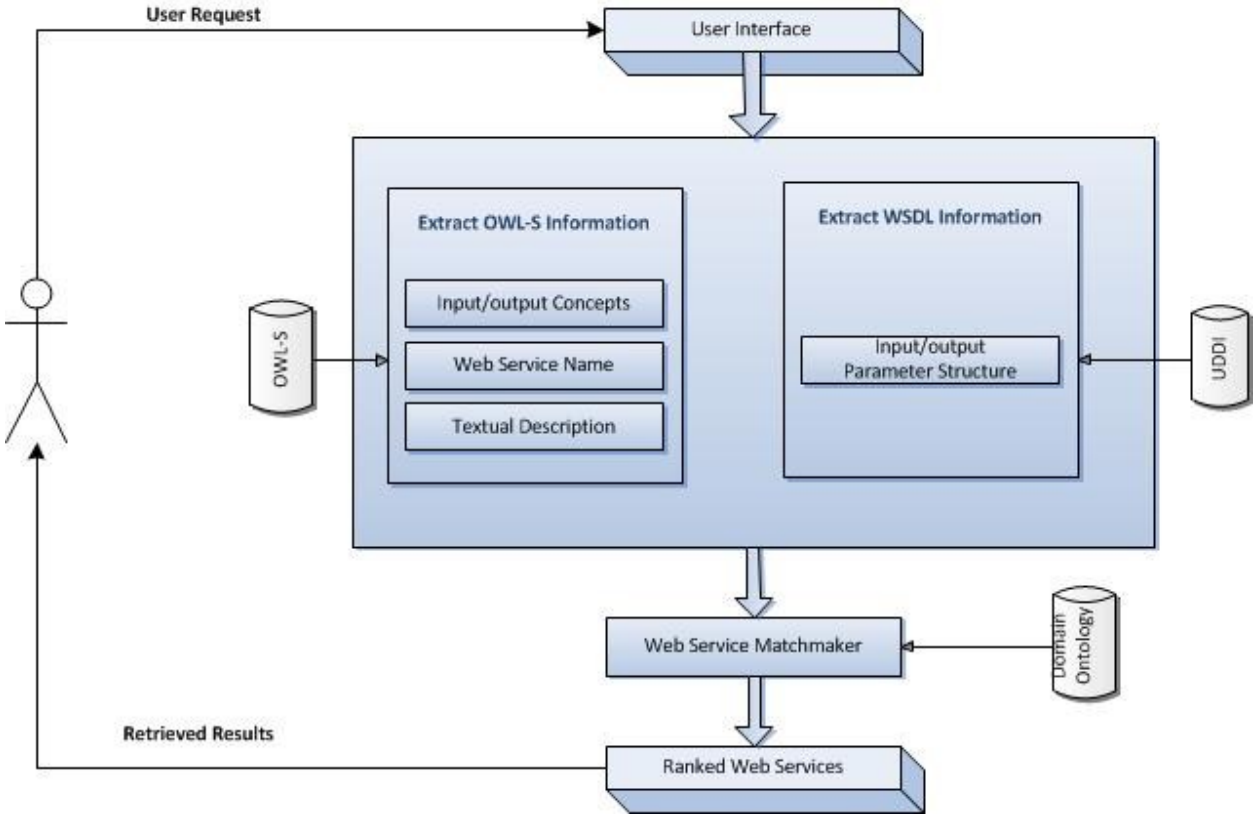

Figure 1.5: Workflow of S^5 Web Service Matchmaker.

1.7 Thesis Contributions

We propose a hybrid web service discovery model that performs the logic-based reasoning on the logical signatures and non-logic-based approach on the both signatures and specifications of a web service. We aim to provide consumers with a list of full/partial similar relevant web services that could help him in two ways: a selection of alternate web service if a selected web service fails, and composition of multiple web services, that partially satisfy consumer needs, into a single web service. To achieve this goal, the three main contributions of the S^5 Web Service Matchmaker are:

1. We propose a fine-grained logical signature matching approach, in which a pure logical similarity between a pair of web services is computed based on each individual service's parameter separately. This fine-grained logical signature matching will always ensure that any partially similar web service may satisfy consumer needs and helps him in web service composition process. This approach results in selecting even those candidate web services which partially satisfy consumer needs. The increase in the number of relevant web services benefits the consumer in two different ways. First, a consumer has many options to select an alternate web service in case a selected web service fails. Second, many partially matched services helps the consumer in developing composite web service to achieve his desired requirements.
2. For syntax-based measure for semantic similarity of web service textual descriptions, we present a grammatical approach for analyzing the short textual descriptions of services. In our work, instead of decomposing a service description into a vector of terms, we exploit the syntax of a textual description

and split the textual description into subject, verb and object using syntactic functions [25] and then apply semantic similarity measures on similar syntactic functions. We argue that instead of calculating the term frequencies for a sentence comprising few words, matching the syntactic structures is more practical and our results demonstrate that the approach works well for matching service textual descriptions. Syntax-based sentence similarity measure helps in refinement of the irrelevant services based on the web service specification, which results in increase in precision of retrieved web services.

3. We not only consider the logical similarity of service's parameters for signature matching but also consider the structure of the signatures (i.e. input/output parameters). To make the discovery more effective, logical signature matching is further coupled with the parameters structure matching whereby we use Structure Preserving Semantic Matching (SPSM) algorithm [28] to semantically compute the structural similarity between a parameters' structure. This approach power the logical matching by contribution in signature matching. In any case where the logical matching fails, structure matching play its role and helps in selecting relevant web services.

1.8 Thesis Organization

This thesis is organized as follows. Chapter 2 gives the background knowledge of semantic web services with its standards, technologies facilitate semantic web and efforts done in the area of semantic web service discovery that provide spirit to our research

work. Chapter 3 describes our approach adopted for semantic discovery of four components of web services illustrated with examples and pseudo codes. We present a series of experiments and evaluation techniques in Chapter 4. At last, the thesis presents conclusion of our proposed hybrid approach and future work of our method in Chapter 5.

Chapter 2. Background and Literature Review

2.1 Web Services

The World Wide Web is a system of global databases and applications that could be accessed via internet. It is widely used for one to one communication between applications. Web Services are designed to support communication between systems over the internet [1]. It is based on the open standards, such as Hypertext Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI). Based on these open standard technologies, applications written in different languages, running on different platforms and different operating systems can exchange data using web services. The core technologies of web services, i.e. WSDL, SOAP and UDDI, are XML-based. A WSDL document provides an interface, which describes the location and functionality of web services and hides the implementation of the program from the consumer. Information from the WSDL document is then published on the UDDI registry. The UDDI registry helps web service consumers to search and locate the services that satisfied their requirements. Once the web service is discovered, the service consumer constructs SOAP messages to communicate with a service using HTTP protocol.

2.1.1 The Web Services Model

The basic underlying data model of a web service is based on XML that represents the data in a standard format to make it possible to understand by any other application. Based on XML, SOAP protocol is used for message transmission, WSDL to describe web service interfaces, and UDDI to publish web services. A Web Service Architecture is realized through an interaction between three main roles: (i) Service Provider/publisher, (ii) Service Requestor and (iii) Service Broker.

i. A *service provider/publisher* is the creator/owner of a web service who has privileges to remove, change or update the web service without informing web service users.
ii. *A service requestor/consumer* is the one who wishes to use the functionality of a web service.
iii. A web *service broker* or web service registry is the middle layer that connects service requestors with the service providers. Service providers register web services on the registry and service consumers search for the services in that registry.

The three main operations of a web service architecture are: (i) *publish,* by which service providers register their services in the service registry, like UDDI, to make it publically available for everyone; (ii) *find/discover,* by which a service requester searches for a

service in the registry that satisfies their requirements; and (iii) *bind,* by which, after selecting a web service from the registry, a service requestor invokes the service using the binding information available in the web service description (as shown in Figure 2.1).

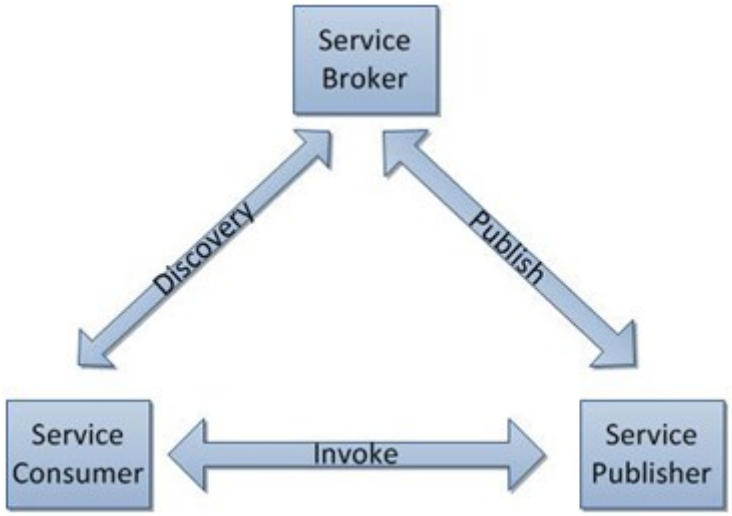

Figure 2.1: Web service data flow model

2.2 Web Service Technologies

2.2.1 XML

XML (eXtensible Markup Language), a W3C recommendation, is a flexible and simple data format, designed to support the exchange of data over the internet [31]. It represents the data in a structured way so that it is easy to understand by human as well as machine. It is the core of all the technologies used for web services, such as WSDL, SOAP and UDDI, as shown in Figure 2.2.

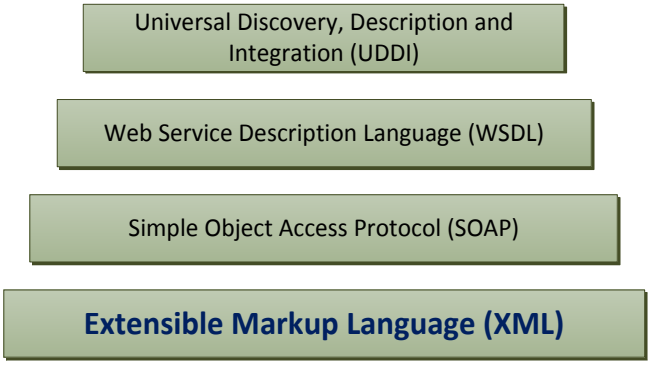

Figure 2.2: Web services stack technologies.

XML is not the replacement for HTML. HTML is used to display the data to the user while XML is used to represent the data in a structured format. The user can create their own tags to represent the data, which is contrary to HTML where the user has to use predefined tags. XML simplifies the data sharing by storing the data in the plain text format, which ease the sharing of data between different applications. An example XML document is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Shopping>
        <Price>
                <amount>10</amount>
                <currency>$</currency>
        </Price>
</Shopping>
```

These are not the XML standard tags but are defined by a user to store the data for amount and currency. This representation makes it understandable for machine as well as human. Nowadays, XML has become the backbone of the web. It is the building block

for many other languages to transmit data in a structural format between different representational frameworks, such as eXtensible HyperText Markup Language (XHTML), Web Service Description Language (WSDL), Wireless Application Protocol (WAP), Resource Description Framework (RDF), and Web Ontology Language (OWL).

2.2.2 SOAP

Simple Object Access Protocol, SOAP, is an XML-based light weighted protocol that aims to provide the exchange of structured information between independent decentralized distributed environments [32]. SOAP is platform independent as it is meant for exchanging messages between web services. It defines the format for sending messages between different applications. The two main goals of SOAP are *simplicity* and *extensibility,* which are achieved by using XML [32]. Its extensible message framework provides a mechanism to encode data into message construct so that it could be exchanged independently over a variety of protocols. An example SOAP message is shown below:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n: alertcontrol xmlns:n="http://example.org/alertcontrol">
    <n: expires>2001-06-22T14:00:00-05:00</n: expires>
  </n: alertcontrol>
 </env:Header>
 <env:Body>
                <m: alert xmlns:m="http://example.org/alert">
                        <m: msg>return price of a car</m:msg>
                </m: alert>
                <SOAP-ENV:Fault>
                        <faultcode>SOAP-ENV:Client</faultcode>
                        <faultstring >Failed to locate method</faultstring>
                </SOAP-ENV:Fault>
 </env:Body>
</env:Envelope>
```

A SOAP message is comprised of the following elements:

- *Envelop*: It is the outermost or root element of a SOAP message and contains the information for the receivers to know where the message starts and where it ends. It contains two elements: *header* and *body*.
- *Header*: This is an optional element in a SOAP message, used to specify additional application-level information, like digital signature or encoding of the data. There could be zero or more header blocks within a SOAP message.
- *Body*: It contains the collection of zero or more actual SOAP messages and is required to be received at the other end (recipient). It is contained within an envelope and must be defined by a header.
- *Fault*: Error handling is done by using the *fault* attribute of a SOAP message. If any error occurs during processing, SOAP fault mechanism returns a predefined error code back to the sender. It is defined only once within the *body* element.

SOAP is not tied to any particular transport protocol. SOAP requests/responses are sent and received via HTTP request/response.

2.2.3 UDDI

Universal Description Discovery and Integration (UDDI), proposed by Microsoft, is an XML-based electronic registry for web services, which helps the businesses to register and locate their web service applications [3]. UDDI provides a framework based on XML

and SOAP standards to facilitate publishing, discovering and managing web services over the internet. A Web service interface, represented in WSDL, is published on UDDI. Businesses create and publish their services on UDDI to make them available for public usage. Web service consumers search and discover the web services on UDDI that satisfy their requirements and invoke these web services using their metadata. The UDDI registry does not support semantic description of web services. Since the web services are described as WSDL, which is an XML-based document, no semantic description of a web service is available.

2.2.4 WSDL

Web Service Description Language (WSDL) is an XML-based description language that provides a simple interface to a web service [14]. It allows the applications that are running on distributed environments to independently communicate with each other using their interfaces. A WSDL interface is comprised of input/output messages grouped into operations. Each operation corresponds to a function in an application. A WSDL interface may contain one or more operations. It also provides the binding information for each interface. In short, a WSDL contains all the information which could be necessary for a consumer to invoke a web service. Each operation in a WSDL document defines the abstract description of messages that need to be exchanged using SOAP protocol. Here is the WSDL example that shows the operation “1personbicycle4wheeledcarPrice” with its messages.

```
<wsdl:message name="get_PRICEResponse">
        <wsdl:part name="_PRICE" type="string"></wsdl:part>
```

```
</wsdl:message>
<wsdl:message name="get_PRICERequest">
        <wsdl:part name="_4WHEELEDCAR" type="string"></wsdl:part>
</wsdl:message>
<wsdl:portType name="1personbicycle4wheeledcarPriceSoap">
        <wsdl:operation name="get_PRICE">
                <wsdl:input message="tns:get_PRICERequest"/>
                <wsdl:output message="tns:get_PRICEResponse"/>
        </wsdl:operation>
</wsdl:portType>
<wsdl:binding                                    name="1personbicycle4wheeledcarPriceSoapBinding"
type="1personbicycle4wheeledcarPriceSoap">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="get_PRICE">
      <wsdlsoap:operation soapAction=""/>
        <wsdl:input>
                 <wsdlsoap:body          use="encoded"          encodingStyle          =
                http://schemas.xmlsoap.org/soap/encoding/          namespace          =
                "http://127.0.0.1/wsdl/Onepersonbicycle4wheeledcarPrice"/>
        </wsdl:input>
        <wsdl:output>
                <wsdlsoap:body                                                  use="encoded"
                encodingStyle="htp://schemas.xmlsoap.org/soap/encoding/"     namespace     =
                "http://127.0.0.1/wsdl/Onepersonbicycle4wheeledcarPrice"/>
        </wsdl:output>
        </wsdl:operation>
</wsdl:binding>
<wsdl:service name="1personbicycle4wheeledcarPriceService">
         <wsdl:port                                    name="1personbicycle4wheeledcarPriceSoap"
binding="1personbicycle4wheeledcarPriceSoapBinding">
                <wsdlsoap:address
location="http://127.0.0.1/wsdl/Onepersonbicycle4wheeledcarPrice"/>
         </wsdl:port>
</wsdl:service>
```

A WSDL document describes a web service using five major components:

- <*types*> element contains data type definitions that are used for exchange of messages.
- <*messages*> element is an abstract representation of the transmitted information. Typically, a message contains one or more logical parts (i.e. parameters). These parts are associated with a type definition.
- <*portType*> is an important component in WSDL documents in which a set of abstract operations (i.e. functions) that can be performed by the web service are defined. Each operation is associated with an input and/or output message.

- <*binding*> component specifies a communication protocol and data format for each operation and message defined in a particular portType element.
- <*service*> element is a composite operation that aggregates multiple related ports or functions.

2.3 Semantic Web

The web is a collection of network-accessible information that includes document and other resources linked via hyperlink or URL. These documents and other resources are solely meant for the human to read, not for the computer to manipulate. By contrast, semantic web adds structure and meaning to the content available on the web to make it understandable for computers and humans [5]. By making contents more meaningful, the intelligent and sophisticated approaches can be developed that can help interaction between web pages, services, and other programs. Semantic web service is the most important application of semantic web which adds semantics to the web services descriptions. As a result, it facilitates the automation of web service discovery, composition, invocation, and interoperation [4].

XML defines the organization of data but it does not explain the actual meaning of that data. Semantic web is the solution to this. All Semantic web technologies are built on top of XML. There are many semantic web technologies and standards, like RDF, RDFS, and OWL, which provide metadata information to XML data to make it more meaningful for the computer systems.

2.3.1 RDF

Resource Description Framework (RDF) [22], a fundamental component in the semantic web, is built on top of XML and stands as a standard model for web data interchange [1]. The related data on the web is represented as triples in RDF. Each triple has the form <subject, predicate,object>. This way of describing data makes it easy for machine to understand and manipulate. These triples are identified by URI. RDF uses URI to represent data to give them a unique definition and make them accessible on the Web [5]. Along with the RDF, RDF Working Group developed the RDF schema (RDFS), which provides the framework to describe the classes and their properties.

2.3.2 OWL

The Web Ontology Language (OWL) is built on top of XML, RDF and RDFS. It is designed to bring more expressiveness to data on the web, in order to assist machines to perform reasoning tasks automatically and integrates information. It is W3C standard and aims to promote the interoperability and sharing between linked-data. OWL is an extension of RDF(S), which provides additional vocabulary for the RDF(S) classes and properties, such as disjointness, cardinalities, equalities between the classes and symmetric properties relationships. Based on the varied expressiveness level, OWL is classified into three sub-languages, as shown in Figure 2.3.

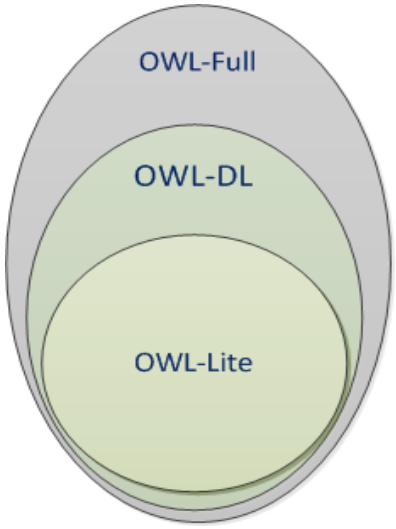

Figure 2.3: OWL Classification.

- *OWL Lite:* OWL Lite is a simple extension of RDF with addition of simple constraints on concepts and their relationships [34]. It provides limited expressiveness to the users, for example, it supports cardinality value of 0 or 1.
- *OWL-DL:* OWL-DL ensures computational completeness and decidability by extending OWL Lite up to the decidable fragments [24].
- *OWL-Full:* OWL DL and OWL Full share the same ontology language constructs but the only difference is usages of restrictions and RDF features. It allows free integration of RDF schemas while, in OWL-DL, it puts restrictions, like disjointness of classes, properties and individuals.

The selection between the OWL-Lite and OWL-DL depends on the expressiveness required by the user. On the other hand, selection between OWL-DL and OWL-Full

depends on the extent to which a user required RDF-S meta-modeling facility, for example defining subclasses or attaching properties to the classes [34].

2.3.3 OWL-S

The Semantic Markup for Web Service (OWL-S) enables greater access to the web service to power the automation of web service tasks i.e. publishing, discovering, composition, invocation, and monitoring [3]. OWL-S is ontology for web services that allows the service providers to create and publish their web services with much more capabilities and properties. This, in turns, makes it more machine readable and increase automation of web service tasks. The three main parts of the service ontology are: (i) *Service Profile*, (ii) *Service Model* and (iii) *Service Grounding*, as shown in Figure 2.4.

