SEMANTIC SERVICE DISCOVERY WITH HEURISTIC RELEVANCE CALCULATION

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

BY

MÜGE ÖZYÖNÜM

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

FEBRUARY 2010

Approval of the thesis:

SEMANTIC SERVICE DISCOVERY WITH HEURISTIC RELEVANCE CALCULATION

submitted by **MÜGE ÖZYÖNÜM** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

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

Prof.Dr. Müslim Bozyigit _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali Hikmet Doğru _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Nihan Kesim Çiçekli _____
Computer Engineering Dept., METU

Assoc.Prof.Dr. Ali Hikmet Doğru _____
Computer Engineering Dept., METU

Asst. Prof. Dr. Pinar Senkul _____
Computer Engineering Dept., METU

Haluk Aydın _____
Biznet Bilişim Sistemleri

Aysun Tuncer _____
Telkotek Bilisim Teknolojileri Sanayii Ticaret A.S.

Date: 03.02.2010

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

Name, Last name : **Müge ÖZYÖNÜM**

Signature :

ABSTRACT

SEMANTIC SERVICE DISCOVERY WITH HEURISTIC RELEVANCE CALCULATION

Müge Özyönüm
M.Sc., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Ali Doğru

February 2010, 103 pages

In this thesis, a semantically aided web service and restful service search mechanism is presented that makes use of an ontology. The mechanism relates method names, input and output parameters for ontology guided matches and offers results with varying relevance corresponding to the matching degree. The mechanism is demonstrated using an experimental domain that is tourism and travel. An ontology is created to support a set of web services that exist in this domain.

Keywords: Semantic Web, UDDI, OWL, Web Services, Restful Services, Web Service Discovery, Semantic Service Discovery, Relevance

ÖZ

YAKINLIK DERECESİ KULLANARAK ANLAMSAL WEB SERVİS BULMA

Müge Özyönüm
Yüksek Lisans, Bilgisayar Mühendisligi Bölümü
Tez Yöneticisi: Doçent. Dr. Ali Doğru

Şubat 2010, 103 sayfa

Bu tezde, web servislerinin ve rest tabanlı servislerin anlamsal bakımdan desteklenmiş bir şekilde bulunmasını sağlayan bir arama mekanizması geliştirilmiştir. Bu mekanizma bir ontololoji kullanmaktadır. Kullanıcıların web servislerinin isimlerine, girdi ve çıktılarına göre ontoloji yardımıyla kolay bir şekilde aradıkları servisleri bulmaları amaçlanmaktadır. Sonuçlar değişen uygunluk derecelerine göre sıralanmaktadır. Bu mekanizmayı göstermek için deneysel olarak turizm ve seyehat alanı kullanılmıştır. Bu alanda mevcut web servislerini desteklemek için bir ontoloji geliştirilmiştir.

Anahtar Kelimeler: Anlamsal Ağ, UDDI ,OWL, Örün Agları, REST tabanlı servisler, Örün Servisi Bulma, Anlamsal Servis Bulma, Yakınlık Derecesi

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my supervisor, Assoc. Prof. Dr. Ali Hikmet DOĞRU for his efforts and guidance throughout this thesis work.

I would like to thank my family for their support and patience.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

In the early days, web was just a collection of static documents. However, with the progress in technology, this limited capability has improved. With the development of applications and services, web started to contain dynamic data [57]. This improvement brought easiness to the businesses as well as individuals.

In today's competitive world, companies aim to produce their services at a faster and cheaper rate without the loss of quality. Businesses understood the importace of the service oriented architecture(SOA) towards achieving their goals. SOA is a computing paradigm that uses software services while developing distributed applications [49]. By using SOA, companies can reuse their resources and this decreases the time and cost of the production phase. This advantage increases the popularity of SOA. Web services are used in order to implement service oriented architecture, web services are used.

"A web service is a self describing, self contained software module available via network, such as Internet, which completes tasks, solves problems, or conducts transactions on the behalf of a user or application. " [49] Companies started to give more importance to web services because of their simplicity and the data interoperability provided by their components which are XML, SOAP, and WSDL [61]. XML [70] stands for Extensible Markup Language and it is formed by a set of rules for encoding documents electronically. The Web Service Description Language (WSDL)[80] is a XML based language for describing web services and how to access them. It contains definitons that are necessary to define and describe a web

service. The Simple Object Access Protocol (SOAP) [49] is a simple XML based messaging protocol for web services to exchange information. SOAP is based on XML and it uses Internet protocols, such as http, to carry its data.