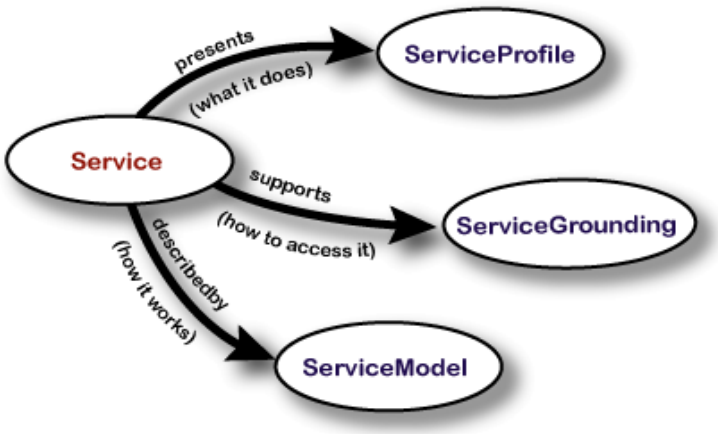

Figure 2.4: Web service ontology components.

i. *Service Profile:* It tells what functionalities a web service provides to the service consumers. Service profile is used for the advertisement of web services. It represents the service name, service textual description, input/output parameters, and publisher contact information. It helps the web service seekers to determine whether the service fulfills their requirements.

ii. *Service Model:* It tells how a web service consumer can use this web service. Service model describes information related to the web service, such as semantic description of input/output parameters, preconditions and effects. It also provides in-depth information to a web service seeker in discovering relevant services.

iii. *Service Grounding:* the detailed information of how to access a web service using SOAP messages is provided by Service Grounding. It defines consumer communication protocols, port type, and message formats.

2.4 Domain Ontology

With the exponential increase of online web service, manually performed web service operations, such as web service discovery, composition and integration, are becoming difficult. The emergence of the semantic web service technology facilitates the automation of the web service lifecycle model. OWL-S provides constructs to combine two kinds of ontologies in a web service description.

1) Generic web service description ontology that describes the semantics of web services, such as inputs, outputs, category.

2) Domain ontology that provides external knowledge to web services data like a "person" data type in web service description can be combined to a concept in domain ontology.

Domain ontology is an ontology that is designed for a specific domain to provide semantic descriptions of web services. Reasoning tasks are performed based on these semantic web service descriptions. The quality of reasoning tasks directly depends on the quality of domain ontologies. For complex reasoning, the domain ontology should be rich enough to cover maximum knowledge for that domain.

2.5 Short Sentence Matching

Most of the models described above processed the textual description to determine the web service similarity. A Textual description of web service is a short sentence which is provided by a publisher to describe the behavior of a web service. Textual description contains useful information about the web service functionality that needs to be exploited efficiently. Previous approaches considered textual description as a bag of words and applied traditional information retrieval techniques, such as cosine similarity, extended jaccard. Oliva [29] proposed a method for determining the semantic similarity between a pair of short sentences. Semantics of a sentence is not only based on the meaning of its individual words but also the structural way the words are combined. This approach exploits the syntactic and semantic information from a sentence to compute the semantic similarity between two sentences. Syntactic information extraction is a process of

obtaining the structure of sentence through deep parsing. Semantic information matching is the process of computing the semantic relationship between the concepts that have same syntactic roles. The overall process is split into two sub steps:

1. Given two sentences, the process starts by parsing the structure of both sentences, as shown in Figure 2.5. The result of parsing sentence is a dependency structure which contains syntactic functions.

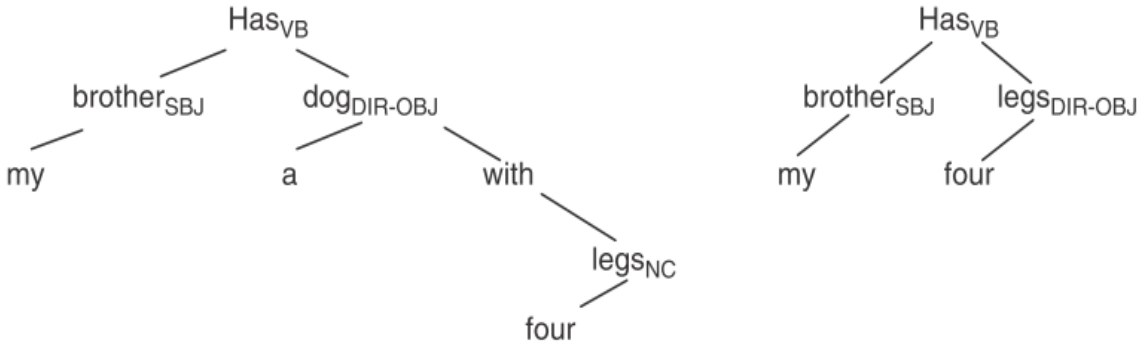

Figure 2.5: Parse tree of two sentences.

2. Once the syntactic information is extracted, the semantic similarity for two sentences is computed by semantically computing the similarity between same syntactic functions. The similarity between two words is computing using LIN similarity measure [38] which is based on WordNet.

2.5.1 Example

Figure 2.5 shows the structure of two sentences. In this case, the LIN similarity between principal verbs that is “has” and “has” is 1.the heads of subject “brother” and “brother” gets 1. While LIN similarity between the heads of direct object that is “dog” and “leg” is 0.1414. As a result the overall similarity is 1/3(1+1+0.1414) = 0.71. However, in this case, other string similarity measures would give it more similarity because of the presences of words “leg” and “four” in both sentences.

2.6 SPSM Algorithm

Structure matching is the process of comparing the semantic relationship between the nodes of two graphs. In some cases, it is required to compute the similarity between the identical elements of two structures. An S-Match [28] is a semantic matching framework that provides several matching algorithms. Structure Preserving Semantic Matching (SPSM) algorithm [40] is one of them. SPSM takes a tree-like structure as an input, generates concepts of labels and concepts of nodes, and then computes a semantic similarity between those concepts. The relationship that could exist between two concepts is: *equivalent*, *general*, *specific*, or *mismatch*. Every node in a tree-like graph has a number and a label. Nodes are uniquely identified by numbers, while labels are used to

represent concepts. The preprocessing module in SPSM takes an input tree and returns an enriched tree that contains concepts of labels and concepts of node. The generation of concepts of labels and nodes involves external knowledge that is WordNet. This enriched tree is then entered in a matching module, which computes the similarity relationship between all pairs of label and node's concepts of both trees. The result would be the overall similarity between the two trees, which ranges between 0-1, where '0' represents no match and '1' represents an exact match.

2.7 LIN Similarity Measure

As mentioned in Oliva [29], the Information Content (IC) measures the specificity of a concept, which is higher for more specific concepts. Several proposed similarity measures used IC to find the commonality between two concepts. LIN extends the IC measure and proposes a method to compute the similarity between two concepts [38]. This method is powered by WordNet. If there is a path between two concepts in WordNet, LIN's measure returns their relatedness score. LIN measure can be defined as "the ratio of the information content of the lowest common subsumer to the information content of each of the concepts" [29]:

$$sim_{lin}(c_1, c_2) = \frac{2 * IC(lcs(c_1, c_2))}{IC(c_1) + IC(c_2)}$$

Where, C_1 and C_2 are the two concepts and Sim_{lin} is the similarity between C_1 and C_2.

2.8 Literature Review of service discovery

As described above, web services are the self-describing application component that can be used by any other application using open protocol. The components of web service description which facilitate the web service discovery model are usually categorized into two: web service signature and web service specification [11]. Web service signatures are the input/output parameters that directly correspond to the function's parameters. Web service signature does not describe the behavior of web services. On the other hand, web service specifications are the behavioral description of web services that are explicitly defined by the web service publisher, such as textual description, category, and business. Web services are described by Web Service Description Language (WSDL) and Web Service Ontology (OWL-S). WSDL document contains input/output messages, which are abstract definitions of data, and port types, which are abstract collection of operations [14]. WSDL is published on the UDDI, where a web service consumer comes and discovers the required service based on the available descriptions. UDDI APIs also allows the web service publisher to provide web services specifications. The WSDL document, which is also called non-semantic web services, is more popular and easily adopted by industry and development tools.

Many matchmaker tools are developed that exploit the web service's signatures and specification information to discover relevant web services based on user requests. Early developed WSDL document matchmakers adopted information retrieval techniques to discover relevant web services for user requests. These matchmakers extract the signature information from WSDL documents published on UDDI. UDDI supports the key word

search; That is, for a keyword it matches the web service name, location, business, category to retrieve results. If there is no exact match for a keyword, no web service would be retrieved. For example, if the query keyword contains “car”, UDDI would not retrieve web services that contain term “vehicle”. The lack of semantics in web service description’s matching result in low accuracy.

2.8.1 WSDL Signature Matching Approaches

Elgazzar [9] proposed an approach that exploits the WSDL document information and group them into clusters based on similar functional web services. The keyword-based search engine for non-semantic web services matches web service name, location, and business information from the WSDL documents and retrieves results. Elgazzar proposed an approach based on the five features of WSDL document: WSDL content, WSDL types, WSDL messages, WSDL ports and web service name. Information retrieval techniques are applied on all these five features to calculate web service similarity as shown in Figure 2.6.

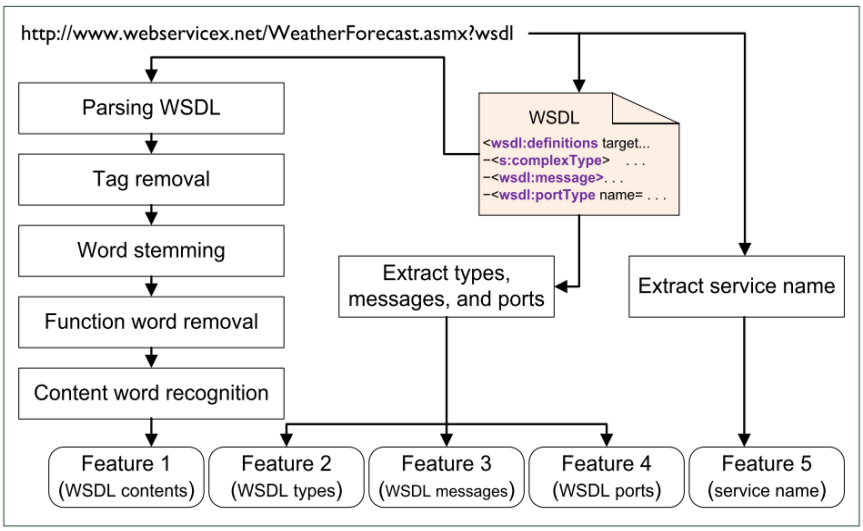

Figure 2.6: Block diagram of the features extraction process.

Elgazzar's [9] main contribution was to propose an approach that found the web service similarity based on the five features and clustered based on the functionality. All five features determine the semantics and behavior of a web service. The extraction process of features from the WSDL document is mentioned as:

- Feature-1 WSDL Content: they start by reading the content of a WSDL document using WSDL URI and converted it into a vector of terms after preprocessing. The preprocessing involves:
    - Parsing WSDL: vector of token are extract by splitting the WSDL documents based on space.
    - Tag Removal: tokens that are parts of XML tags are removed from the vector.
    - Word Stemming: all tokens in a vector are stemmed to their base words.

    - Function Word Removal: the function words are removed from the vectors of token.
    - Content Word Recognition: WSDL documents contain certain specific words like ‘data’, ‘web’, ‘port’ that does not contain any semantic information. These words are removed from token vectors.

- Feature-2 WSDL Types: WSDL document contains input/output identifiers that are used by messages. These identifiers are defined by label and their types. They could be a simple or complex type. The similarity between the two web services is performed by comparing the numbers of data type matches.
- Feature-3 Messages: WSDL messages contain one or more message parts. These message parts correspond to a simple or complex data. The number of similar matches between messages determines the similarity between two web services.
- Feature-4 WSDL Ports: port type in WSDL document defines the combination of messages for an operation. The number of same port types with the same messages gives the similarity between two web services.
- Feature-5 Web Service Name: a web service name is normally a composite word, such as ‘WeatherForecast’. Elgazzar [9] decomposed a web service name base on capital letters. The similarity between two web service names is computed using Normalized Google Distance (NGD).

The overall similarity between two web service descriptions is computed by averaging the similarities of all five features.

Problem: Elgazzar's approach [9] suffers from a lack of semantic matching between identifiers. It only considers an identifier's data type matching without considering its structure. He also decomposed web service names only based on capital letters. Whereas, a web service name could contain numbers, underscore, and dash. They computed the similarity between two words without considering their semantic similarity.

Lui [10] proposed a similarity measure to determine the web service similarity with more relatedness. As mentioned in [10], it is not sufficient to apply traditional IR techniques on WSDL document content as it contains a very little text, which is contrary to [9]. The motive behind their proposal is that terms within documents are not isolated. There is some kind of semantic relationship between terms which could help in finding a similarity between web services. Lui [10] also considered the ports, port types, input/output messages, operations, and other definitions. Extracted information is converted into a vector of terms. The approach adopted in [10] is divided into two steps. First the similarity between the words is calculated using web search instead of WordNet database because there could be some terms which are not included in WordNet such as "mp3", and "wordnet". Secondly, they propose a different similarity metric that computes the similarity between two different terms based on their semantic relatedness. Each term in a vector represents a node and the edge between two terms is represented by a semantic similarity score. For example, two vectors of terms (nodes) and weights between their terms are shown in Figure 2.7.

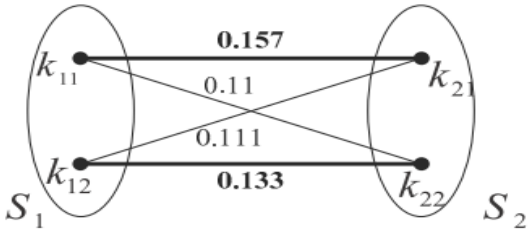

Figure 2.7: Two web services as a bipartite graph.

Once the semantic distance is computed between each term of both vectors, the maximal weight matching of bipartite graphs is calculated. This maximum value between a pair of bipartite graphs is the total similarity between two web services.

Problem: In this approach the researchers did not consider the structure of the WSDL document into their measure.

Dong [18] proposed a web service discovery model very similar approach to [9, 10]. They proposed a search engine that combines evidence from multiple sources and calculated the similarities. These sources of evidence are the service name, input/output parameters, and operations of a web service described in WSDL. The similarity between these source of evidence is done by applying syntactic techniques i.e. TF/IDF. The main point of this search engine is to cluster web services based on the parameter names.

Nayak [17] proposed the idea of applying data mining techniques on the web services as the quality of data represented in WSDL is much finer than any other data. There are some hidden data which are rarely exploited. Like, in the case of web services, keywords

that represent a service can play an important role in selection of that web service. Data mining techniques can be used to explore this hidden information from the web service logs. For every user query, a log file of search term is maintained for that session. Later on, the user would be provided with the suggested words while searching for a web service. This suggested word list is collected from the previous searched sessions stored in the query logs. The search session similarity is computes by using Jaccard coefficient similarity measure between searched terms of web services.

2.8.2 WSDL Signature and Specification Matching Approach

To improve the web service discovery, Wang [11] proposed a method that exploits semantics of the WSDL description's identifiers and the structure of their operations, messages and data types. Their proposed method expects the user to provide a WSDL document and a textual specification of the desired web services. In the first step they process WSDL specification of all available published web services on UDDI. Web services specifications that match with the textual description of user request are selected for further filter. Structure of all retrieved web services are now compared with requested web service. This structure matching algorithm is powered by WordNet. Semantic distance calculation between identifiers and data type of WSDL structure refines and accesses the candidates web services. The overall steps are described as follow:

- Step 1: user will provide the textual description of a web service. This textual description is compared with the textual description of all available web services which were provided by a publisher in WSDL specifications. A textual

description is specified in natural language that can be exploited by using traditional information retrieval methods such as vector space model. It is converted into a vector of terms. These terms are refined by stemming, stop word removal, and weighing each term. The weight of a term is product of term frequency and inverse document frequency.

$$w_{ij} = tf_{ij}\ idf_i = tf_{ij}\log_2 (N/df_i)$$

After obtaining the vector of terms with their weight, they applied cosine similarity which shows the similarity between two web services.

- Step 2: Retrieved set of web services from step 1 are not pruned down based on their structural similarity. They proposed a method which compared the structures of two web services using WordNet. The structural similarity between two web services is determined by semantically comparing their operations, which in turn is based on the semantic comparison of the input/output messages of each operation, which is again based on the data types of the messages identifiers. They also performed the structural comparison between the labels of the identifier instead of their data type. WordNet is used for semantically comparing the label of identifiers in structure matching. The overall structural similarity is the average of structure data type matching and structure identifier matching.

Problem: we find a number of problems in their web service discovery model:

1. They applied traditional information retrieval technique that is tf.idf for few words of textual description. As already mentioned in [9,10,29] that applying traditional IR techniques like TF/IDF is not good for a sentence of few words as

this technique rely on term frequencies to determine the semantics of web services.

2. The structure matching algorithm did not consider what identifiers are matching. It just consider their types without considering their semantics and preserving structure, such as string matched to string, float matched to float.

2.8.3 OWL-S Signature Matching Approaches

The WSDL document does not provide the semantic information about the web services. There is no notion of relationship between the parameters of a web service. The emergence of semantic web and integration of semantic web with web services open a new door for the researcher with new challenges. To solve the problem, many semantic web service matchmaker engines are developed. Klusch [4] described the first hybrid approach, called OWLS-MX, for the semantic web service discovery based on the OWL-S, which is the combination of logic-based semantic similarity of OWL-S services and content based syntactic similarity. In this model, the services advertised and user request are described in same description language, i.e. OWL-S. OWL-S facilitates the standard usage of description logic reasoning to automatically determine the web service similarity based on their concepts subsumption relations. OWLS-MX performs logical input/output concepts matching. The process of content-based and logic-based similarity is described as follow:

- Content-Based Similarity: First, their model processes OWL-S web service description of all available web services in their database (OWLS-TC). Web

service description includes web service name, textual description and unfolded input/output concepts. These web service descriptions are converted into a vector of terms along their weights, after stemming and stop word removal. They apply token-based string similarity measures to compute the content similarity between query and available web services. These measures are cosine, extended jaccard, loss of information and Jensen-shannon information divergence.

- Logic-Based Semantic Filters: OWLS-MX determines the semantic similarity between two pair of services by defining five logical filters that are EXACT, PLUG-IN, SUBSUMES, SUBSUMES-BY and FAIL. If there is service S and request service R then the logical filters would be defined as:
    - *Exact Match*: Service S perfectly matches with service R if and only if all parameter of S exactly matches with any parameters of R.
    - *Plug-in Match*: Service S plug-in matches with Service R if and only if all parameters of S is least specific concept of parameters of R and number of service S parameters should be greater or equal to number of service R parameters.
    - *Subsumes Match*: Service S subsumes service R if and only if all parameters of S are sub-concepts of parameters of R and number of service S parameters should be greater or equal to number of service R parameters. This filter is weaker than plug-in match.
    - *Subsumed-by Match*: Service S subsumed-by service R if and only if all parameters of S are least general concept of R parameters and number of

service S parameters should be greater or equal to number of service R parameters.

- *Logical Fail*: Service S is failed to match with service R if none of the above filter matches.

OWLS-MX first performed the logical matching between the pair of web services parameters. If the web services failed to match logical, then OWLS-MX match performed the content based similarity. OWLS-MX implemented the variant algorithms. Each algorithm is the combination of logical-based semantic filter and one of the content-based filter that are cosine, extended jaccard, Jensen-shannon information. According OWLS-MX, the best information retrieval similarity measure performed close to pure logic-based match is cosine similarity.

Problem: we believe that they adopted very strict logical matching. They applied the logical filters at web service level instead of parameter level which results in high precision but very low recall, hence giving very limited web service options to consumers for selection.

Wenjie [6] worked in the area of biomedicine where there is requirement of dynamic discovery of web services. They tried to solve the web service discovery issue in biomedicine environment where web services are published on UDDI in WSDL form, which lacks the flexibility and expressiveness for dynamic web service discovery. They proposed the idea of describing web services in OWL-S and suggest the hybrid approaches for matching process. Their hybrid approach adopts the method proposed by

Klusch [4] with additional web service specification similarity. For logical matching, input/ output concepts are matched based on five degree of similarity measures define by [4]. They also perform the pair wise matching between the input/output concepts of web services. In addition with logical matching for parameters, they believed that web service name also contain concepts from biomedicine ontology. They extract the concepts from web service names and perform the logical matching between concepts of two web services names. The overall similarity between two OWL-S descriptions of web services is the weighted sum of input/output parameters similarity and web service names similarity. They applied logical and non-logical similarity method on all biomedicine web services describe in OWL-S.