When a user wants to use a web service for his application, first he must find the web service that satisfies his needs. The process of finding a suitable web service for a given task is known as web service discovery [81]. As the popularity of web services increase, the number of web services available also increases and makes it hard to find the web service with the desired capability. This brings a new problem known as web service discovery problem. Finding a web service that can fulfill the request alone is referred to as web service discovery problem [47]. UDDI is one of the web service discovey approach. The Universal Description Discovery and Integration standard (UDDI) [6] defines a way to publish and discover information about web services. UDDI is a XML based registry. Web services are published to world via UDDI and businesses find web services through UDDI. However, UDDI has some drawbacks. First of all, UDDI supports only keyword based search. Keyword based search causes problems like the necessity of human involvment and sensitivity to vocabulary. Also, since XML is used to describe the data model of UDDI, a semantic description of the contents can not be provided [61]. Thus, UDDI is not powerful enough for semantic web service discovery.

In today's web, human interaction is mainly needed to perform a task, since web contents do not have computer-interpretable descriptions. Semantic web focuses on this issue. It can be thought of as a web that is highly intelligent. It is known as intelligent web, because semantic web needs little or no human intervention to carry out tasks such as scheduling appointments and searching for complex documents. In semantic web, web pages are understandable by machines and use ontology for information integration. Ontology is a set of concepts in a certain domain. Ontology represents a domain by describing relationships between those concepts.

Web Ontology Language (OWL) is the technology behind semantic web and it is a semantic markup language for publishing and sharing ontologies. OWL-S is an ontology for describing semantic web services. OWL-S allows for the description of a web service in terms of a Profile, which tells "what the service does", a Process Model, which tells "how the service works", and a Grounding, which tells "how to access the service".

In this thesis, we aim to develop an approach in order to find a solution to the web service discovery problem. For this purpose, we have developed an application that can be alternate to UDDI. Besides the capabilities of UDDI, our aplication is able to make a semantic search. OWL ontology is used while discovering services semantically. Another advantage of our approach is that it is possible to discover both web services and restful services. For discovering services effectively and efficiently, we have developed a searching mechanism which matches available services with the user's needs and then lists them by adding a relevance degree. There are many algorithms that suggest a varying degree of match, for discovery. However, it is very difficult to locate any that has been implemented as an open application. This research attempted to fill such a gap also including restful services that are usually not covered in the existing search mechanisms. We have also make a comparison of existing semantic relevance calculation algorithms with our heuristic algorithm. Our work focuses only to the tourism domain.

The rest of the thesis is organized as follows: Section 2 describes the related work that includes information about web services, restful services, semantic web, RDF and Ontologies. Also, current technologies about web service discovery and composition are discussed in this section. Section 3 describes information about semantic service discovery and OWL. The implementation of our solution is described in Chapter 4. Chapter 5 contains an example use case in order to show the usage of our solution. Explanation of available semantic relevance calculation methods and a comparison with these methods with our method can be found in Chapter

6. In chapter 7, a brief comparison of existing service discovery registries. Finally, conclusions and future work considerations are presented in Chapter 8.

CHAPTER 2

RELATED WORK

In this chapter, some background information on Web services, Web service discovery/composition methods, Restful Web Services, Semantic Web and OWL is provided. The purpose of this chapter is to describe the basic concepts, introduce the necessary terminology, and present relevant definitions.

2.1 Web Services

2.1.1 Introduction to Web Service

According to the World Wide Web Consortium (W3C), a web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format that is basically WSDL. Other systems interact with the Web service using SOAP messages. A “Good” web service is any service that is available over the internet or intranet network, that uses XML messaging format, does not depend on a specific operating system or programming language, and it is self-describing and discoverable. A web service satisfies only a specific function. It can be used in many different applications. Web services should publish themselves to the world so that any application can find and call it. Thus, we can say that web services connect applications with each other. A service that makes credit card verification, currency conversion or stock quotes can be an example for a web service.