2.9 Conclusion

We note that existing web service discovery models exploit the four main components of a web service description—i.e. input/output parameter, parameter's structure, textual description, and web service name. Table 2.1 illustrates the different service discovery approaches.

	Signature Matching				Specification Matching			
	Input/output Parameter matching		Input/output Parameter's Structure Matching		Textual Description Matching		Web Service Name Matching	
	Traditional IR Technique	**Logical Matching**	**Traditional IR Technique**	**Graph matching Algorithm**	**Traditional IR Technique**	**Short Sentence approach**	**Traditional IR technique**	**Semantic Method**
Elgazzar [9]	●		●				●	
Liu [10]	●		●				●	
Dong [18]	●		●				●	
Nayak [17]	●		●				●	
Yiqiao [11]				●	●			
Klusch [4]		●			●		●	
Wenjie [6]		●					●	
S5-Match maker		●		●		●	●	

Table 2.1: Comparison between web service matching models.

It is noticed from Table 2.1 that WSDL based matchmakers have applied traditional information retrieval techniques for all four components. On the other hand, OWLS-based matchmakers performed logical reasoning on input/output concepts with additional content-based information retrieval techniques for textual description and web service names.

Chapter 3. Hybrid Web Service Discovery Approach

3.1 Overview

As mentioned in Chapter 2, previous web service discovery models exploited non-semantic information from WSDL documents using traditional information retrieval techniques. The main information a WSDL document provides for a web service is the input/output parameters and their structures. This description in a WSDL document could be used to determine web service similarities. WSDL based matchmakers totally rely on this information. The general approach of all previously developed WSDL-based matchmakers is to convert the description of a web service into a vector of terms and to apply string similarity measures like cosine similarity, jaccard similarity. These matchmakers powered the syntactic matching by using the WordNet dictionary to add semantics to words. The problems that we have discovered in previously developed WSDL-based matchmaker are as follows:

1. One of the main problems we identified in WSDL-based matchmakers is that they adapted long-text similarity approaches which are not suitable for handling sentences of only few words. For example, if there is a large corpus then word co-occurrence could be an important source of information for calculating similarity. However if the sentence is very short, then this method would not be effective.
2. With the exception of Yaqio [11], all other WSDL matchmakers convert the structure of input/output parameters into a vector of terms and applied traditional information retrieval techniques. Yaqio [11] developed his structure matching

algorithm that only worked on an identifier's data types. Applying string matching approaches to structure matching totally ignores the structural information of a service.

3. The other problem is the lack of semantics in WSDL documents. There is no external information available for WSDL documents. To overcome this problem, previously developed matchmakers have integrated external information (i.e. WordNet dictionary), which provides semantics for words. But these matchmakers would fail to match words, like "mp3", or "codec", which does not exist in the WordNet dictionary.

OWL-S based matchmaker follows the same traditional information retrieval approaches with the addition of a logical reasoning approach for the service signatures. The second problem for WSDL-based matchmakers is solved using OWL-S. OWLS-based matchmakers contain the semantic information using external domain ontology. This domain ontology has the ability to contain all terms/concepts for a specific domain which is not possible in WordNet. Most of the previous OWLS-based matchmakers have followed the approach of Klusch [9] to the logical reasoning of input/output concepts. Klusch [9] proposed five logical filters that were applied at the parameter level i.e. EXACT MATCH, PLUG-IN MATCH, SUBSUME MATCH, SUBSUMES-BY MATCH and FAILED. Other OWL-S based matchmakers have followed Klusch's logical matching approach. The problem that we identified with previous OWLS-based matchmaker is:

- If R is the requested web service and S is an available web service in a database, then according to Klusch [9], a service S matches with a service R if all of the

parameters of service R match with the same logical filter with all parameters of service S, otherwise the match fails. The drawback of strict matching is that if there is a web service that matches exactly with the request service parameters except one parameter which shows plug-in match, Klusch's matchmaker failed to retrieve that web service. It might be possible that service S provides partial satisfaction to the consumer but it is missed by OWLS-MX due to the strict logical matching.

To solve some of the problems noticed in the existing web service discovery methods, we propose a hybrid approach that performs the logical matching between the input/output concepts, structural matching on the parameters' structure, and syntactical matching on the textual-description and web service name. The approaches followed for all the three matching components are described below:

1. *Logical Matching*: OWL-S based matchmakers compute the web service similarity using logical reasoning on service's signatures. We disagree with the idea of applying a logical filter at the service level as proposed by Klusch [9]. Instead of applying logical matching at the service level, S^5 Web Service Matchmaker is designed to apply a logical filter at the individual parameter level. The fine-grained logical filtering at parameters level helps in selecting the web services that partially satisfy the service consumer request. This partially matched web service could be helpful in web service composition in which two or more web services combine to perform a single task. For example, if there is a requested service R and an available service S, the logical similarity would be

computed for any combination of logical filters between the parameters of service R and service S. The total logical similarity between services R and S would be the cumulative similarity of all different logical matches between the input/output parameters.

2. *Textual Description Matching*: In General, WSDL based matchmakers have applied non-logical information retrieval techniques to input/output parameters, their structures and web service names. The same approach is applied to textual descriptions of web services. Applying long-text similarity approaches for short sentences is not efficient. We propose a short sentence semantic structure similarity measure for textual description; because we believe that structure of a sentence and individual words together give the complete meaning of whole sentence.
3. *Structure Matching*: WSDL documents contain the structure of the service's signatures (i.e. operations with input/output parameters). Previously developed WSDL-based matchmakers convert the parameter's structure into a vector of terms and then apply information retrieval techniques. In order to preserve the structure and perform semantic matching, we are using a Structure Preserving Semantic Matching (SPSM) algorithm that is solely developed for graph matching.
4. *Web Service Name*: For web service names, we have defined our own decomposing rules instead of just splitting on capital letters. After splitting the web service name into a vector of terms, we have applied semantic string similarity measure i.e. semantic cosine similarity.

Our proposed hybrid model is shown in Figure 3.1. This diagram depicts the four component covered by our model and the techniques that are adopted to improve the performance of the discovery model.

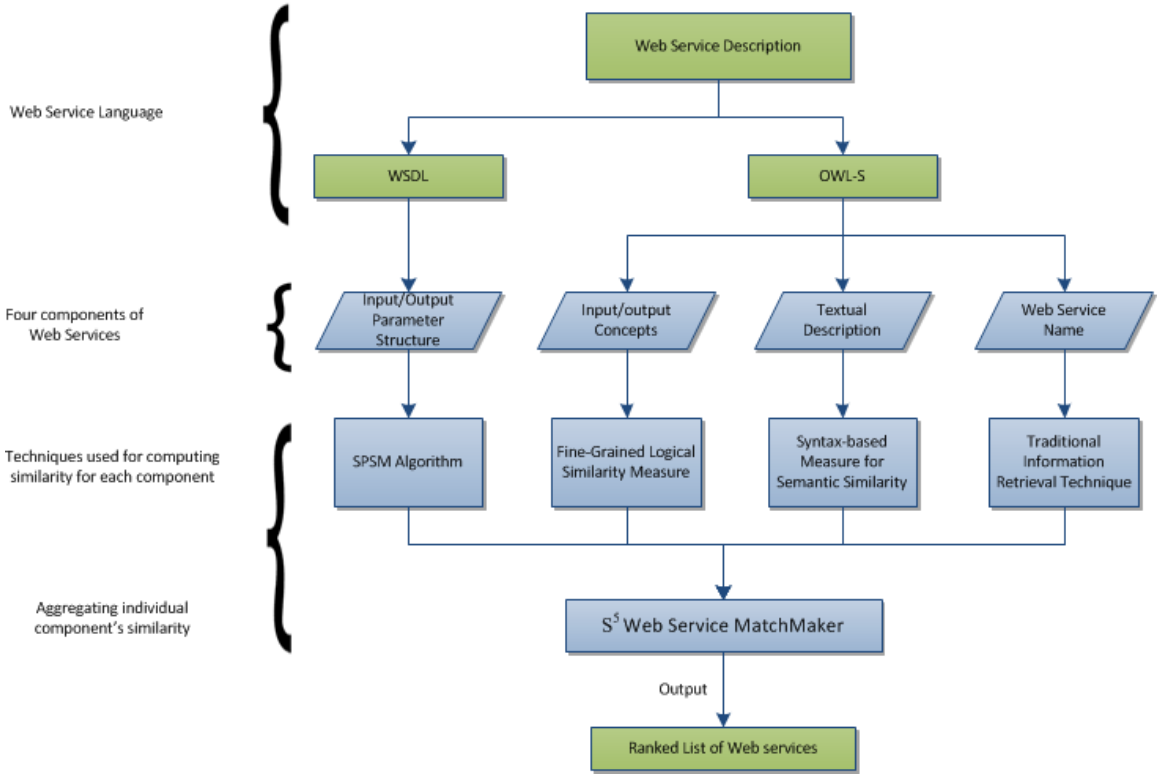

Figure 3.1: System diagram of S^5 Web Service Matchmaker.

Below, we explain the working of each component of our service discovery model.

3.2 Logical Signature Matching

Web service ontology (OWL-S) provides external knowledge, i.e. domain ontology, to represent service's arguments. As explained in Chapter 2, domain ontology provides a model for a specific domain which represents concepts and the relationship between concepts. Logical matching is the process of calculating the similarity between two concepts in domain ontology using their properties and the relationship between them. Taking any two concepts from domain ontology, there could be one of the following relation exist between them i.e. *Exact* or *Subsumption* or *NoMatch*. Klusch [4] proposed five different logical-based filters to compute the semantic similarity between requested and available web service signatures i.e. EXACT, PLUG-IN, SUBSUME, SUBSUME-BY and FAILED matches. They applied logical filters at the web service level to increase the precision of discovery model. Comparison between the signatures of two web services must contain any one of these logical filters. This approach works well where a service consumer is looking only for exact, general or specific web services match. The resulting web service may or may not contribute to web service composition or there might be a chance of missing any useful web services during selection. For example, if a user request a web service that takes 'latitude', 'longitude' and 'time' as inputs and returns 'temperature' as an output, as shown in Figure 3.2(a). Klusch's approach would not retrieve, for example, a web service that takes 'position' and 'time' as inputs and returns 'temperature' as an output, as shown in 3.2(b) (where 'position' parameter is the generic subsumer of 'latitude' and 'longitude'). The match failed because some parameters showed EXACT match while others showed PLUG-IN.

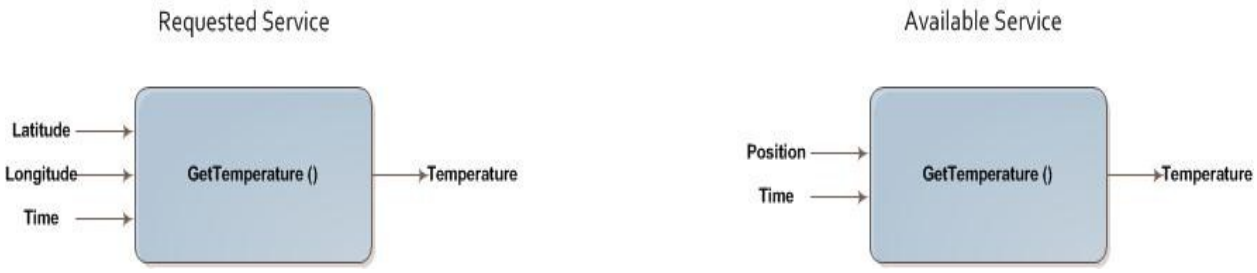

Figure 3.2: (a) Requester web service. (b) Available web service.

In order to solve this problem, we have proposed a fine-grained logical matching which computes the degree of subsumption relation for each parameter separately, instead of determining the same degree of subsumption relation for all parameters of a pair of web services. The intuition behind the fine-grained matching is that the candidate services which partially/completely satisfy user requirements should be retrieved with the low degree of similarity because many partially matched web services could contribute in building composite web service. For example, as mentioned above, service matching failed because the input parameter 'time' showed an exact match while the remaining parameters showed plug-in matches. However, structurally the parameter 'position' subsumes 'latitude' and 'longitude'. By applying logical filters at the parameter level in web service selection, we are providing the consumer with more relevant web services that could partially satisfy user needs and could also contribute to web service composition.

3.2.1 Logical Matching Filters

Logical matching is the process of computing the correspondence between the concepts of ontology. It is based on the deductive approach. It uses the input/output concepts from the ontology and logical rules to determine the subsumption relation between a pair of concepts. In web service matching, it first takes the input/output concepts from a query and advertised web services and then looks into the domain ontology that contains all the concepts of the advertised web services. In domain ontology, there is hierarchical relationship between concepts. The relationship between two concepts in domain ontology determines the semantic similarity between them. The relations could be either exact match or subsumption or no relation, as shown in Figure 3.3-3.6. Klusch first proposed the five logical filters for pure logic-based signature matching. We are using the same logical filter approach with more refinement i.e. fine-grained logical matching.

Let D be the domain ontology that has all the concepts of queries and advertised web services. LSC(C) is the set of least specific concepts C' where C' are the direct children of the concept C. LGC(C) is the set of least generic concepts C' which are the direct parent(s) of the concept C in a domain ontology. We have assigned specific values to all five filters, as proposed by OWLS-MX. The logical matching filters of S^5 Web Service Matchmaker are as follows:

- *Exact Match*: Available service S's parameters exactly match with the request service R's parameters $\Leftrightarrow$ $IN_S = IN_R \vee OUT_R = OUT_S$. This similarity shows

that the input/output parameters of an available web service perfectly match with the input/output parameters of a requested web service based on logic-based equivalence. This logic based equivalence shows that the properties of a pair of concepts in domain ontology are exactly the same, as shown in Figure 3.3. '1' is assigned as the maximum degree of similarity.

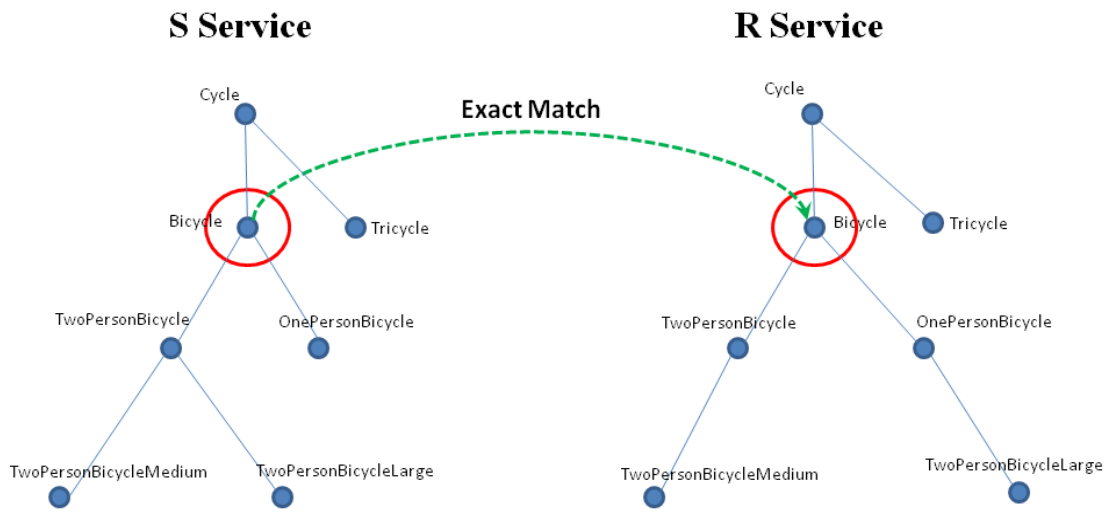

Figure 3.3: Exact similarity between two concepts.

Plug-in Match: Available service S's parameters plugs-into the requested service R's parameters $\Leftrightarrow IN_S \in LSC(IN_R) \vee OUT_S \in LSC(OUT_R)$. This filter gives a little flexibility to the exact match. It guarantees that the available service S will have the most specific input/output parameters as compared to what has been requested by the consumer request service R. Based on OWLS-MX, the degree of similarity for PLUG-IN match is 0.8.

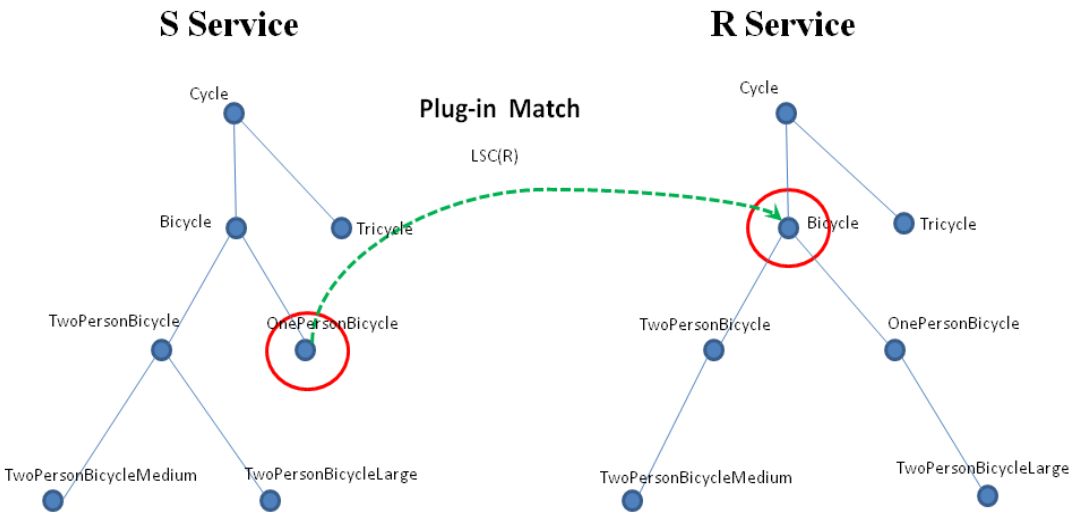

Figure 3.4: Plug-in match between two concepts.

- *Subsume Match*: Available service S's parameters SUBSUME the requested service R's parameters ⇔ $IN_S \in$ Subclass (IN_R) ∨ $OUT_S \in$ Subclass (OUT_R). Subsume match is similar to the plug-in match except that it provides more flexibility in comparing the subsumtption relation in an ontology for the input/output parameters. Instead of comparing with the direct child of service R's parameter, as in PLUG-IN, it looks for any subsumed concept in the hierarchy. If R's input/output concept is the child of S's input/output concept in an ontology at any depth, it would be considered as a SUBSUME match, as shown in Figure 3.5. Based on OWLS-MX, SUBSUME filter has assigned the value of 0.7 degree of similarity.

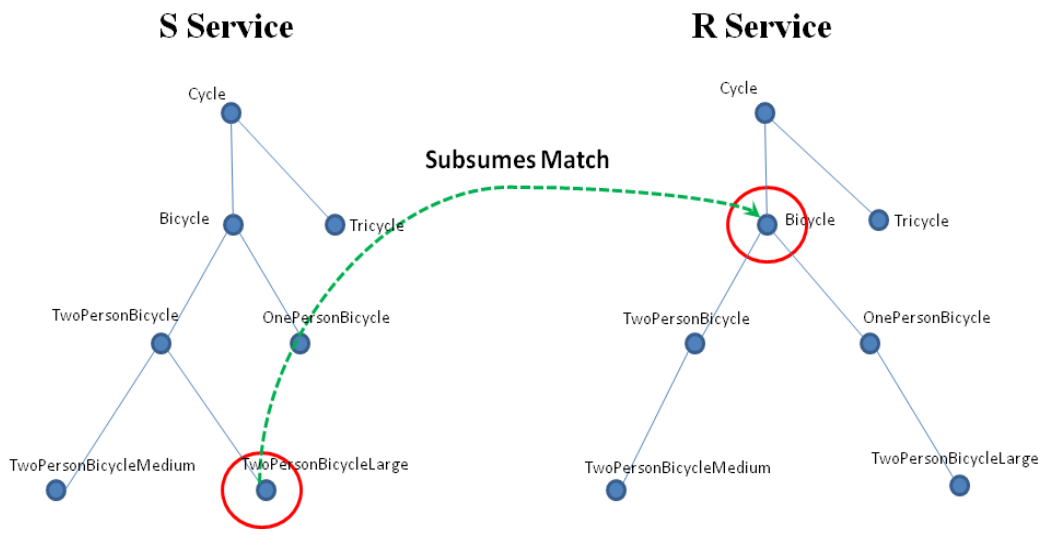

Figure 3.5: Subsumes match between two concepts.

- *Subsumed-by Match*: Requested service R's parameter is SUBSUMED-BY an available service S' parameter ⇔ $IN_S \in LGC(IN_R) \vee (OUT_S = OUT_R \vee OUT_S \in LGC(OUT_R))$. This match focuses only on the direct parent concept instead of going for more general concepts, as shown in Figure 3.5. For example, if a service consumer requests a service that takes the input parameter 'bicycle', least generic concept 'Cycle' might fulfill the user requirement but a service that takes 'automobile' may not provide the desired functionality. The reason is that concept 'automobile' is a general concept which descends concept 'Jeep', 'Scooter', 'Truck', etc. And, there is a direct relationship between the concept 'Cycle' and 'bicycle' and have more commonalities as compare to the concept 'Bicycle' and 'automobile'. Following the OWLS-MX, 0.6 degree of similarity is assigned to SUBSUMED-BY filter.

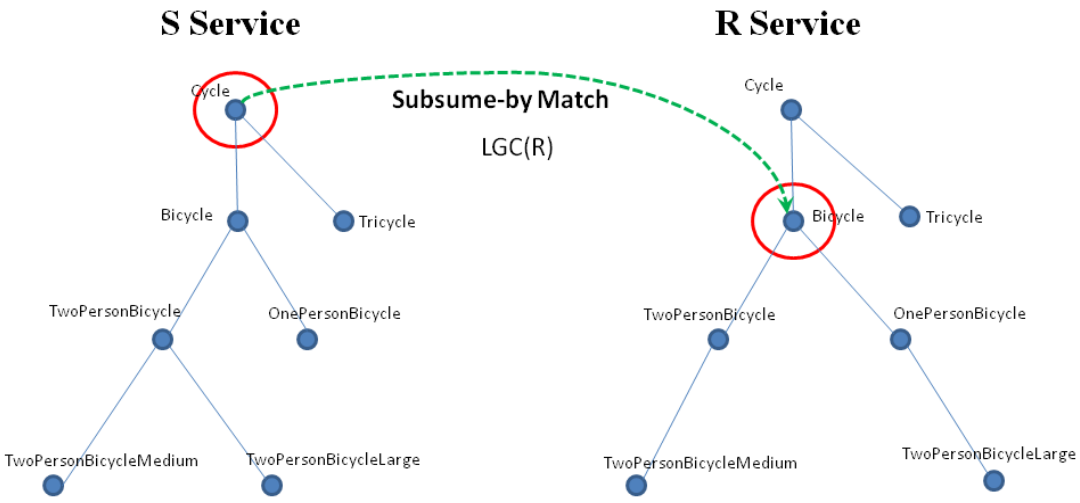

Figure 3.6: Subsume-By match between two concepts.

- *Failed Match*: If none of the above filter match, then the logical comparison between the S's parameters and R's parameters is failed. This will assign the similarity of 0.

Based on these filters, we individually compare the logical similarity between each pair of parameters of available service S and requested service R and then aggregate the similarity of all parameters. The important thing to note that we have given equal weights to the inputs and outputs similarity measures, as shown in Equation 3. We adopt this approach to achieve our goal i.e. to facilitate the consumers in the web service composition process. Below, we present two cases to support our approach.

- Case 1: In case1, we have shown a condition where there is a complete logical similarity between the input parameters but fail to match the output parameter. For example, a consumer request service R that takes inputs R_{i1}, R_{i2}, and R_{i3} and returns output R_{o1}. S^5 Web Service Matchmaker returns two web services S1 and S2 that logically match with the service R. The service S1 shows the total logical input parameters similarities with the R's input parameters while fails to match the R's output (i.e. R_{o1}), as shown in Figure 3.7. On the other hand, service S2 takes take some of the desired consumer input parameters with an addition of one more parameter which is an output of S1, as shown in Figure 3.7. The complete consumer satisfaction could be achieved by composing web services S1 and S2 in such a way that the output of service S1 is attached to the input of Service S2. The service S2 is expected to give the consumer's desired output.

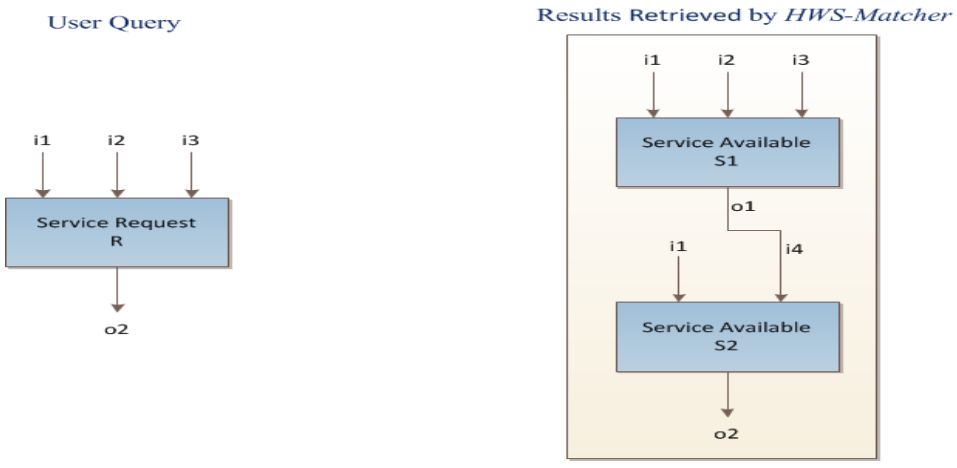

Figure 3.7: Composition of partially matched services.

- Case 2: In case 2, we present a scenario in which the output parameter of the consumer's request logically matches while there is a logical mismatch between the input parameters. For example, taking the previous consumer request R with the input parameters R_{i1}, R_{i2} and R_{i3} and the output parameter R_{o1}, S^5 Web Service Matchmaker retrieves three web services S1, S2 and S3. Each of the retrieved web services partially satisfies the consumer needs, as shown in Figure 3.8. The important thing to note is that the service S3 shows a logical failure for all its input parameters but still gets selected because of the logical similarity of its output parameter. With the composition of the services S1, S2, and S3 (as shown in Figure XX), the consumers could achieve their desired requirements.

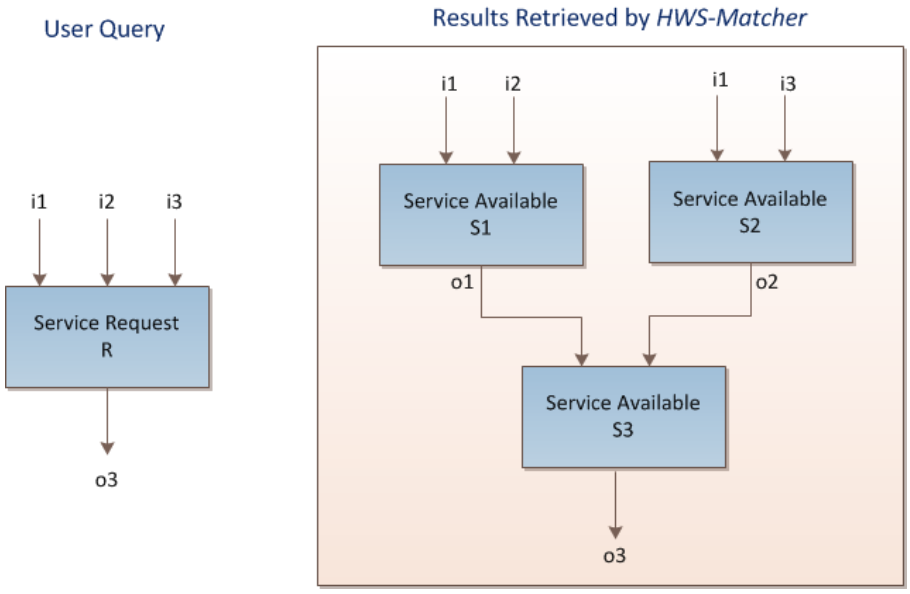

Figure 3.8: Composition of partially matched services.

Hence, the overall similarity is the average of total input similarity and total output similarity. The equations for calculating the overall logical similarity are shown as:

Total input parameters similarity:

$$SIM_{IN}(S_{IN}, R_{IN}) = (\sum_{i=0 \text{ to } N} LogicSim(S_{INi}, R_{INi}))$$ where N = # R's input parameters

(Equation 1)

Total output parameters similarity:

$$SIM_{OUT}(S_{OUT}, R_{OUT}) = (\sum_{i=0 \text{ to } N} LogicSim(S_{OUTi}, R_{OUTi}))$$ where N = # R's output parameters

(Equation 2)

Total Similarity:

$$SIM(S, R) = (SIM_{IN} + SIM_{OUT}) / N$$ where N = # R's input and output parameters

(Equation 3)

The steps involve in the logical signature matching are as follows:

1. Web service consumer provides the query web service to our model represented in OWL-S.
2. S^5 Web Service Matchmaker extracts the input/output parameters that reflect the arguments of a service.
3. For every pair of input concept of requested and available web services, S^5 Web Service Matchmaker computes the degree of similarity using domain ontology. Based on the degree of similarity, it computes the similarity value for all input concepts using equation 1.
4. A similar method is performed for the output parameters and computes their similarity value using equation 2.

5. The total similarity between a pair of services is determined by the input and output parameters similarity using equation 3.

3.2.1 Example

A Service consumer queried for a web service called R which takes two inputs, i.e. 'Car' and 'OnePersonBicycle', and returns "Price" as an output. The input/output description of service R is then compared with all available web services descriptions in OWLS-TC. If there is a web service S in OWLS-TC, 'BicycleCar_Price_service', which also takes two inputs, 'Car' and 'Bicycle' and returns 'Price', similarity would be calculated between all corresponding input/output parameters of service S and service R. Service S's input parameter 'Car' matches exactly with service R's input parameter 'Car' and also compute the similarity value of '1'. S's input parameter "Bicycle" computes 'subsumes-by' relationship when compared with R's input parameter 'OnePersonBicycle', hence compute the similarity value of 0.6. According to equation 1, the overall input similarity would be (1+0.6)/2 = 0.8. The output parameter 'Price' of service S exactly matches with the output parameter 'Price' of service R, which results in similarity value of '1'. According to equation 2, the overall output similarity would be 1/1 = 1. The total logical similarity would be the average of similarity of input parameter and similarity of output parameters that is (1+0.8)/2 = 0.9, using equation 3.

3.2.2 Implementation

Our method performs the logical matching between the two concepts extract from the OWL-S service profile. The relationship between these concepts is extracted from domain ontology. To manipulate domain ontology, JENA API is used. As mentioned in Chapter 2, JENA is a java framework for processing ontologies. In the logical matching, it is used to compute what kind of relationship (i.e. Equivalent, Subclass, Super-class, or no relation) exists between two concepts. The pseudo code for the logical matching is given in Figure 3.9:

```
1: LogicalSimilarity( Signature_SERVICE, Signature_REQUEST)
{
2: foreach s ∈ INPUTSIGNATURES_SERVICE, r ∈ INPUTSIGNATURES_REQUEST
3: InputSimilarity ⟵ InputSimilarity + match(s, r);
4: foreach s ∈ OUTPUTSIGNATURES_SERVICE, r ∈ OUTPUTSIGNATURES_REQUEST
5: OutputSimilarity ⟵ OutputSimilarity + match(s, r);
6: TotalSimilarity = InputSimilarity + OutputSimilarity / size(Signature_REQUEST);
}

8: Match(Concept1, Concept2)
{
9:     if (Concept isEquivalent Concept2)
10:            DegreeOfMatch ⟵ 1
11:    if (Concept1 isLeastSubClassOf Concept2)
12:            DegreeOfMatch ⟵ 0.8
13:    if (Concept1 isSubClassOf Concept2)
14:            DegreeOfMatch ⟵ 0.7
15:    if (Concept1 isSuperClassOf Concept2)
16:            DegreeOfMatch ⟵ 0.6
17:     If ( Concept1 noMatch Concept2 )
18:            DegreeOfMatch ⟵ 0
}
```

Figure 3.9: Logical matching pseudo code.

3.3 Non-logical Signature Matching

The second important method of web service's signature-matching involves the comparison of the WSDL document's parameter structure. Even semantic information plays a very important role in web service discovery, signature's structure matching can be used to empower and enhance the effectiveness of the discovery process. The integration of structure-based matching with logical matching would be helpful where semantic description of web services is not available or domain ontology is not well defined. Signature's structure matching helps in inferring the basic semantics of the web service using the structure of operations and their parameter types.

We are proposing a non-logical signature's structure-matching approach, integrated with logical signature matching, to improve the web service discovery process in a situation where ontology is incomplete or absent. For example, if there is a web service without OWL-S description, structure matching would play a role in determining similarity between a pair of the services while logical matching fails completely.

Since the WSDL document is based on XML, web service input/output parameters are described in a hierarchical way. The structure matching of two web services is based on the similarity of their operations. Each operation consists of input and output messages, which correspond to the signatures of a service, as shown in Figure 3.8. Signature's structure matching takes two graph-like structures and computes similarities between the nodes of the graphs. In order to calculate the structural similarity in our hybrid approach, we are using a Structure Preserving Semantic Matching (SPSM) algorithm, described in Chapter 2. An SPSM algorithm computes the semantic similarity between two graphs while preserving their structures. A graph is extracted from the operation's message of each WSDL document. Nodes in the graph are the input/output identifiers and their data types. The relationship that could exist between any two graph nodes is equivalent, general, specific, and mismatch.

The steps involve in computing semantic structural similarity between two WSDL documents are:

- **Step1**: Extract the WSDL URI from the service grounding. This WSDL URI is used to locate the WSDL document for the web service requested by a consumer.

- **Step 2**: WSDL document is then processed to extract operations' messages and types, along with the identifier names. The extracted structure would be in a well defined format that can be easily processed by an SPSM algorithm. Figure 3.10 shows an extraction of WSDL document into a well formatted graph structure. Graph structure contains the label of operations, input/output parameters, and their data types as nodes.

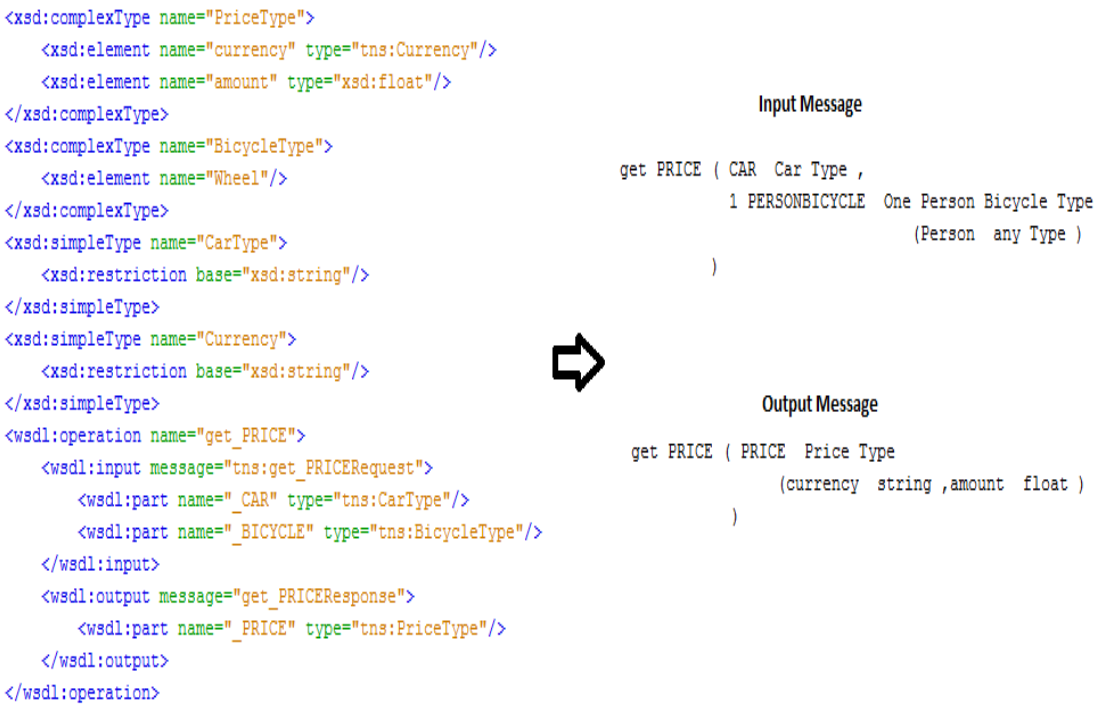

Figure 3.10: Extracted structured information from WSDL document.

- **Step 3:** Extracted graph structure would be given to the SPSM algorithm for calculating semantic structure similarity. SPSM converts the labels of graph nodes into concepts using WordNet. Taking two graph-like structures into account, it determines the semantic correspondence between each pair of nodes. For example, it identifies node 'car' from one graph to be semantically equivalent with the node 'vehicle' from the other graph because they are synonyms in WordNet.

- **Step 4:** SPSM returns the degree of semantic structural similarity between two graph-like structures which ranges between 0-1. '1' means exact match while '0' means no match.

As WSDL document contains a separate graph-like structure for input and output messages, the semantic similarity would be computed separately. First, the semantic structural similarity is calculated between a pair of input message structures. The same procedure is then applied for a pair of output message structures. The overall structural similarity would be the average similarity of input and output messages.

3.3.1Example

A user queried for a web service that takes concept 'Car' and 'OnePersonBicycle' as an input and returns concept 'Price' as an output. The structure of operation's messages of the requested WSDL document is then compared with the corresponding structure of all

available WSDL documents. If there is a WSDL document S in OWLS-TC that takes 'Car' and 'Bicycle' as inputs and returns "Price", then the structural representation of the operation's messages and data types would be as shown in figure 3.11.

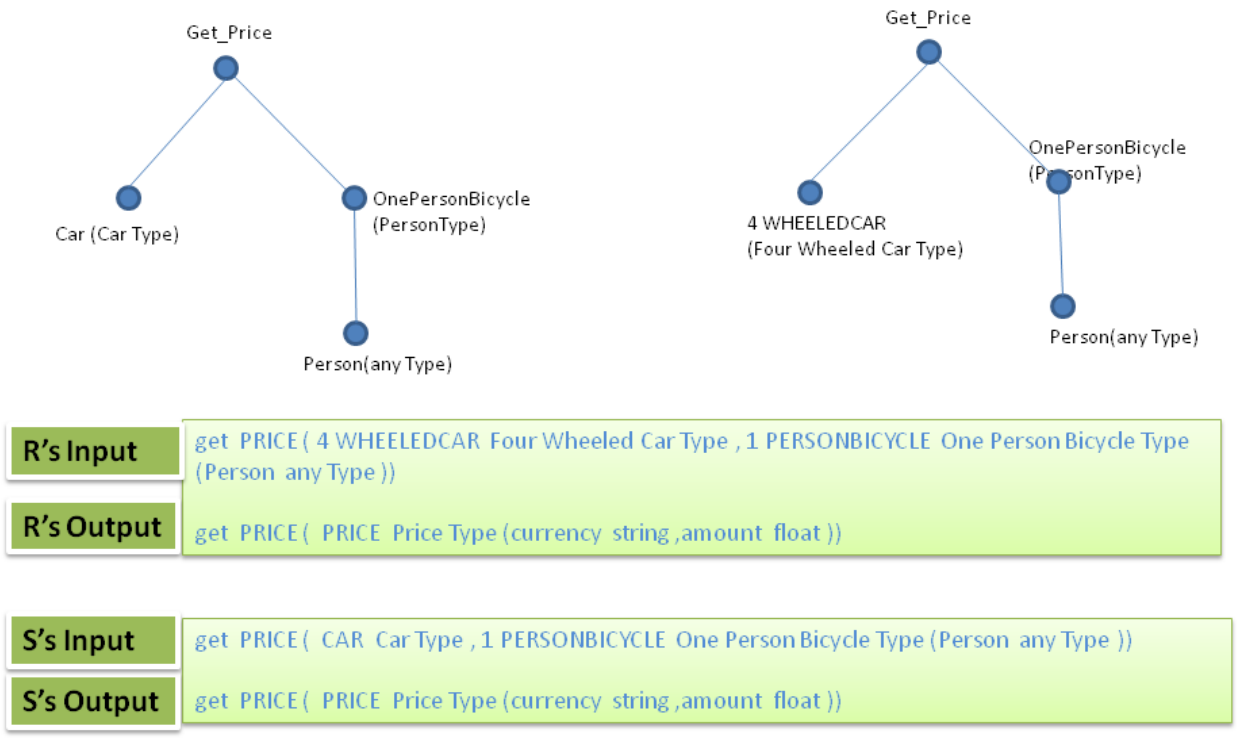

Figure 3.11: Textual representation of WSDL structure.