2.1.2 Advantages of Web Services

Web services have many advantages. First of all, web services take the advantege of object-oriented programming techniques so that they enable developers to build applications from existing software components. Secondly, web services operate using open, text-based standards. This enables components which are written in different languages and for different platforms to communicate. Since web services allow developers to develop applications by combining code written in different languages or platforms, they also improve collaborative software development. Furthermore, they are easy and inexpensive to implement. They can reduce the costs of enterprise application integration (EAI) and business-to-business (B2B) communications, thus offering companies tangible returns on their investments. Lastly, web services encourage a modular approach to programming, since multiple organizations can communicate with the same web service.

2.1.3 Web Service Architecture

Web service architecture consists of three primary roles. These are the service provider, the service requestor and the service registry. Web service provider is the first important role of the architecture. Web service provider is the company that develops and owns the web service. Provider hosts and controls access to the web service. Also, it is responsible for publishing the web services via internet. Second role in the architecture is web service requestor known also as client. Requestor is the consumer of the web service. This is the enterprise that needs certain functions to be satisfied or the application that calls the web service by sending an XML request. Last role of the web service architecture is web services registry. The registry is a searchable directory that provides a central place where developers can publish new services or find existing ones.

Service provider develops the web services and then publishes these services to the service registry. When a client is in need of a certain task it searches the directory. When the web service with the desired function is found, client uses the information in the service description to bind the web service provider in order to use the service.

Figure 2.1 shows the graphical representation of web service roles and operations.

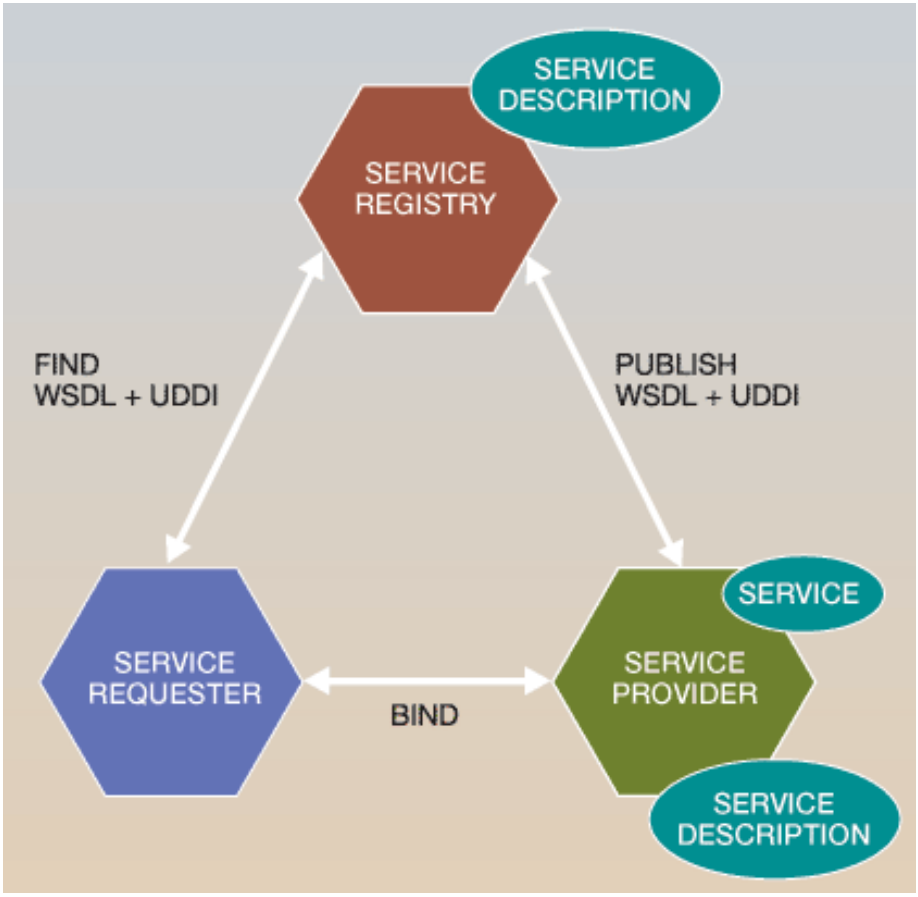


Figure 2. 1 Web Service Architecture

2.1.4 Types of Web Services

Web services can be classified into two types which are simple web services and complex web services. Simple services are Internet based

applications and they do not trust on other web services to satisfy consumer requests. They support only simple request/ response operations. They always wait for a request and when the request comes, they process it and then they respond. A lemon check service can be an example of a simple service. This service provides history information about cars for example it determines whether the car has failed in a previous emission/inspection test or not [14]. On the other hand complex services are the composite of outsourced services. Outsourced services, which can be single or/and complex services, work together in order to offer a value-added service. A car broker service can be an example of a composite service, since this service outsources car dealer, financing and insurance services to provide the complete car sale solution.