An SPSM takes these structures, shown in Figure 3.11, and performs the semantic structural similarity measure on the input/output messages separately. In this case, SPSM returns score of 0.875 and 1.0 for the structural similarity between pairs of input messages and outputs messages, respectively. The overall similarity would be the average of the structural similarity of the inputs and outputs i.e. (1.0 + 0.875) / 2 = 0. 9375.

3.3.2 Implementation

We have used a Structure Preserving Semantic Matching (SPSM) algorithm. Our method extracts the identifier's structure from WSDL using an XML parser and provides the extracted structure to the SPSM for calculating similarity. Pseudo code for structure matching is given in Figure 3.12.

```
1: StructureMatching(WSDL_REQUEST, WSDL_SERVICE)
{
2: requestInputStructure ⟵ ExtractInputStructure(WSDL_REQUEST)
3: requestOutputStructure ⟵ ExtractOutputStructure(WSDL_REQUEST)
4: serviceInputStructure ⟵ ExtractInputStructure(WSDL_SERVICE)
5: serviceOutputStructure ⟵ ExtractOutputStructure(WSDL_SERVICE)
6: inputSimilartiy ⟵ StructurePreservingSemanticMatching(requestInputStructure ,
                                                         serviceInputStructure)
7: outputSimilartiy ⟵ StructurePreservingSemanticMatching(requestOutputStructure,
                                                          serviceOutputStructure)
8: TotalStrucutreSimilarity ⟵ Average (inputSimilartiy, outputSimilartiy)
}
```

Figure 3.12: Structure matching pseudo code.

3.4 Textual description matching

Sometimes logical matching fails because of the lack of the logical relationship between a pair of concepts in a domain ontology. In that scenario, totally relying on signature similarity measures would fail to discover two similar web services. Therefore, in order to get better results, specification similarity measures should be combined with signature similarity measures. OWL-S service profile provides the signatures and specification

description of a web service, as described in Chapter 2. Along with the essential signatures (i.e. input/output parameters), the specifications (i.e. textual description) also play a leading role in discovering relevant web services. Usually, a textual description of a web service provides a brief functional description of it. It contains summarized information about the functionality of the web service offered and the information that the web service publisher wants to share with the web service consumers. Normally a textual description is a sentence of 10-50 words long, which contains a well-formed syntactic structure.

All web service discovery models that have exploited textual description have adopted a keyword-based semantic information-retrieval method. This method was used to identify and rank similar web services based on the textual description. A vector space model is widely used for comparing similar web services. The steps usually followed are: converting a textual description into a vector of terms, removing stop-words, stemming and then calculating the cosine similarity between corresponding vectors using WordNet. The main drawback of a vector space model is that terms within a textual description are treated as isolated words, totally ignoring the semantic associations among them.

We have proposed a method that exploits the structure of textual descriptions because we believe that the structural way the words combine and the individual meaning of words collectively give the meaning of the whole sentence. Hence, it may be possible that two sentences may contain the same words but the different syntactic structure of the sentence may give two different meanings. For example, two sentences “This service receives ‘car’ and returns ‘price’” and “This service receives ‘price’ and returns ‘car’” are semantically the same but their syntactic structure is different. The spirit of textual

description structural similarity is taken from Oliva [29], which proposed a new approach for computing text similarity of short sentences. Our method extracts the textual description from a service profile, processes its structure and then compares the semantic similarity with the textual description of corresponding service. The steps involved in computing the textual semantic similarity are:

- *Step 1:* Web service textual description is extracted from a service profile for semantic comparison. This textual description is in natural language form.
- *Step 2:* Textual description is then parsed using the Stanford parser to extract the syntactic structure. This syntactic structure is represented in typed dependency forms, provided by Stanford, which shows grammatical relationships in a sentence. Typed dependency relationships show a relation between two words like "Subject of word 'take' is 'service'", "Object of word 'take' is 'price'". Based on Marneffe [25], there are 52 grammatical relations that are all binary relations between a governor/head and a dependent of a sentence.

Typed dependencies are extracted using Penn Treebank part-of-speech and phrasal labels, are shown in Table 3.1.

Abbrev	Auxpass	Det	Npadvmod	Poss	Quantmod
Acomp	CC	Dobj	Nsubj	Possessive	Rcmod
Advcl	Ccomp	Expl	Nsubjpass	Preconj	Ref
advmod	Complm	Infmod	Num	Predet	Rel
Agent	Conj	Iobj	Number	Prep	Tmod
Amod	Cop	Mark	Parataxis	Prepc	Xcomp
Appos	Csubj	Mwe	Partmod	Prt	xsubj
Attr	Csubjpass	Neg	Pcomp	Punct	
Aux	Dep	Nn	Pobj	Purpcl	

Table 3.1: 52 grammatical relations.

In our method, we are focusing on the subjects, verb, and objects of a sentence for computing structural similarity between textual descriptions. We are considering 11 out of 52 grammatical relations for our method and have classified them into three groups that are subject, verb and object.

1. ***Subject*** *: csubj, csubjpass, nn, nsubj, nsubpass, num, number, xsubj*
2. ***Verb****: dobj*
3. ***Object****: dobj, iobj, pobj*

- *Step 3:* After extracting the syntactic information from a sentence, our method calculates the semantic similarity between the same syntactic function of two textual descriptions. The LIN similarity measure is used to calculate the semantic relationship between two words i.e., in this case, heads and dependents. As already defined in Chapter 2, LIN similarity measure is the ratio of the information content of the least common subsumer to the information content of each word. The external information is powered by

WordNet. In matching a syntactic function, if the similarity between heads fails then the similarity between their dependents would be totally ignored, no matter what the dependent similarity would be computed and vice versa. For example, there is a comparison between two syntactic functions (i.e. dobj(returns, price) and dobj(receives price)). In this comparison, the similarity between heads (i.e. 'returns' and 'receives') is 0 while the similarity between dependents (i.e. 'service' and 'service') is 1. This shows that one web service receives input 'price' and the other returns output 'price'. In such a case, the overall similarity of a syntactic function is considered as 0.

- *Step 4:* The overall syntactic structural similarity between two textual descriptions is computed by taking an average of subject, verb and object similarity. If R is a requested web service and S is a available web service, then the overall textual similarity would be:

$$SIM_{TEXT}(S, R) = SIM_{SUBJECT}(S, R) + SIM_{VERB}(S, R) + SIM_{OBJECT}(S, R)/3$$

3.4.1 Example

A user queried matchmaker for a web service, named R, which takes "Car" and "OnePersonBicycle" as an input and returns "Price" as output. The queried web service description would be described in OWL-S. As we have already mentioned, a service profile in OWL-S contains the textual description of a web service. The textual description of user query R is "This service returns price of the pair of a car and 1 person

bicycle". On the other hand, the matchmaker extracts the textual description of all available web services from OWL-S service profiles. If there is a service "BicycleCar_Price_service", named S, in OWLS-TC and the textual description of S is "This service returns price of the requested pair of a car and a bicycle". Our method takes these descriptions and extracts the grammatical relations using Stanford parser. The resulting relations for subjects, verb and objects are shown in Table 3.2.

	R Service	S Service	LIN Similarity
Subject + Object	Num (bicycle, 1)	Nsubj (returns, service)	0
		Dobj (returns, price)	Different Group
	Nn (bicycle, person)	Nsubj (returns, service)	0.1223
		Dobj (returns, price)	Different Group
	Nsubj (returns, service)	Nsubj (returns, service)	1.0
		Dobj (returns, price)	Different Group
	dobj (returns, price)	Nsubj (returns, service)	Different Group
		Dobj (returns, price)	1.0
	SIM (Subject + Object)		2.1223/4 = 0.5305
Head of Sentence (Verb)			
	dobj (returns, price)	dobj(returns, price)	1.0
			(1.0+0.530575) / 2
Total Similarity			**0.765**

Table 3.2: Textual description similarity.

In Table 3.2, column 'R service' shows the grammatical relations that refer to the subject, object or verb in a sentence. The same is true for column 'S service'. As mentioned above, the grammatical relation is a binary relation between a head word and its dependents. So in our example, four grammatical relations are extracted for a query service: Num (number modifier), Nn (Noun modifier), Nsub (nominal subject) and dobj (direct object). Similarly two grammatical relations are extracted for a web service S from OWLS-TC: Nsubj (nominal subject) and dobj (direct object). Our method performs

the comparison on every pair of syntactic function from two different web services, if and only if, they are in the same group of subject, verb or object. The comparison is done between the heads and dependents. Using LIN measure, similarity is computed between the heads and dependents. Num (bicycle, 1) is compared with Nsubj (returns, service), as both lie in the subject group. LIN computes similarity score of 0.0 between the heads (i.e. 'bicycle' and 'returns') and also computes similarity score of 0.0 between dependents (i.e. '1' and 'service'). There is no match between Num (bicycle, 1) and Dobj (return, price) as they both lie in different groups. In this way, the similarities of all the syntactic function are calculated for the subject and object groups which is, in this case, (0+0.1223+1+1)/4 = 0.5305. The head of a sentence, that is a verb, is computed from the syntactic function 'dobj' (direct object). As mentioned in Marneffe [25], "the direct object of VP is the noun phrase which is (accusative) object of a verb". In Table 2, the semantic similarity between the heads of sentences (i.e. 'returns' and 'returns') is 1. The overall semantic similarity computed between the R service's textual description and S service's textual description is (Sim (Subject+Object) + Sim (verb))/2 = (0.5305 + 1.0) / 2 = 0.765.

3.4.2 Implementation

For textual descriptions, we first extract the tagged dependency parse tree using Stanford parser API and remove all tags except those for subject, verb and object. Similarity between same group tags is computed using the LIN similarity measure. Pseudo code for the textual description matching algorithm is given in Figure 3.13.

```
1:  TextualDescriptionMatching(DESCRIPTION_SERVICE, DESCRIPTION_REQUEST)
2: {
3:   serviceDesTags ←  extractTaggedDependcyStructure(DESCRIPTION_SERVICE)
4:   requestDesTags ←  extractTaggedDependcyStructure(DESCRIPTION_SERVICE)
5:   foreach sTaggedKey ∈ serviceDesTags
6:           foreach rTaggedKey ∈ requestDesTags
7:                   if( sTaggedKey and requstDesTags are in same Group)
8:                   {
9:                           similarity ← similarity + LIN_SIMILARITY (sTaggedKeyValue, rTaggedKeyValue)
10:                  }
11 :}
```

Figure 3.13: Textual description matching pseudo code

3.5 Service Name Matching

One of the specifications that a web service provides is a service name. A web service name is a text of few words that is usually used to identify a web service. Sometimes, it may contain words to describe the functionality of a web service itself. For example, the service "OnePersonBicyclePrice" contains very important terms i.e. 'one', 'person', 'bicycle', and 'price', which itself define the functionality of a web service. The intuition behind considering the web service name into our hybrid model is that in a situation where logical/non-logical signatures and textual-description similarity score is very low; it could play a role in web service selection. The logical signature matching could decrease because of the lack of logical relation between a pair of concepts in domain ontology. It might be possible that two services provide the same functionality but have different signature structures, which results in a failure of signature's structure matching. As textual descriptions are provided by human, two similar services could fail to match

because one or both textual descriptions are missing. In the case, where all three measures show very small similarity between a pair of services, the only information that could possibly select two similar services is web service name. Usually, the web service name is composed of words using capital letters, numbers, '_', and '-'to separate words. We split web service name into multiple single valid words. This splitting is done based on the assumption that every capital word, number, '_' and '-' indicates the start of another word. Decomposing rules are defined in Table 3.3.

Rule	Name	Result
Case Change	OnePersonBicyclePrice	One Person Bicycle Price
Underscore '_'	OnePerson_BicyclePrice	OnePerson BicyclePrice
Number	3PersonBicycle	3 PersonBicycle
Dash '-'	OnePerson-Bicycle-Price	OnePerson Bicycle Price
Space	OnePerson BicyclePrice	OnePerson BicyclePrice

Table 3.3: Decomposing rules for web service name.

Keyword-Based matching is not efficient because if a web service name contains a word "car", it would not retrieve web services from the database which contain words "vehicle" or "automobile" in their names. This will cause poor recall for non-semantic web service name matching. To overcome this problem, semantics have been added using WordNet. Instead of just comparing the exact words, a similarity is performed between the synonyms of the word. The steps involved in computing web service name similarity are:

- *Step 1:* Extracts the web service name from the service profile.
- *Step 2:* Decomposes composite web service name into a vector of meaningful words using rules defined in Table 3.3. In our case, we are not only considering the string words but also taking numbers into account because, for example, service name "3PersonBicyle" is close to "3PersonBicycle" than "2PersonBicycle".
- *Step 3:* Once the vectors are extracted for the web service name, computes the similarity between two vectors of web service names using semantic cosine similarity. Here, we are not just comparing the exact match, but including the synonyms for each word using WordNet.

The result of cosine similarity based on semantics ranges between 0 and 1, where 0 shows no similarity between the web service names and 1 indicates an exact match.

3.5.1 Example

User sends a web service request, "CarOnePersonBicycleService", to our matchmaker to discover all available services similar to query. The first step involved in the web service name matching is preprocessing of web service name. As mentioned above, service name could be a composite of multiple words. Based on the rules defined in Table 3.3, CarOnePersonBicylceService is decomposed into a vector of "Car", "One", "Person", "Bicycle", and "Service". This vector of individual words is semantically compared with all available vectors of web service names using cosine similarity. For example, it would

be compared with a vector “Bicycle”, “Car”, “Price”, “service”, extracted from a “BicycleCarPriceService”. The semantic cosine similarity computed for these two vectors is 0.8164.

3.5.2 Implementation

Web services names are first decomposed into a vector of words using a regular expression. For each word, synonyms are extracted from WordNet. Semantic cosine similarity is then applied between two vectors representing two different web service names. Pseudo code for web service matching algorithm is given in Figure 3.14:

```
1:  ServiceNameMatching( NAMESERVICE, NAMEREQUEST)
2:  {
3:  serviceVector   ←—  SPLIT (NAMESERVICE)
4:  requestVector   ←—  SPLIT (NAMEREQUEST)
5:  similarity      ←—  SEMANTICCOSINESIMILARITY (serviceVector, requestVector)
6:  }
7:
8:
9:  SPLIT (NAME)
10: {
11:  Split Composite word based on
12:       a: Case Change
13:       b: UnderScore '_'
14:       c: Number
15:       d: Dash '-'
16:       e: Space
17:}
```

Figure 3.14: Web service name matching pseudo code.

3.6 Total Web Service Similarity

S^5 Web Service Matchmaker performs the logical and non-logical similarity approaches for signatures and specifications of web services. It computes the similarity of all four components individually and then aggregates using equation 4 and equation 5. We have prioritized all four similarity measures based on their functional importance. Instead of just averaging all components, we have given importance to signature matching up to 0.7 while keeping the textual description specification similarity to 0.3, as shown in equation 4. In the domain of our data set, logical and structural descriptions of web service signatures represent the same hierarchy and the same relations. If there is any pair of service for which parameter structure does not match to concepts in the domain ontology, then the overall signature similarity would be affected. Service name specification similarity has the least contribution in our hybrid model because if a pair of services fails logically, structurally and textually, then having the same service name would not provide the desired functionality to the service consumer. If their total similarity does not give the maximum similarity (i.e. 1), the 20% of remaining similarity would be compensate by web service name similarity, as shown in equation 5.

$$SIM_1(R, S) = 0.7\ (\ Avg\ (Logical\_SIM(R, S), Structural\_SIM(R, S))) + 0.3*Textual\_SIM(R, S) \qquad \text{(equation 4)}$$

$$SIM_2(R, S) = SIM_1(R, S) + 0.2 * (1 - SIM_1(R, S))* Name\_SIM(R, S) \qquad \text{if } SIM_1(R, S) < 1 \qquad \text{(equation 5)}$$

The intuition behind giving specific percentages to all four components is described as:

1. The logical signature similarity measure captures the similarity between the input/output concepts in the domain ontology, while the structural signature similarity assures that the two web services are similar in term of input/output

parameters structure. If two services have a significant level of match/mismatch in their concepts and structural level then the overall similarity should be affected significantly. Therefore we have chosen the highest weight for this measure (i.e. 70%).

2. On the other hand, it is also important how a service captures the instance of a particular concept. The exact hierarchy and data types (i.e. structure) of a parameter in the representation of a particular concept of the domain may vary from one service to another. For example, the same concept 'Car' in the domain may be captured as a combination of concepts 'Make' and 'Model' in one representation and another representation may associate additional concept 'Color' with the description of a concept 'Car'. In this case, although the two representations capture same concept semantically, their structures are different. While considering similarity among services it is also important to consider similarities between structures of concepts corresponding to service parameters. Therefore, we weigh logical and structural similarity equally.
3. Textual description specification also has a major role in distinguishing a web service. The intention/behavior of a web service is usually expressed in a textual description. It has equal importance as logical and non-logical signatures because it might be possible that two web services have the same signatures but different specifications.
4. Web service name could also be used to identify similar web services. But if logical, structural and textual description gives maximum similarity then there would not be any use of considering service name similarity. But, if equation 4

fails to gives maximum similarity then the name similarity would contribute 20% of the remaining similarity. For example, if total similarity of logical, structural and textual similarity is 0.7 then the name similarity would be used. If the name similarity is 0.8, it will contribute its 20% in the remaining total similarity i.e. 0.3 (1 – 0.7). The overall similarity using equation 5 would be 0.748.

S^5 Web Service Matchmaker discovery model works on signatures and specifications of a web service, hence all the four similarity components are considered in Equation 4 and 5. Depending on the requirements, a consumer can select either a pure logical S^5 Web Service Matchmaker or hybrid S^5 Web Service Matchmaker. The logical S^5 Web Service Matchmaker discovery model is totally based on semantic signatures matching whereas the hybrid S^5 Web Service Matchmaker exploits the signatures and specifications of a web service using logic-based and non-logic-based techniques. If a user request does not contain the signature's structure and textual-description, then he should use pure logical similarity model. On the other hand, if a consumer request contains more web service description (i.e. signatures' structure and textual-description) then hybrid S^5 Web Service Matchmaker is a best option. Below we have shown the different scenarios at different similarity scores of signatures and specifications of web services in Table 3.4.

	Logical Similarity	***Structural Similarity***	***Textual Similarity***	***Name Similarity***	***Total Similarity***
Case 1	*1*	*1*	*1*	*1*	0.7(*Avg* (1, 1)) + 0.3*1 = 1
Case 2	*1*	*1*	*0.8*	*1*	*SIM1* = 0.7(*Avg* (1, 1)) + 0.3*0.8 = 0.94 *SIM2* = *SIM1*+*0.2*(1-*SIM1*)*1 = 0.952
Case 3	*0*	*1*	*1*	*1*	*SIM1*= 0.7(*Avg* (0, 1)) + 0.3*1 = 0.65 *SIM2* = *SIM1*+0.2*(1-*SIM1*)*1 = 0.72
Case 4	*1*	*0*	*0*	*0*	*SIM1*= 0.7 (*Avg* (1, 0)) + 0.3*0 = 0.35 *SIM2* = *SIM1*+0.2*(1-*SIM1*)*0 =0.35

Table 3.4: Different Scenarios of similarity of four components of web services.

Case 1:

If two web services show exactly similar logical, structural and textual description similarity, no matter what their names would be their overall similarity would be 1. A similarity score of '1' shows the exact match between a pair of services, and the selected service would be ranked at the top in candidate list. The total similarity would be computed using *equation 4, i.e.*0.4*1 + 0.3*1 + 0.3*1 = 1.

Case 2:

As shown in Table 3.4, textual description similarity shows a partial match in the comparison between a request and available web services. It might be the case where all signatures of web services match logically and structurally but due to change in behavioral information of a web service provided by a publisher, it does not perfectly match with the consumer requirements. In this case, the total similarity of the logical, structural, textual description components is less than 1, using equation 4. The web

service name similarity would be used to compensate the remaining portion of total similarity, using equation 5. The overall similarity would be 0.952 which is very close to an exact match, as compare to case 1. Our assumption is that the behavioral (textual) description should contribute nearly equally to half of logical and structural component. This is because it contains the detail description of functionality of a web service, which could help in identifying two similar web services and it should give almost equal importance.

Case 3:

Table 3.4 shows a scenario where matching between a pair of input/output concepts logically fails but other components show an exact match. It might be the case where domain ontology fails to show any kind of relation between input/output concepts of web services due to lack of semantic annotation capture in a domain ontology. If logical matching totally fails then 35% of the total similarity is removed. Yet, the candidate service could provide a certain level of satisfaction if the structure of the web service signatures and the textual description matches perfectly. In this case, using equation 4 and 5, the total similarity between a pair of web services would be 0.68. Due to a logical failure, it shows a great decline in the overall similarity as compare to the total similarity of case 1.

Case 4:

It might be the case where two web services exactly match for logical similarity but do not have any similarity for structure, textual description and web service name. Since available web service has the same logical concepts as the requested web service, it

would hardly meet consumer needs because the structure of parameters that a consumer is expecting would not be supported by the candidate service. The total similarity in this case is 0.35, which shows that the candidate service would only provide 35% satisfaction to the web service consumer.

3.7 Summary

In this Chapter, we have discussed the signatures and specification similarities of a web service i.e. logical similarity, structural similarity, textual-description similarity and name similarity. In logical signature matching, instead of computing logical similarity at service level, we are calculating it at service parameters level. This approach provides a fine-grained logical filter and allows many candidate services to participate in the selection process. Hence, it provides a consumer with more alternative options if one selected service fails or become unavailable. For structural signature similarity, we are using a Structure Preserving Semantic Matching (SPSM) algorithm that is provided by S-match framework. In signature matching, along with logical matching, we are also considering the structure of signatures to be similar with the consumer request. Logical and non-logical signature matching collectively assures the similarity of signatures between a pair of web services. The service specifications (i.e. textual description and service name) are also considered in our hybrid model. Textual description of a web service contains the detail description of behavior of a web service in natural language. In textual description matching, instead of converting text into a bag of words, we have exploited its syntactic structure and have computed the semantic structural similarity

between two sentences, because the meaning of sentence is not only in the words but also in the structural way the words are combined. For web service name matching, we have defined our decomposing rules to refine and extract the useful terms from a service name. The similarity between the terms of a vector is determined using semantic cosine similarity. S^5 Web Service Matchmaker computes all signatures and specifications similarities individually and finds the cumulative similarity using the above equation 4 and 5.

Chapter 4. Evaluations and Results

In this Chapter, we present an evaluation of our service discovery model i.e. S^5 Web Service Matchmaker. We performed a series of experiments using the standard OWLS-TC dataset that is routinely used by researchers to evaluate the performance of web service discovery models. We use the OWLS-MX matchmaker [4] as a benchmark for our model, and compare our model's performance with it. It may be noted that OWLS-MX [4] entails a similar hybrid approach for a semantic web service matchmaker that performs logical-based reasoning and content-based matching.

The evaluation experiments were carried out on 150 web service descriptions which were randomly selected from the OWLS-TC dataset. Randomly selected web services covered seven different domains: education, medical care, food, travel, communication, economy, and weapons. We tested our service discovery model for seven different queries. We then compared our results with OWLS-MX for the same set of queries.

We divided our experiment into two phases:

1. *Phase I* - Logical OWLS-MX versus Logical S^5 Web Service Matchmaker: In this phase, we compared the logical components of both models i.e. S^5 Web Service Matchmaker and OWLS-MX to show the efficiency of our fine-grained logical approach. This comparison is based purely on the signatures of the web services. In the case of OWLS-MX, a semantic description of a web service signature (i.e. OWL-S) is only exploited for web service discovery as a source of functional

information. OWLS-MX have adopted logical signature matching at the web service level i.e. two web services should have the same degree of logical filter for all input/output parameters. This strict matching similarity measure results in the selection of a few candidate web services. Hence, there are very limited options for the consumer to select other services if a selected service fails. Our pure logical signature matching component builds on the assumption that consumers should have more similar web service options when a web service fails or when a consumer is building a composite web service. In phase I, we present the performance of our logical signature similarity approach and show the success of retrieving relevant web services for fine-grained logical matching.

Phase II – In most of the discovery models, the logical signature similarity measure is coupled with other similarity measures to improve the selection of the relevant web services. OWLS-MX improved the pure logical discovery model with an integration of content based information retrieval techniques. As a result, they achieved improvement in the number of relevant retrieved web services. We claim that this improvement could be much improved by using efficient information retrieval techniques. For this purpose, we have proposed the integration of syntax-based measure for textual description, structural input/output parameters, and name similarities with the fine-grained logical matching. To evaluate the performance of our hybrid model in phase II, we compare hybrid S^5 Web Service Matchmaker with the hybrid OWLS-MX.

4.1 Dataset

OWLS-TC [16] is a collection of 1000 web services descriptions that has been solely developed to facilitate the evaluation of the performance of web service discovery algorithms. It also provides 25 test queries with their relevant datasets. For every query, there are 40-50 relevant web services. This relevant data set is a gold standard for evaluating any web service discovery model.

For our experiment, we have randomly selected 150 web services, covering seven different domains, from OWLS-TC database as a sample dataset. Only 150 web services are randomly selected to evaluate the performance of S^5 Web Service Matchmaker because the preprocessing of OWL-S files takes too much time to load each individual file separately and then applies pellet-reasoner on each service ontology file. But, during matching process, only the relationships between the concepts are computed as the reasoner was already applied in the preprocessing stage. To best show the performance of result, we give an equal chance to every web service and randomly select only 150 web services. We have also selected 7 different queries. Each query represents a different domain which covers all the seven domains. The reason we have chosen 7 queries is to show the effectiveness of S^5 Web Service Matchmaker model in all the seven different domains. Our randomly selected test dataset also contains the relevant web services for each of seven queries. The total number of relevant web services for each query is given in Table 4.1.

	Query Web Services	Total Relevant Set
Query 1	Car1PersonBicyclePriceServie	17
Query 2	Comedy Film finder service	5
Query 3	HikingSurfingDestination	9
Query 4	GroceryStoreFoodService	5
Query 5	UniversityLectureService	18
Query 6	InvestigatingFinding	5
Query 7	GovernmentMissileFundingService	8

Table 4.1: Total number of relevant web services.

Top 20 retrieved web services, that satisfy user requests, are selected in all experiments. Only top 20 ranked web services are selected because the maximum relevant web services for a query in our sample dataset are 18. In an ideal condition, these 18 relevant web services could be placed at top in the list.

4.2 Evaluation Criteria

Precision and Recall are the common evaluation measures for determining search performance. They are calculated based on the relevant dataset.

- Precision is a measure to determine a system's ability to show only relevant items. This can be shown mathematically as:

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

- Recall is a measure to determine a system's ability to show all relevant items. It can be calculated as:

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

In our experiments, we have used the precision and recall as a measure to evaluate the performance of S^5 Web Service Matchmaker. As our goal is to facilitate the consumers by providing more options of partially/fully matched web services to automate web service discovery process and to select another web service in case a selected service fails, we focus on having high recall at a cost of minimum precision. We may note a significant change in precision and/or recall depending on each of the four similarity measures:

- We believe that there would be a significant increase in the recall of S^5 Web Service Matchmaker using fine-grained logical approach. This is because our fine-grained logical approach is flexible in finding the logic-based similarity between services' signatures. Any web service that shows even partial match with the user query would be selected as relevant which results in a significant increase in recall, at a minimum cost of precision. Increase in recall means more relevant web services, which lead to more alternate options for consumers in the case of a service failure or a service composition.
- Syntax-based measure for textual-description matching helps in discovering services that are syntactically similar with respect to their textual-description. Moving from the content-based IR approach to a syntax-based similarity approach may give high precision and high recall. This is because, in textual-description

similarity, the similar services are selected based on their syntax similarity as well as semantic similarity.

- Structure Preserving Semantic Matching (SPSM) algorithm is first time used for determining the signatures' structure similarity in web service discovery process. It is expected to give above 80% both precision and recall as this algorithm computes the structural similarity by considering the semantics of a tree-like graph while preserving its structure.
- As compared to previously proposed decomposing rules for web service name, we propose a few more rules to split a web service name into a vector of meaning terms. The more meaningful the terms are the more efficient would be the semantic matching of terms; hence positively affect the precision and recall of discovery model.

4.3 Phase I: Logical OWLS-MX versus Logical S^5 Web Service Matchmaker

In signature matching, one of the main contributions of S^5 Web Service Matchmaker is the fine-grained logical matching. In phase *I*, we evaluated the performance improvement of the logical similarity component of S^5 Web Service Matchmaker comparing with the pure logical similarity component of OWLS-MX. Both the logical components are purely based on the logic-based similarity of signatures of a web service (i.e. input/output concepts). This comparison was performed to show that at a cost of small precision, a significant increase in the recall is achieved by adopting fine-grained logical matching approach. Although fine-grained logical similarity measure also retrieves partially

matched web services but these partially matched web services could contribute in the process of web service composition.

For comparison purpose, we selected the logical component of OWLS-MX and applied the seven different queries one at a time and recorded the list of retrieved web services, as shown in Table 4.2. The retrieved results are the set of web services that are marked as relevant by OWLS-MX. In OWLS-MX, the degree of similarities ranges between 0 – 1 (EXACT (1.0), PLUG_IN (0.8), SUBSUME (0.7), SUBSUME-BY (0.6) and FAILED (0)). As shown in the below table, OWLS-MX retrieved only three relevant web services from the OWLS-TC dataset that showed some degree of logical similarity based on five logical filters with the Query 1(i.e. 'Car1PersonBicyclePriceService').

We applied the same queries and test dataset to the fine-grained logic-based similarity measure of S^5 Web Service Matchmaker. The pure logical S^5 Web Service Matchmaker processed all seven queries separately and returns the ranked list of relevant web service. Table 4.3 shows the number of relevant web services S^5 Web Service Matchmaker retrieved out of the 150 available web services. It is important to note that in order to compare our result with OWLS-MX, we followed the same scoring scale as used by OWLS-MX for each of the logical filters (i.e. EXACT (1.0), PLUG_IN (0.8), SUBSUME (0.7), SUBSUME-BY (0.6) and FAILED (0)).

	Query Web Services	OWLS-MX No. of Retrieved Web Services	S^5 WS Matchmaker No. of Retrieved Web Services
Query 1	Car1PersonBicyclePriceServie	3	12
Query 2	Comedy Film finder service	1	7
Query 3	HikingSurfingDestination	2	5
Query 4	GroceryStoreFoodService	2	6
Query 5	UniversityLectureService	3	9
Query 6	InvestigatingFinding	2	5
Query 7	GovernmentMissileFundingService	2	10

Table 4.2: Retrieved results from Logical OWLS-MX.

4.3.1 Analysis

In a comparison between the logical components of both matchmakers, we note that there is a significant change in the number of retrieved results for each query. The results produce by S^5 Web Service Matchmaker show that it retrieves all those relevant web services that were selected by OWLS-MX with the addition of some other relevant web services that were not picked-up by OWLS-MX. The fine-grained logical matching increases the retrieval of the web services as relevant by 350%.

As mentioned earlier, S^5 Web Service Matchmaker selects web services that also partially satisfy the consumer's need which are ignored by OWLS-MX. While on the other hand, pure logical OWLS-MX always ensures that there should be the same degree of logical similarity for each of the service's parameters. The OWLS-MX adopts the coarse-grained logical approach to ensure that the discovery model will always retrieve web services that fully satisfy all consumer needs. Any service that fulfills pure logic-based criteria gets selected and always results in high precision but low recall, as observe in Table 4.3. On

contrary, any web service that shows a small deviation from the consumer query would not be selected. OWLS-MX realized the strictness of the pure logic-based similarity approach; hence integrated the content-based similarity measure to make the discovery model more effective. The low recall in OWLS-MX shows that there are many other relevant web services that still need to be retrieved. To increase the recall as well as precision of discovery models, we propose a fine-grained logical signature similarity approach that facilitates the consumers in two ways. First, there should alternative options for a consumer if a selected service fails. Second, many partially matched services helps the consumer in developing a composite web service.

For example, consider Query 1, "Car1PersonBicyclePrice_Service", which takes inputs "Car" and "1PersonBicycle", and returns "Price". This query is compared with an available web service in OWLS-TC, e.g. "CarBiyclePrice_Service", that takes inputs 'Car' and 'Bicycle' and returns output 'Price'. OWLS-MX fails to match 'CarBiyclePrice_Service' with query 1 because there exists an EXACT degree of similarity for the parameters 'Car' and 'Price' but a SUBSUME-BY degree of similarity between the parameters 'Bicycle' and '1PersonBicycle'. The logical similarity computes the SUBSUMES-BY degree of similarity because concept '1PersonBicycle' is the direct child of concept 'Bicycle'. Contrary to OWLS-MX, S^5 Web Service Matchmaker also retrieves all the same subsumption relations and it calculates the overall similarity by averaging the individual similarities. The total similarity score for the candidate service "CarBicyclePrice_Service" is 0.9. This similarity score shows that the candidate web service could satisfy consumer needs by 90% which is better than totally ignoring the

web service. In this case, for example, if a consumer was looking for a service that gave the total price of a car and one person bicycle, other service with inputs 'Car' and 'bicycle' could also provide nearly maximum satisfactory results.

Similarly, consider a service consumer requests for a service, for example 'StudentRegistration', which takes input 'Name', 'DOB', and 'Address' and register a student. In OWLS-TC data set, there is also a web service 'StudentReg_Service' that takes 'LastName', 'DOB, and 'Address' that perform the same operation. OWLS-MX would fail to retrieve service 'StudentReg_Service' because all the input parameters have an EXACT degree of similarity except the parameter 'Name' that has a PLUG-IN logical degree of similarity with input 'LastName'. Note that, in domain ontology, concept 'LastName' is the subclass of the concept 'Name'. On the contrary, S^5 Web Service Matchmaker considers all types of relationships in computing similarity and retrieves the service 'StudentReg_Service' with logical score of 0.96. This similarity score shows how close the candidate service 'StudentReg_Service' is to the user query and it is expected that this service would provide 96% user requirements.

Table 4.3 shows the precision, recall, and accuracy of each query for OWLS-MX and S^5 Web Service Matchmaker.

		OWLS-MX			S^5 Web Service Matchmaker		
	Query Name	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**
Query 1	Car1PersonBicyclePriceServie	1.00	0.17	0.91	0.91	0.64	0.95
Query 2	Comedy Film finder service	1.00	0.20	0.97	0.60	0.80	0.97
Query 3	HikingSurfingDestination	1.00	0.20	0.95	1.00	0.55	0.97
Query 4	GroceryStoreFoodService	1.00	0.40	0.98	0.50	0.80	0.98
Query 5	UniversityLectureService	1.00	0.16	0.90	1.00	0.50	0.94
Query 6	InvestigatingFinding	1.00	0.40	0.98	0.60	0.60	0.97
Query 7	GovernmentMissileFunding Service	1.00	0.25	0.96	0.70	0.88	0.97

Table 4.3: Precision/Recall/Accuracy for Logical OWLS-MX and S^5 Web Service Matchmaker.

By adopting the fine-grained logical signature matching, it is always expected that there are chances of selecting irrelevant web services as it selects any web service that partially satisfy consumer needs. The important point to note from the results is that the increase in recall is achieved at a minimum cost of precision. If a consumer is looking for exactly matched services then OWLS-MX and S^5 Web Service Matchmaker both could satisfy his requirements. But in a scenario, such as web service composition, where a consumer looks for many alternative options of web services which could partially satisfy his needs; S^5 Web Service Matchmaker is the best choice.

We can conclude from the Table 4.4 that there is 2% increase in the accuracy of logical S^5 Web Service Matchmaker as compared to logical OWLS-MX, which is due to the fine-grained logical approach. Although there is not significant improvement in the accuracy of logical S^5 Web Service Matchmaker but we note that, at a little cost of precision, we have achieved a significant improvement in recall. This increase in recall, while keeping precision same, is resulted from retrieving fine-grained logical matching.

We note that with the increase in more relevant web services, a service consumer has more options to select another web service if a web service fails. Having more relevant web services will also contribute to web service composition where two or more web services are required to compose a service. If we see the results of logical OWLS-MX, we notice that the precision of the relevant web services is 100%. The reason is that any web service that shows a slightly mismatch in parameters would not be considered in the selection. Logical OWLS-MX always assures that any web service that would be retrieved as relevant should always fully satisfy the consumer requirements. Due to the strict logical matching, precision for logical OWLS-MX is always high with low recall. We proposed a fine-grained logical matching approach, mentioned in Chapter 3, in which we compute the similarity at each parameter level instead of each web service level. The reason is that even if a service does not fully satisfy the consumer requirements, it might be possible that this service will contribute in the web service composition process. Figure 4.1 shows the significant change in performance of the S^5 Web Service Matchmaker that is achieved by adopting a fine-grained logical approach.

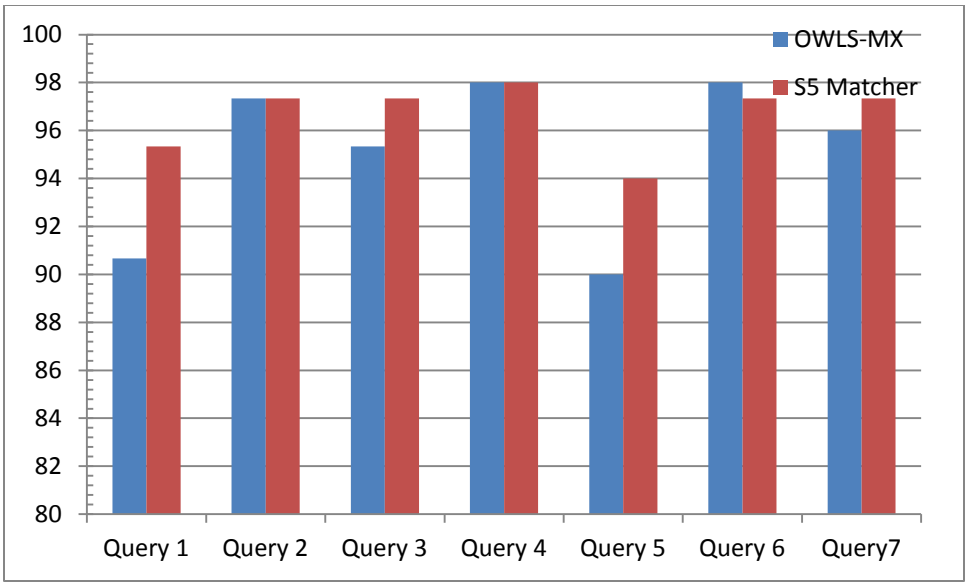

Figure 4.1: Comparison of Accuracies of Logical OWLS-MX and Logical S^5 Web Service Matchmaker.

To further analyze the effectiveness of fine-grained logical matching of S^5 Web Service Matchmaker, we show the experiment results of both matchmakers for Query1 (i.e. Car1PersonBicyclePrice) in Table 4.4. It is noted that there were 17 relevant web services in the test dataset for Query1. OWLS-MX retrieved only 3 relevant web services out of 17 relevant services with a 100% precision and 17% recall. On the other hand, S^5 Web Service Matchmaker retrieved 12 web services in which 11 were relevant and 1 irrelevant web service with a precision of 91% and recall 64%. If we consider only this query we can analyze that at a cost of 9% precision, S^5 Web Service Matchmaker provided an almost 50% increase in recall.