2.1.5 Web Service Protocols

Web Services have promised to change the Web from a database of static documents to an e-business marketplace. This technology is being adapted by Business-to-Business applications and by some Business-to-Consumer applications [61]. Since different organizations that have different platforms and languages, object models use web services, it is important to make web services interoperable. Interoperability of web services is provided by standards. WSDL, SOAP and UDDI are the basic standards of web services and they will be explained in the following sections.

2.1.5.1 WSDL (Web Service Descriptive Language)

In order to develop service-based applications, web services should be described in a consistent manner, so that they can be published by providers and found by service clients. WSDL addresses this issue by offering a way for providers and clients to work together easily.

Web Service Descriptive Language (WSDL) is a specification which tells us how to describe a web service. A WSDL document is an XML document that contains all the information that is needed to contact a service. Author of the web service creates the WSDL and publishes it by sending it to a service directory. Web service clients use this registry, and they refer to WSDL representations, to find a Web service that meets their needs.
A WSDL representation can include:

- Operational information like the available operations,
- XML message protocols which are supported by the web service,
- Data type information for the messages,
- Binding information about the transport protocol to be used, and
- Address information for locating the web service.

WSDL description consists of the following elements [26]:

- *Messages* are one-way communications from one computer to another.
- *Types* is a container of datatypes definitions.
- *Port Types* are the set of all operations that a web service can accept.
- *Operations* are analogous to a method call in Java. Several operation types can be determined in a WSDL document which are:

    - **Request/Response**: A client makes a request, and the Web service responds to it.
    - **Solicit/Response**: A Web service sends a message to the client, and the client responds.
    - **One-way**: A client sends a message to the Web service but expects no response.
    - **Notification**: A Web service sends a message to the client but expects no response.

- *Binding* defines the message format and protocol details of a web service.

- *Port* contains the actual IP address and port number of the web service.
- *Service* is the collection of web services.

Figure 2.2 shows an example of WSDL Document taken from http://www.w3.org/TR/wsdl#_introduction

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

To continued..

```
    <operation name="GetLastTradePrice">
           <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort"
binding="tns:StockQuoteBinding">
           <soap:address
location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Figure 2. 2 WSDL Document Example

2.1.5.2 SOAP (Simple Object Access Protocol)

SOAP is a simple XML-based protocol to let applications exchange information over a transport protocol. SOAP is a communication protocol on which applications communicate. It is a format for sending messages. The goal of SOAP is to remove the handicaps of heterogeneity that separates distributed computing platforms. SOAP accomplishes this by its simplicity, flexibility, firewall friendliness, platform neutrality and XML based messaging [49].

SOAP Message exchange model determines how components exchange messages. One-way messages, request/response interactions and peer-to-peer conversations are some examples of message exchange models.

While exchanging SOAP messages, any transport protocol can be used as long as the application that sends and receives the messages understand the protocol. Most common way to transport SOAP messages is HTTP,

since it is ubiquitous. RPC, HTTPS, SMTP can also be used as the transport protocol.

SOAP message includes the following elements:

- Envelope: A SOAP XML document instance is called a SOAP envelope. This is a required element.
- Header: Contains application specific information about the SOAP message. This is an optional element.
- Body: Contains the actual SOAP message. This is a required element.
- Fault: Provides information about errors that occurred while processing the message. This is an optional element.

Web service communication model describes how to invoke web services and depends on SOAP. SOAP communication model is defined by its communication style and its encoding style. SOAP supports two communication styles which are RPC (Remote Procedure Call) and document (message). SOAP encoding style is for describing how the contents of a particular element in the header or body element of the SOAP message are encoded. Encoding style can be “Encoded” or “Literal”. Thus, SOAP communication styles come in four styles which are RPC/Literal, Document/Literal, RPC/Encoded and Document/Literal. Detailed information can be found in http://www.w3.org/TR/2007/REC-soap12-part0-20070427/#intro

Simplicity, portability, firewall friendliness, use of open standards, interoperability, universal acceptance can be listed as the advantages of SOAP. On the other hand, too much reliance on HTTP, statelessness and serialization by value and not by reference can be the disadvantages of SOAP messaging.

Figure 2.3 shows an example of SOAP Message [23]:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<GetWeather xmlns="http://www.webservicex.net">
<CityName>Innsbruck</CityName>
<CountryName>Austria</CountryName>
</GetWeather>
</soap:Body>
</soap:Envelope>
```

Figure 2. 3 An Exapmle Of SOAP Message

2.1.5.3 UDDI (Universal Description, Discovery and Integration)

UDDI is a platform independent, XML based registry for businesses to list their web services on the Internet. The reason that UDDI is acceptable is that it is built on the same SOAP standards the ordinary Web services use.