Query	Relevant	OWLS-MX (Retrieved)	S^5 Web Service Matchmaker (Retrieved)
Car1Person BicyclePrice	Car1PersonBicyclePrice	Car1PersonBicyclePrice	Car1PersonBicyclePrice
	CarCyclePrice	Kohl Car1PersonBicyclePrice	Kohl Car1PersonBicyclePrice
	Kohl Car1PersonBicyclePrice	4WheeledCar1PersonBicyclePrice	4WheeledCar1PersonBicyclePrice
	Bicycle4Wheeledcar_Price		Bicycle4Wheeledcar_Price
	Vehicle price		Car2PersonBicyclePrice
	T-car price		CheapCar 2PersonBicyclePrice
	4WheeledCar1PersonBicyclePrice		2PersonBicycle4Wheeledcar_Price
	Auto Year price		4wheeledcar year price
	Recommended price of car model		4wheeledcar year price report
	4wheeledcar year price		Recommended price of car model
	leynthu rent a car		T-car price
	Car2PersonBicyclePrice		CarCycleTaxPrice
	2PersonBicycle4Wheeledcar_Price		
	car price report		
	car price		
	4wheeledcar year price report		
	CheapCar 2PersonBicyclePrice		
Precision		1.00	0.91
Recall		0.17	0.64
Accuracy		0.91	0.95

Table 4.4: Results of logical OWLS-MX and S^5 Web Service Matchmaker

4.4 Phase II: Hybrid OWLS-MX versus Hybrid S^5 Web Service Matchmaker

In phase II, we performed a comparison between the hybrid models of OWLS-MX and S^5 Web Service Matchmaker, where the hybrid models integrate the logical, contextual and structural similarity approaches. OWLS-MX performs both logical similarity and content-based information similarity. If two web services logically fail to match using the logical filters, then content-based information retrieval techniques that involve the

application of cosine, or jaccard on the vectors of terms retrieved from textual-descriptions, service names, and input/output concepts are applied.

To evaluate the performance of the hybrid S^5 Web Service Matchmaker as compares to the hybrid OWLS-MX, we used the same dataset with same queries as in phase I. To check the relevancy of the results, Top 20 retrieved web services were selected from both hybrid. The threshold of web service similarity score was set to 0.44 for both hybrid models. The reason was because, for S^5 Web Service Matchmaker, if our model fails completely in signatures matching, then the maximum collective score of specifications would be 0.44. If the signature matching fails then there is no need to consider a web service based on their specifications similarities. The number of relevant web services retrieved by each matchmaker is shown in Table 4.5.

	Query Web Services	**OWLS-MX No. of Retrieved Web Services**	**S^5 Web Service Matchmaker No. of Retrieved Web Services**
Query 1	Car1PersonBicyclePriceServie	8	20
Query 2	Comedy Film finder service	14	12
Query 3	HikingSurfingDestination	5	18
Query 4	GroceryStoreFoodService	4	8
Query 5	UniversityLectureService	9	19
Query 6	InvestigatingFinding	9	9
Query 7	GovernmentMissileFundingService	13	11

Table 4.5: Results from hybrid OWLS-MX and S^5 Web Service Matchmaker.

In phase *II*, there is a significant increase in the retrieved results of both methods as compared to phase *I* which only involved the use of logical methods. This increase is due to the fact that if logical matching between a pair of web services fails; both models

exploit other web service information to compute web service similarities by structure, textual-description and service name matching.

4.4.1 Analysis

By integrating a logic-based signature similarity measure with other non-logical similarity measures, we can clearly see an increase in the number of web services retrieved. The goal is not to retrieve more web services but the more of the relevant ones. In order to check the relevancy of retrieved results, the important thing to note here is that how many relevant web services are retrieved, that can be evaluated by computing precision, recall and accuracy, as shown in Table 4.6.

		OWLS-MX			S^5 Web Service Matchmaker		
	Query Name	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**
Query 1	Car1PersonBicyclePriceServie	1.00	0.40	0.94	0.85	1.00	0.98
Query 2	Comedy Film finder service	0.30	1.00	0.94	0.40	1.00	0.95
Query 3	HikingSurfingDestination	1.00	0.55	0.97	0.50	1.00	0.94
Query 4	GroceryStoreFoodService	0.75	0.60	0.98	0.60	1.00	0.98
Query 5	UniversityLectureService	0.77	0.38	0.91	0.73	0.77	0.94
Query 6	InvestigatingFinding	0.44	0.80	0.96	0.56	1.00	0.97
Query 7	GovernmentMissileFunding Service	0.46	0.75	0.94	0.73	1.00	0.98

Table 4.6: Precision/Recall/Accuracy of hybrid OWLS-MX and S^5 Web Service Matchmaker.

With the integration of a content-based information retrieval technique in logical OWLS-MX, it is noted that the overall precision of the model is decreased. This is due to the fact

that some irrelevant web services are now selected because the string similarity measures have been applied on their content. For example, in pure logical matching, "CarCycle_Price_Service" was not retrieved because of a logical matching failure. However in a hybrid OWLS-MX, a content similarity of 0.46 is computed which therefore makes the service relevant for the service consumer. Similarly, S^5 Web Service Matchmaker also shows the increase in relevant retrieved results. With the integration of structure, textual-description, and name matching, we have achieved 97% recall with 63% precision as compared to hybrid OWLS-MX where it has 65% recall and 68% precision. The important point is that these experiment results are from the top 20 retrieved web services which have the similarity score above the threshold (i.e. 0.44). We realize that mostly the precision of the S^5 Web Service Matchmaker is always less than the precision of the OWLS-MX which means that the hybrid OWLS-MX still provides the consumers with the most suitable web services. But we also realize that there are many relevant web services that are not retrieved by hybrid OWLS-MX which results in low recall. On the other hand, the approaches adopt in S^5 Web Service Matchmaker help in retrieving those relevant web services that are missed by OWLS-MX. It is also noted that S^5 Web Service Matchmaker retrieved all those relevant web services that are retrieved by OWLS-MX with an addition of other relevant web services. We also see an overall increase of 1% accuracy which shows that at a cost of precision (i.e. 5%) we have significantly improved the recall (i.e. 32%) of S^5 Web Service Matchmaker. It also shows that with a slight decrease in precision, we can provide a service consumer with 96% relevant web services that give more selection options and help in the web service composition process and the search refinement. This supports our assumption that

services partially/fully satisfy user requirements should be retrieved to facilitate web service composition process. A comparison of the accuracy of both models is shown in Figure 4.2 which shows that on average there is an improvement in the performance of S^5 Web Service Matchmaker.

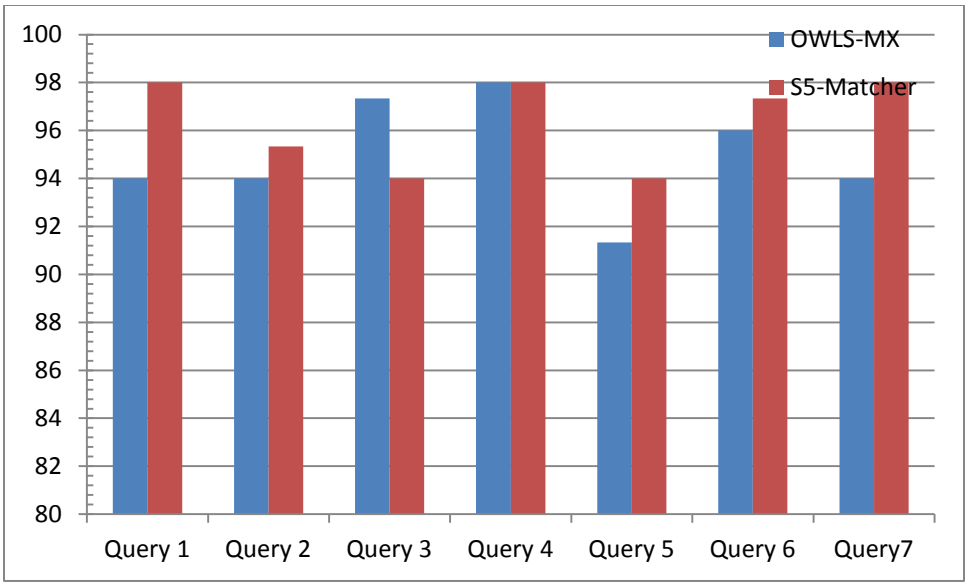

Figure 4.2: Comparison of Accuracies of hybrid OWLS-MX and hybrid S^5 Web Service Matchmaker.

In Table 4.7, we have shown the retrieved results from both hybrid models for the Query 1 (i.e. Car1PersonBicyclePrice). As compared to the results of Table 4.5, we could see an increase in the relevant results for both hybrid models. In case of the hybrid approach, we can conclude from the Table 4.7 that OWLS-MX gives high precision for Query 1 with a very low recall. This is due to the content-based matching in which the content of the

web services are taken into the consideration when the pure logical matching between input/output parameters is failed. It is also noted that there are many relevant web services that are missed by OWLS-MX either because there is no syntactic match between the contents of web services or the content similarity score is less than the threshold. On the other hand, we can see an increase in the hybrid S^5 Web Service Matchmaker results where the services, which were not retrieved by fine-grained logical matching in phase I, are now retrieved by integrating structure, textual-description, and name similarity measures. Although for Query 1, we note that the precision is affected 15% in S^5 Web Service Matchmaker but there is significant increase in recall, which is 100% for Query 1. S^5 Web Service Matchmaker retrieved all the relevant web services from test dataset with an addition of 3 irrelevant web services, as shown in Table 4.7.

Query	Relevant	OWLS-MX (Retrieved)	S^5 Web Service Matchmaker (Retrieved)
Car1Person BicyclePrice	Car1PersonBicyclePrice	Car1PersonBicyclePrice service	Car1PersonBicyclePrice service
	CarCyclePrice	CarCyclePrice	CarCyclePrice
	Kohl Car1PersonBicyclePrice	Kohl Car1PersonBicyclePrice	Kohl Car1PersonBicyclePrice
	Bicycle4Wheeledcar_Price_	4WheeledCar1PersonBicyclePrice	Bicycle4Wheeledcar_Price_
	Vehicle price	4wheeledcar year price	Vehicle price
	T-car price	Car2PersonBicyclePrice	T-car price
	4WheeledCar1PersonBicyclePrice	2PersonBicycle4Wheeledcar_Price	4WheeledCar1PersonBicyclePrice
	Auto Year price	4wheeledcar year price report	Auto Year price
	Recommended price of car model		Recommended price of car model
	4wheeledcar year price		4wheeledcar year price
	leynthu rent a car		leynthu rent a car
	Car2PersonBicyclePrice		Car2PersonBicyclePrice
	2PersonBicycle4Wheeledcar_Price		2PersonBicycle4Wheeledcar_Price
	car price report		car price report
	car price		car price
	4wheeledcar year price report		4wheeledcar year price report
	CheapCar 2PersonBicyclePrice		CheapCar 2PersonBicyclePrice
			CarCycleTaxPrice
			BookPrice
			TR NovelPersonPrice
Precision		1.00	0.85
Recall		0.40	1.00
Accuracy		0.94	0.98

Table 4.7: Experiment results of Hybrid OWLS-MX and S^5 Web Service Matchmaker.

In phase II, we compared the hybrid OWLS-MX with our hybrid proposed model. In these experiments, we improved the performance by exploiting and integrating the signatures and specifications of a web service. Those relevant web services that were missed in pure logical matching, because of a missing relationship between concepts or incomplete domain ontology, are now retrieved by the integration of structural and textual descriptions similarity measures.

4.6 Impact of Web Service Textual Description

The textual description is one of the important specifications of a web service that describes the functional and non-function description of a web service in natural language. We argue that an efficient similarity measure for textual descriptions of services could facilitate the discovery process. To this end, we have exploited the structure of a sentence rather than relying only on the meaning of the individual words. Our main contribution is proposing a syntax-based measure for semantic similarity of textual-description which exploits the syntactic and semantic information of a sentence. To evaluate the performance of our proposed service description matching method, we have performed a comparison of our proposed approach with the traditional information retrieval techniques (i.e. cosine similarity).

In this phase, we have shown the performance improvement achieved by proposing a syntax-based measure for semantic similarity for the textual description of a web service. In order to evaluate the performance, we have compared syntax-based measure for semantic similarity approach with traditional information retrieval techniques (i.e. cosine similarity). We ran two different experiments i.e. pure logical S^5 Web Service Matchmaker with traditional information retrieval technique (S^5 Web Service Matchmaker I) and pure logical S^5 Web Service Matchmaker with syntax-based measure for semantic similarity technique (S^5 Web Service Matchmaker II). The reason we compared different textual description approaches combined with logical signature

matching was that textual descriptions matching does not provide any substantial results if logical signature matching totally fails. To perform our experiments, first we ran seven different queries on the S^5 Web Service Matchmaker I on the dataset of 150 available web services using equation 6:

$$SIM_1(R, S) = 0.7 \, (Logical\_SIM(R, S)) + 0.3*Textual\_SIM(R, S) \qquad (equation\ 6)$$

A similar experiment was performed for the S^5 Web Service Matchmaker II, with the same dataset and using equation 6. The threshold of similarity score was set to 0.3. The reason was that if the logical signature similarity between a pair of services fails, the candidate service should not be selected as relevant even if its specifications exactly match. Below, we have shown the impact of our syntax-based measure for semantic similarity approach in a web service discovery. In order to compare the results of our proposed textual description matching, the precision, recall, and accuracy measures are used to evaluate the performance. Top 20 web services are selected to evaluate the effectiveness of both approaches. Table 4.8 shows a comparison of hybrid S^5 Web Service Matchmaker I (i.e. logical similarity and content-based similarity) and hybrid S^5 Web Service Matchmaker II (i.e. logical similarity and syntax-based textual-description similarity).

	Query Name	S^5 Web Service Matchmaker I (logical + traditional IR)			S^5 Web Service Matchmaker II (logical + Syntax-based Matching)		
		Precision	Recall	Accuracy	Precision	Recall	Accuracy
Query 1	Car1PersonBicyclePriceServie	0.75	0.88	0.95	0.85	1.00	0.98
Query 2	Comedy Film finder service	0.25	1.00	0.91	0.40	1.00	0.94
Query 3	HikingSurfingDestination	0.40	0.88	0.91	0.50	1.00	0.94
Query 4	GroceryStoreFoodService	0.22	0.80	0.94	0.60	1.00	0.97
Query 5	UniversityLectureService	0.70	0.77	0.93	0.73	0.77	0.95
Query 6	InvestigatingFinding	0.38	1.00	0.95	0.50	1.00	0.97
Query 7	GovernmentMissileFunding Service	0.47	0.88	0.94	0.67	1.00	0.97

Table 4.8: Precision/Recall/Accuracy of hybrid OWLS-MX and S^5 Web Service Matchmaker.

We see a high difference in the number of the retrieved web services for the both hybrid models as compare to the results of phase I. S^5 Web Service Matchmaker I shows a large number of web service selections because of the traditional string similarity measure. Even if the meaning of a sentence is totally different, the similar words in both sentences give a higher textual similarity score, no matter what the syntactic structure is. S^5 Web Service Matchmaker I results in retrieving more web services as relevant. On the contrary, S^5 Web Service Matchmaker *II* retrieves a few irrelevant web services for each query as compared to S^5 Web Service Matchmaker I. This is due to the textual description matching in which services having textual descriptions with same words but different syntactic structure are not retrieved. For example, a web service 'TR NovelPersonPrice'' was selected by S^5 Web Service Matchmaker *I* for query 1 because it showed a score of 0.25 logical similarity but the textual description similarity ranked it high with a score of 0.60 similarity, resulting in overall similarity of score 0.36. As the overall similarity was above the threshold, the service 'TR NovelPersonPrice' was

retrieved as a relevant web service. On the other hand, S^5 Web Service Matchmaker II, that adopted the syntax matching approach for textual description, computed textual similarity score of 0.25 which resulted in overall similarity score of 0.25 and marked as irrelevant. Our results show an improvement of 15% precision and 8% recall for the syntax-based measure for semantic similarity technique as compare to traditional information retrieval techniques. By adopting a syntax-based measure for semantic similarity of a textual description, we have achieved an increase of 3% accuracy. The overall improvement of our proposed approach for a textual description is shown in term of accuracies in Figure 4.4.

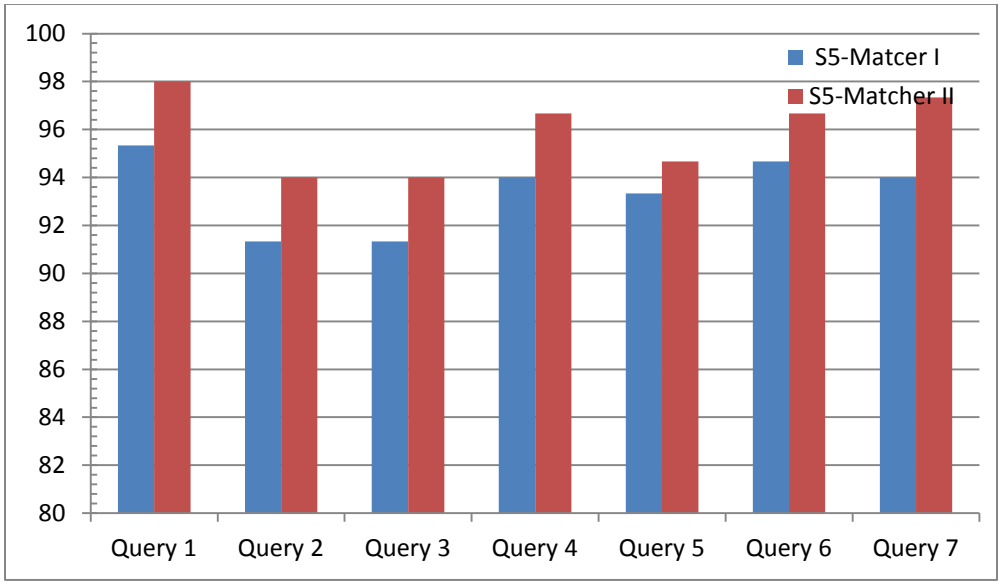

Figure 4.3: Comparison of Accuracies of S^5 Web Service Matchmaker I and II.

To further analyze the impact of syntax-based measure for textual description similarity, we present the experiment results of Query 1 (i.e. Car1PersonBicyclePrice) in Table 4.9. For Query 1, Top 20 web services which are retrieved from the S^5 Web Service

Matchmaker *I* shows 88% recall, which was 64% in pure fine-grained logical matching shown in Table 4.4. This is because the overall similarity of many web services, that had logical similarity score less than threshold (i.e. 0.3), have increased by an integration of the information retrieval technique for text-description similarity; hence results in an addition of five irrelevant web services. On the other hand, integration of syntax-based measure for semantic similarity of textual-description with fine-grained logical matching also results in more relevant web services as compared to S^5 Web Service Matchmaker *I*. But we can see that by exploiting the syntax of a textual-description for discovering web services, we have retrieved all the 17 relevant web services for Query 1 at an 85% precision, which were only 15 relevant webs services for S^5 Web Service Matchmaker *I*.

Query	Relevant	S^5 Web Service Matchmaker I (Retrieved)	S^5 Web Service Matchmaker II (Retrieved)
Car1Person BicyclePrice	Car1PersonBicyclePrice	Car1PersonBicyclePrice	Car1PersonBicyclePrice
	CarCyclePrice	Kohl Car1PersonBicyclePrice	Kohl Car1PersonBicyclePrice
	Kohl Car1PersonBicyclePrice	4WheeledCar1PersonBicyclePrice	4WheeledCar1PersonBicyclePrice
	Bicycle4Wheeledcar_Price_	2PersonBicycle4Wheeledcar_Price	CarCyclePrice
	Vehicle price	Car2PersonBicyclePrice	Car2PersonBicyclePrice
	T-car price	CheapCar 2PersonBicyclePrice	CheapCar 2PersonBicyclePrice
	4WheeledCar1PersonBicyclePrice	4wheeledcar year price	2PersonBicycle4Wheeledcar_Price
	Auto Year price	Bicycle4Wheeledcar_Price_	Bicycle4Wheeledcar_Price_
	Recommended price of car model	3wheeledcarYearRecommended price	Vehicle price
	4wheeledcar year price	Vehicle price	T-car price
	leynthu rent a car	T-car price	4wheeledcar year price
	Car2PersonBicyclePrice	4wheeledcar year price report	leynthu rent a car
	2PersonBicycle4Wheeledcar_Price	car price	car price report
	car price report	car price report	Recommended price of car model
	car price	CarCyclePrice	Auto Year price
	4wheeledcar year price report	Auto Year price	car price
	CheapCar2PersonBicyclePrice	CarCycleTaxPrice	4wheeledcar year price report
		AuthorNovelPrice	CarCycleTaxPrice
		TR NovelPersonPrice	BookPrice
		BookPrice	AuthorNovelPrice
Precision		0.75	0.85
Recall		0.88	1.00
Accuracy		0.95	0.98

Table 4.9: Experiment results of S^5 Web Service Matchmaker I and II.

4.7 Conclusion

Based on the signatures and specifications of web services, S^5 Web Service Matchmaker propose the logical and the non-logical approaches to exploit web service descriptions semantically in order to provide an effective and efficient web service discovery model. The flexibility of S^5 Web Service Matchmaker is that it allows the consumer to discover

web services depending on either pure logical approach or hybrid approach. Figure 4.4 and 4.5 shows the comparison of performance measures (i.e. precision and recall) for pure logical S^5 Web Service Matchmaker and hybrid S^5 Web Service Matchmaker.

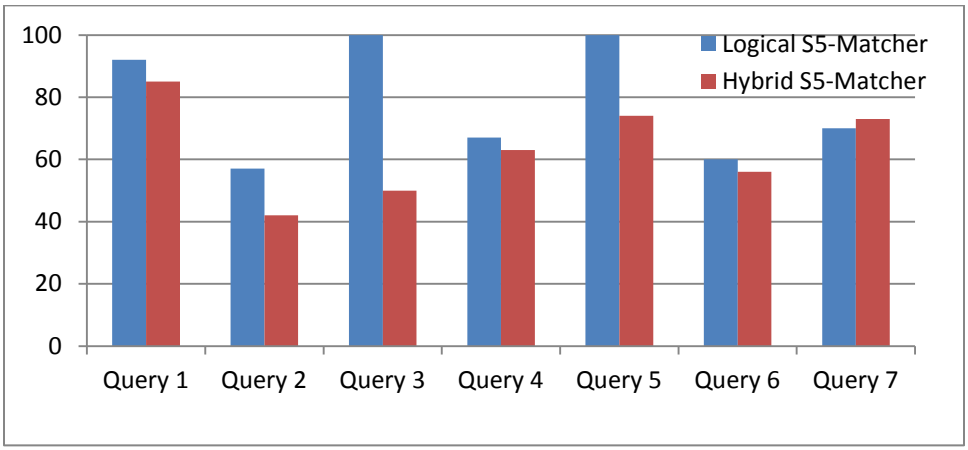

Figure 4.4: Precision comparison of logical and hybrid S^5 Web Service Matchmaker.

Figure 4.4 shows the precision comparison of the logical and hybrid S^5 Web Service Matchmaker based on the results of phase I and phase II. It is noted that the precision of logical S^5 Web Service Matchmaker is always higher than the hybrid S^5 Web Service Matchmaker. This means that logical S^5 Web Service Matchmaker would always provide relevant web services to the consumer; no matter what the recall would be. Based on this comparison, we can suggest that if a consumer is looking for exactly similar web services then logical S^5 Web Service Matchmaker is a best option. A consumer could also use the logical S^5 Web Service Matchmaker if the structure and textual-description of a request service is not available.

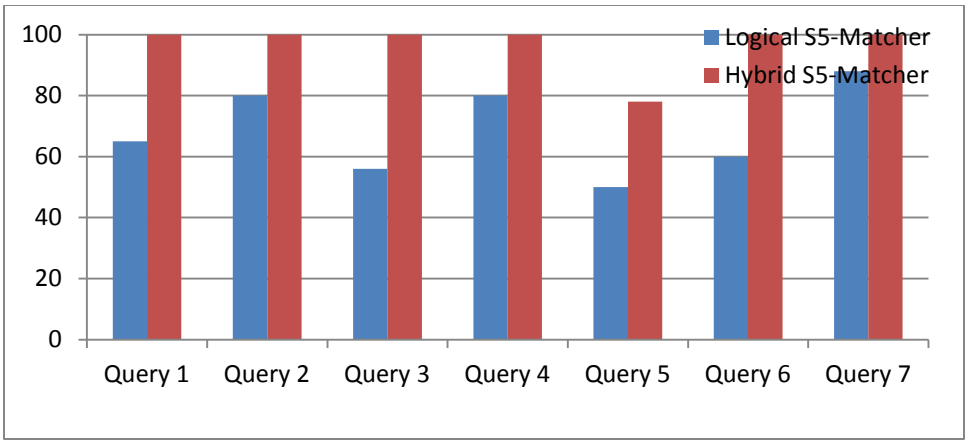

Figure 4.5: Recall comparison of logical and hybrid S^5 Web Service Matchmaker.

From Figure 4.5, we can conclude that the recall of the hybrid S^5 Web Service Matchmaker is always higher (i.e. 100%) then the recall of logical S^5 Web Service Matchmaker. The reason is that the hybrid model is computing service similarity using the signatures and specifications information of web services. Based on the comparison of Figure 4.5, we can conclude the Hybrid S^5 Web Service Matchmaker provides almost all relevant web services, no matter what the precision score would be. Hybrid model could be beneficial for consumers who are interested in discovering all available relevant web service from the database. A consumer should select the hybrid model if and only if the user request and all available web services contain the specification description along with the signature description.

4.8 Summary

In this Chapter, we performed a series of experiments to evaluate the performance of our proposed approaches (i.e. fined-grained logical signature matching and syntax-based

measure for sentence matching). These approaches exploited the maximum information from a web service description to make the discovery process effective and efficient. We used OWLS-MX as a benchmark to compare our results. The relevant dataset of web services from OWLS-TC was used a gold standard. We divided our experiment into two phases. In phase I, we performed the comparison between the pure logical OWLS-MX with our proposed S^5 Web Service Matchmaker logical component. This comparison showed a significant increase in the relevant services retrieved by a pure logical S^5 Web Service Matchmaker as compare to a pure logical OWLS-MX. By adopting fine-grained logical signature matching, we achieved a 43% increase in recall at a cost of 22% precision with an overall increase of 2% accuracy. This is due to the selection of all those web services that had even a small logical signature similarity with the user request. In phase II, we compared the hybrid models. The purpose of this comparison was to show the impact of other web service similarity measures (i.e. signature's structure and specifications) on the discovery process. To evaluate the performance of our proposed approaches, we compared our model with the hybrid OWLS-MX because OWLS-MX also exploits the logic-based and content-based information from the OWL-S document. By taking the WSDL and OWL-S information of a web service and integrating signatures and specifications similarities, we achieved 1% improvement in accuracy as compare to our benchmark. This improvement in accuracy is achieved by an increase of a 32% recall at a cost of a 5% precision. These results support our approach which provides a service consumer with more relevant options when there is a service failure or to compose a web service. Finally, we showed the effectiveness of adopting syntax-based measure for semantic similarity approach for textual descriptions. Our claim is that textual description

contains behavioral information of a web service in a sentence of few words. Applying vector similarity methods for comparing two sentences is not an efficient approach because the semantics of a sentence is extracted from the individual words and the structural way the words are combined. To exploits the semantics of a sentence, we propose the syntax-based measure for semantic similarity for textual-description similarity. We compared our proposed textual approach with traditional information retrieval technique (i.e. cosine similarity as used in OWLS-MX). Our results showed 3% improvement in accuracy with an increase in 15% precision and 8% recall, which was achieved by adopting the syntax-based measure for semantic similarity for textual-description. We saw a decrease in retrieving relevant web service by adopting string similarity measures because web services having same words but different syntactic structure are also marked as relevant. This selection of irrelevant web services decreases precision and recall, which, as a result, decrease the accuracy of discovery model. Whereas the syntax-based measure for semantic similarity approach solves this problem by considering the syntactic structure of a sentence which results in improving the overall accuracy of the model.

Chapter 5. Conclusion and Future Work

5.1 Contributions

Web Service Matchmakers, developed for web service discovery, either consider the signatures (i.e. input/output parameters and their structures) or specifications (i.e. textual descriptions and web service names). WSDL-based matchmaker considered a textual description as a vector of terms and adopted information retrieval techniques. Some matchmakers also applied the same traditional approaches on other content of a WSDL document i.e. operations, messages and data types, as mentioned in Chapter 2. As WSDL document does not provide the semantic information of a web service, OWL-S helps in the automation of web service discovery by providing the semantic information. Many service matchmakers are developed which exploit the semantic descriptions of web services and have applied different logical and non-logical techniques. Klusch [4] proposed the first logical service matchmaker (OWLS-MX) based on OWL-S description. In OWLS-MX, they proposed five logical filters which are applied at each web service signature level, i.e. a service is relevant if it exactly matches or completely subsumes the user request. Whereas services partially logically matched are ignored. OWL-S also provides the textual description of a web service in natural language. This textual description is a sentence of a few words. All discovery models that had exploited textual description treated this behavioral information as a bag of a words and applied

traditional information retrieval techniques. This resulted in completely ignoring the syntactic structure of a sentence.

We focus on the shortcomings of previously developed approaches for web service discovery. We propose a hybrid service discovery model, Hybrid Web Service Matcher (S^5 Web Service Matchmaker), which overcomes the shortcomings of different components of a discovery model at web service signature and specification levels. S^5 Web Service Matchmaker proposes four logical and non-logical similarities measures. For signatures matching, we propose logical and non-logical similarity measures whereas for specifications matching we propose non-logical textual-description and name similarity measures. S^5 Web Service Matchmaker integrates the maximum information available in the web service description to achieve maximum consumer satisfaction. Our main goal is to focus on providing consumers with many relevant web service options which help in two ways. First, a consumer should have many alternatives to select another web service if a service fails during invocation. Second, a list of relevant web services gives more opportunity to a consumer to compose a complex web service, which results in developing new web services to achieve consumer tasks. To achieve these goals, we propose a fine-grained logical signature matching, signature's structure matching, and syntax-based measure for semantic matching. We also define some new decomposing rules to split a web service name into meaningful words, which helps in service name comparison.

The different components of our hybrid model and their needs are described below:

1. *Logical Signature Matching*: logical signature matching plays a vital role in achieving our goals. Instead of performing a strict logical matching on service parameters, we propose a fine-grained logical matching. The idea is to select even those web services that show very little logical similarity with consumer request. This is because two or more partially matched services could facilitate in the web service composition process to provide maximum satisfaction to a consumer. Fine-grained logical signature matching results in discovering more web services as relevant, which is contrary to the former approach where only exactly matched or fully subsumed web services were retrieved.
2. *Textual-Description Matching*: Textual-description of a web service, provided by a service publisher, provides the behavioral information of a service. Distinguishing a web service based on textual description may be helpful if it is exploited efficiently. We see that the previously adopted approaches for a textual-description similarity measure did not extract the maximum semantic information. All the proposed approaches treated a textual description as a bag of words and applied traditional information retrieval techniques (i.e. cosine, jaccrad, etc.). The drawback is that two sentences that have different semantic meanings but the same words would be considered similar. The one most important problem with these approaches is that they adopted the long-text similarity approaches on a sentence of few words (i.e. TF/IDF, word co-occurrence, etc.). Our approach is inspired by the method of short sentence semantic matching [29]. We believe that exploiting the structure of a textual description would be more helpful in web service matching, instead of just applying traditional string similarity measures on

individual words. The reason is that the semantic of a sentence is extracted not only from the words but also the syntactic way the words are combined. We proposed a hybrid model that also exploits the structure of a sentence and performs semantic matching.

3. *Non-Logical Signature Matching*: As mentioned earlier, the WSDL document provides the signatures' definition and their structures. It is one of the important sources of a service description that helps to uniquely identify a web service. The structure of signatures (i.e. input/output parameters) cannot be ignored in service comparison because a service's functionality is dependent on the structure of its parameters. It is necessary for a consumer to provide exactly similar or compatible structures which let the service work perfectly and gives the desired result. Structure matching facilitates the process of selecting similar web services that structurally match with the user request. The approaches adopted for services structure matching are not satisfactory as they used simple term/data type matching for every element of a parameter structure without considering their structure. Previously developed discovery models considered an element as an individual word and applied information retrieval techniques to compute the similarity between two words. We argue that the structure similarity should be computed by semantically annotating the graph-like structure while preserving their internal structure. For this purpose, we introduce the application of a Structure Preserving Semantic Matching (SPSM) algorithm on the web service discovery model. SPSM helps in providing an efficient way of computing

parameter structural similarity, hence refining web service search and retrieving relevant services.

4. *Web Service Name*: Sometimes, a service name is also used to uniquely identify desired services. Service name matching is also considered by discovery models to select relevant web services. As the service name is composed of different words, decomposing rules are defined to split a composite name into individual words. Previously, the service name was split based on capital letters. However there could be possibility of having numbers, spaces, underscores or dashes in a web service name. We define a list of decomposing rules that works for different name formats. These decomposing rules results in extracting meaningful words from a service name. Once the name is split into individual words, a string similarity measure is applied to compute the similarities between two service names.

We proposed a hybrid discovery model, S^5 Web Service Matchmaker, that integrates all the above four components. It provides an opportunity for a consumer to search the desired services either based on all four components or any specific component. We have evaluated the performance of our approach in three different ways: hybrid model, pure logical model, and textual-description based model.

We evaluated the performance of S^5 Web Service Matchmaker by conducting a series of experiments. An OWLS-TC data set was used to perform experiments. OWLS-MX was used as a benchmark for our evaluation. We used the same dataset and same set of

queries for both matchmakers. Precision/Recall/Accuracy measures were used to evaluate the performance of our model. By examining the results from both models, we can draw the following conclusions:

1. Our fine-grained logical signature matching approach showed a significant improvement in retrieving relevant web services. It supports our assumption of retrieving more web services which could facilitate the web service composition process. The important thing to note is that we have achieved a great increase in recall (i.e. 43%) with a small loss of precision (i.e. 22%). The overall accuracy of our model is increased by 2%.
2. We also noticed that our hybrid model (i.e. signatures and specifications matching) performs outstandingly as compared to hybrid OWLS-MX. As our goal was to provide consumers with more service options, we achieved that goal with an increase of 32% recall and 1% accuracy at a cost of 5% precision.
3. Lastly, we computed the impact of the syntax-based measure for semantic similarity approach for textual description in our hybrid model. To evaluate it, we performed a comparison of our proposed textual-description approach with the traditional information retrieval technique. It showed that by exploiting the structure of a sentence in text similarity, we have achieved an overall increase of 3% accuracy with an increase of 15% precision and 8% recall.

These experiments showed that our proposed approaches have given significant improvement for web service discovery. The logical and non-logical similarity

components can work individually and collectively to provide consumers with desired relevant web services.

5.2 Future Work

Based on our literature review, we tried to solve few problems that we think are the barrier toward developing an efficient semantic web service discovery process. We still believe that there are some areas in our discovery model which could bring more improvement to our discovery model, which are mentioned below:

1. For textual description matching, we are considering only 11 grammatical relations out of 52 which are related to subject, verb, and object. We can extend short sentence structure matching by in-cooperating more grammatical relations

2. Structure matching algorithm converts the nodes of graphs into concepts using WordNet dictionary. If we replace WordNet with domain ontology then it will gives more power to structure matching algorithm to compute the concept's relationship.

3. In web service name matching, we are decomposing web service name into words and applying string matching. We are planning to convert these terms into concepts and use domain ontology to compute similarity, instead of relying on WordNet.

Bibliography

1. Web Services. http://www.w3.org/TR/ws-arch/
2. Universal Description, Discovery, and Integration (UDDI). http://uddi.xml.org/uddi-101
3. OWL-S Semantic Markup for Web Service. http://www.w3.org/Submission/OWL-S/
4. Matthias Klusch, Benedikt Fries, Mahboob Khalid . OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Journal Web Semantics: Science, Services and Agents on the World Wide Web archive Volume 7 Issue 2*, April 2009.
5. Tim Berners-Lee, James Hendler and Ora Lasssila. The Semantic Web. *Scientific American: The Feature Article,* May 2001.
6. Wenjie Li, Wenjing Guo. Semantic-based Web Service Matchmaking Algorithm in Biomedicine. *International Conference on BioMedical Engineering and Informatics,* 2008.
7. Shalini Batra, Dr. Seema Bawa. Review of Machine Learning Approaches to Semantic Web Service Discovery. *Journal of Advances in Information Technology, Vol. 1 No. 3,* August 2010.
8. Dexin Zhao, Zhiyong Feng, Degan Zhang. Research on fuzzy semantic service matchmaking. *2nd International IEEE Biomedical Engineering and informatics Conference,* 2009.
9. Khalid Elgazzar, Ahmed E. Hassan, Patrick Martin. Clustering WSDL Documents to Bootstrap the Discovery of Web Services. *School of Computer, Queen's University, Canada.*
10. Fangfang Liu; Yuliang Shi; Jie Yu; Tianhong Wang; Jingzhe Wu. Measuring Similarity of Web Services Based on WSDL. *IEEE International Conference on Web Services*, 2010.

11. Yiqiao Wang and Eleni Stroulia. Semantic Structure Matching for Assessing Web-Service Similarity. *Computer Science Department, University of Alberta, Edmonton, AB, T6G 2E8, Canada*, 2003.
12. Henar Muñoz Frutos, Ioannis Kotsiopoulos, Luis Miguel Vaquero Gonzalez and Luis Rodero Merino. Enhancing Service Selection by Semantic QoS. The Semantic Web: Research and Applications, 2009.
13. Jyotishman Pathak, Neeraj Koul, Doina Caragea, Vasant G Honavar. A Framework for Semantic Web Services Discovery. *WIDM Proceedings of the 7th annual ACM International Workshop on Web Information and Data Management*, 2005.
14. Web Service Descriptive Language WSDL. http://www.w3.org/TR/wsdl.
15. Aphrodite Tsalgatidou, Thomi Pilioura. An Overview of Standards and Related Technology in Web Services. *University of Athens, Distributed and Parallel Databases, 12, 135–162*, 2002.
16. OWLS-TC V2, OWL-S Service Retrieval Test Collection. http://www.semwebcentral.org/projects/owls-tc/.
17. Richi Nayak. Data Mining in Web Services Discovery and Monitoring. *International Journal of Web Services Research*, 2008.
18. Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, Jun Zhang. Similarity Search for Web Services. *Proceedings of the 30th VLDB Conference, Toronto, Canada*, 2004.
19. Khalid Elgazzar, Ahmed E. Hassan, Patrick Martin. Clustering WSDL Documents to Bootstrap the Discovery of Web Services. *International Conference on Web Services,* 2010.
20. Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer. *Extended Semantic Web conference*, 2009.
21. Semantic annotations for WSDL. http://www.w3.org/2002/ws/sawsdl/.

22. Resource Description Framework RDF. http://www.w3.org/RDF/.
23. The DARPA Agent Markup Language DAML. http://www.daml.org/.
24. Web Ontology Language OWL. http://www.w3.org/2004/OWL/.
25. Marie-Catherine de Marnee and Christopher D. Stanford typed dependencies manual. Manning September 2008 Revised for Stanford Parser v.1.6.5 Nov 2010.
26. Dekang Lin. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. *Department of Computer Science University of Manitoba.*
27. WordNet, a lexical English database.http://wordnet.princeton.edu/.
28. Fausto Giunchiglia, Pavel Shvaiko and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. *In Proceedings of the European Semantic Web Symposium, LNCS 3053, pp. 61-75*, February 2004.
29. Jesús Oliva, José Ignacio Serrano, María Dolores del Castillo, and Ángel Iglesias. SyMSS: A syntax-based measure for short-text semantic similarity. *Data & Knowledge Engineering 70, 390–40,* 2011.
30. Peng Liu, Jingyu Zhang, Xueli Yu. Clustering-Based Semantic Web Service Matchmaking with Automated Knowledge Acquisition. *WISM Proceedings of the International Conference on Web Information*, 2009.
31. Extensible Markup Language. http://www.w3.org/XML/.
32. Simple Object Access Protocol. http://www.w3.org/TR/soap12-part1/.
33. Gruber, T. R. A translation approach to portable ontology specification. Stanford Knowledge Systems Laboratory, International Journal Human-Computer Studies 43, p.907-928, 1993.
34. OWL Web Ontology Language. http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.
35. RDF Working Group. http://www.w3.org/2011/rdf-wg/wiki/Main_Page.

36. Naveen Srinivasan, Massimo Paolucci, Katia Sycara. Semantic Web Service Discovery in the OWL-S IDE.; Carnegie Mellon University.
37. Ontologies and Knowledge Base. http://www.ling.helsinki.fi/~stviitan/documents/Ontologies_and_KB/ontology.html
38. Dekang Lin. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. *Department of computer Science, University of Manitoba*, 1998.
39. JENA API http://jena.sourceforge.net/.
40. Fausto Giunchiglia1, Fiona McNeill2, Mikalai Yatskevich1, Juan Pane1, Paolo Besana2, Pavel Shvaiko3. Approximate structure-preserving semantic matching. *University of Trento, ODBASE*, 2008.