UDDI enables service providers to describe their services in a global and open environment on the Internet. It also enables service clients to discover information about companies that offer web services, find information about web service interfaces, and information about how to interact with the web service.

Main characteristics of a UDDI can be listed as follows:

- UDDI provides a mechanism to categorize businesses and services using *taxonomies*.
- UDDI *business registration* is an XML document used to describe a business entity and its Web services.
- UDDI is not bound to any technology.

UDDI is composed of three components which are:

- *White pages:* Contain basic contact information about the business that developed the web service. This information can be company name, contact names and phone numbers.
- *Yellow pages:* Web services are organized according to their categories and yellow pages contain these classifications using various taxonomy systems.
- *Green pages:* Contain technical details about the service like how to program a client that can invoke the service or WSDL descriptions.

UDDI also defines a data structure standard for representing company and service description information. The UDDI XML schema defines four main types of information which are:

- *businessEntity*: A description of the organization that provides the service.
- *businessService*: A list of all the Web services offered by the business entity.
- *bindingTemplate*: Describes the technical aspects of the service being offered.
- *tModel ("technical model")*: Used to store technical information on how to use the service, conditions for use, guarantees, etc.

Some drawbacks of UDDI are:

- Need for keywords, service name and manual selection of discovered services,
- Lack of publicly available web services, and
- Lack of correlations between web services and quality of service information.

2.2 Web Service Discovery

Web service discovery is the process of locating, or discovering, one or more related documents that describe a particular Web service using the Web Services Description Language (WSDL). With the discovery process, service clients find that a web service exists and they also find where to locate that web service's description document. Thus, web service discovery is the process of finding a suitable Web service for a given task.

Number of web services increase everyday, since the popularity of web services increase. The increase in the web service count brings a problem which is finding a suitable web service. Finding a web service that can fulfill a specific request alone is known as Web Service Discovery Problem.

2.2.1 Types of Service Discovery

There are two types of service discovery:

- Static service discovery: Static service discovery usually occurs in the design phase. In this case, a human designer examines the list of web services in order to find the service that he needs.
- Dynamic service discovery: In dynamic discovery, service requestor has to specify preferences to enable the application to infer which web service the requestor is most likely to want to invoke. A retrieval operation is performed against the service registry at run time. The decision about which one to choose from the results of retrieval operation is done according to QoS considirations like best price, performance, security and so on and this decision is done by the application.

Heterogeneity is the main obstacle affecting the web service discovery. There are different kinds of heterogeneities [25]:

- Technological heterogeneities include different platforms or different data formats.
- Ontological heterogeneity includes domain-specific terms and concepts within services that can differ from one another, especially when developed by different vendors.
- Pragmatic heterogeneity includes different development of domain-specific processes.

2.2.2 Service Discovery Mechanisms

Different mechanisms exist for web service discovery which are registry approach, index approach, and peer to peer methods [13].

- Registry Approach: In registry approach, web services are collected under an an authoritative, centrally controlled store. In order to make the web service available to the world, the service provider should put the description of the web service into the registry. Owner of the registry decides who can publish its web services and again the owner decides what information should be entered into the registry. UDDI can be given as an example for registry approach.
- Index Approach: Index approach is not authoritative and does not centrally control the information. In this case, web services that are stored in different locations are found and listed. Search engines are examples for this method. Xmethods [83], BindingPoint, Web Service List [67], and StrikeIron [63] are some search engines that are used for web service discovery.
- Peer to Peer Discovery: This method does not rely on centralized registries. In this case, web services discover each other dynamically. P2P method does not have a single point of failure since it is not centralized. Think of each web service as a node. When a requestor is in need for a certain web service, it asks this to its neighboring nodes. If a match is found, then the process ends. However, if no matching

response occurs, each neighbor node, forwards the request to its own neighbors. This continues until a specific termination criterion is reached. CAN [53], Pastry [54] and Chord [62] can be considered as the P2P System.

2.3 Web Service Composition

Today, most companies implement web services that are specific to their businesses. They prefer to outsource other type of web services from other companies since web services are published to the world via Internet. In this scenario, finding the right web service to carry out the needed function becomes an important issue. However, since the number of available services increase from day to day, finding the appropriate service becomes difficult. Also, sometimes, service client can not find one web service to reach its goal. In this case, two or more services can be combined. Reusability of web services provides opportunity for service composition. In particular, the problem of combining multiple web services to satisfy a single task is known as web services composition problem [47].

Let us give an example about the usage of web service composition problem from [47]. Imagine that there exit two web services in UDDI named findRestaurant and findDirection. (1) findRestaurant returns a name, phone number, and address of the closest restaurant provided a zip code and food preference; and (2) findDirection returns driving direction and a map image provided a start and destination addresses are supplied. “Sylvie” visits “State College, PA” on a business trip and stays in the “Atherton” hotel at “100 Atherton Ave, 16801, PA.” She wants to find a Thai restaurant near the hotel along with a driving direction. As you can see, neither of these services can satisfy the user needs. To reach the user’s goal, these two web services shoud be combined. First, findRestaurant(“16801”, “Thai”) service should be invoked to get the address of the closest restaurant, for example, “410 S. Allen St. 16802,

PA"; and then findDirection("100 Atherton Ave, 16801, PA", "410 S. Allen St. 16802, PA") web service should be called in order to get the driving direction.

2.3.1 Web Service Composition Approaches

In order to solve the web service composition problem, several approaches are proposed which can be grouped as manual solutions, semi-automated solutions and automatic solutions.

In manual solutions, each task of the composition process is done manually. The user generates the workflow, finds the services and then again the user sends these services to the execution engine. However, this solution is a complex and challenging one. Difficulties of manual composition can be listed as follows. First challenge is the number of available services. In order to compose services, the developer/provider should first choose which services to use in the combination operation. However, when there is a very large list of web services to choose from, the selection process becomes harder. Secondly, web services are continuously being created and updated. Thus, the developer/ provider should use the most recent version of the service. Lastly, heterogeneity of the services is a very important problem. Web services that will be used in the composition could have been implemented by different organizations. Since there is no unique way for defining and evaluating a web service, different organizations can use different models to describe the services. These drawbacks increase the importance of semi-automated and automated solutions.

Semi- automated solution facilitates the user's job by making suggestions about which web services can be used in the composition. However, the user should still make a selection from a shorter web service list and construct the workflow.

Automatic solution aims to remove the user involvement from the service composition procedure. In this method, agents select, combine, integrate and execute web services. There are two types of automatic web service composition methods which are workflow based and AI planning based.

- *Workflow based service composition:* A workflow specifies the flow of work items. Similarly, a composite service consists of a group of atomic services together with the control and data flow among the services. In this context, a workflow is similar with a composite web service. The current achievements on flexible workflow, automatic process adaption and cross-enterprise integration provide the means for automated Web services composition as well. In addition, the dynamic workflow methods provide the means to bind the abstract nodes with the concrete resources or services automatically [52]. Two kinds of workflow generation techniques exist :

    - Static workflow generation: In this technique, selection and binding of web services are done automatically. However, the process model should be provided by the requestor. With process model, we mean a set of tasks and their data dependencies. Process model can be provided to the tool by different methods. The most common method is providing the model as a graph. However, it can also be provided by using a language that represent the model. Eflow [16] is a platform for the specification, enactment and management of composite services. This platform is an example for static workflow generation based service composition.
    - Dynamic Workflow Generation: In this technique, process model is also created automatically as well as selection and binding of services. In this case, the requestor should specify the constraints of the composition.

- *AI Planning based service composition:* AI Planning based methods are widely used for web service composition problem, since there is a great similarity between these fields. AI Planning can be thought of as searching for a set of actions that can be executed to achieve a goal. Thus, both AI planning and web service composition want to achieve a goal with given initial state. A number of web service composition methods exist that are based on AI planning. Some examples are situation calculus, hierarchical task network planning, PDDL, graph based planning, estimated regression planning, rule based planning, and theorem providing.

2.4 RESTful Web Services

2.4.1 What is REST (REpresentational State Transfer)

REST is an architectural style of networked systems and this style can be used to develop web services. It is based on the way that web already works. The Web is composed of resources and each resource has a unique URL that enables us to access them. For example, METU is a resource that can be reached by the url: http://www.metu.edu.tr/ . Clients can access the METU resource using this URL. After accessing the resource, the representation of the resource is returned. HTML page and XML document can be an example for the representation of the resource. This representation puts the client in a state. Each different representation puts the client in a different state. Then, client can access another resource using the link in the returned representation. In our case, this link can be http://www.metu.edu.tr/about/misguide.php. The state changes according to the returned representation. Thus, the transfer of state occurs with each resource representation that defines the Representational State Transfer.

According to Roy Fielding [49], REpresentational State Transfer adopts the way that how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the client proceeds through an application by choosing links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the client and rendered for their use.

REST is just an architectural style, it is not a standard. It uses standards like HTTP, XML and URLs.

2.4.2 Characteristics of REST

Characteristics of REST style can be listed as:

- Client-Server: Client and server should be separated. This means that clients are not responsible for the things like data storage that servers are responsible and vice versa.
- Stateless: Each request from a client should contain all of the information that is needed to service the request.
- Cache: Responses should label themselves as cacheable or non-cacheable in order to prevent client to reuse inappropriate data.
- Uniform interface: Resources should be accessed by a generic interface like HTTP methods.
- Named resources: The system is composed of resources and each resource should be named by a URL.
- Layered components: Intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to improve system performance, since they enable load balancing and provide shared caches.

2.4.3 When to Use REST

It is important to decide when to use restful services. Restful services are a good choice when:

- The web services are completely stateless.
- The bandwidth needs to be limited. Thus, it can be used with applications that are developed for devices like PDAs and mobile phones.
- Both the producer and consumer of the service understand the content and context being passed along. This is because, restful web service interfaces are not described formally.
- The data, which is returned by the web service, is not generated dynamically so that it can be cached.

2.4.4 RESTful Web Services

Web services can be implemented by using http and the principles of rest. RESTful web services are developed as an alternate to SOAP and WSDL based web services. In REST design, web services are viewed as resources and they are identified by their URLs. HTTP methods are used to implement a web service.

2.4.4.1 HTTP Methods

GET, POST, PUT, DELETE methods are supported by the RESTful services.

GET: HTTP GET method is called in order to retrieve data or perform a query on a resource. The response to this request is a representation of the requested resource.

POST: POST method is for creating a new resource. The response to this request can be a success or failure.

PUT: PUT method is used for changing/ updating the representation of an existing resource.
DELETE: In order to remove/ delete a resource, DELETE method should be used.

2.4.5 RESTful Web Service – An example

Companies like Yahoo, Google, Amazon enable us to use RESTful services. For example, Yahoo has a service that helps us to search news over the Internet. This service is called “Yahoo! News Search”. The following listing shows some of the parameters that should be sent in order to request this service:

- appid - string (required) - The application ID.
- query- string (required) - The query to search for.
- type - all (default), any, or phrase - The kind of search to submit. “All” returns results with all query terms. “Any” returns results with one or more of the query terms. “Phrase” returns results containing the query terms as a phrase.
- results -integer: default 10, max 50 - The number of results to return.

For example, with the following URL, the service can be requested:
http://search.yahooapis.com/NewsSearchService/V1/newsSearch?appid=YahooDemo&query=madonna&results=2&language=en
While requesting the service, HTTP GET method is used in this case.

The Figure 2.4 shows the response from this RESTful service when requesting the service to query for Madonna.

```
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="urn:yahoo:yn"
xsi:schemaLocation="urn:yahoo:yn
http://search.yahooapis.com/NewsSearchService/V1/NewsSearchRes
ponse.xsd" totalResultsAvailable="890"
totalResultsReturned="2" firstResultPosition="1">
<Result>
    <Title>Madonna's Holiday Rip-Off</Title>
    <Summary>Madonna has a bee in her bonnet and the 'B'
stands for Blige. Anyone who thought Mary J. Blige's
Barbershop 2 song 'Not Today' sounds an awful lot like
Madonna's 'Holiday' is not alone.</Summary>

<Url>http://www.muchmusic.com/news/story.asp?id=15432</Url>

<ClickUrl>http://www.muchmusic.com/news/story.asp?id=15432</Cl
ickUrl>
    <NewsSource>MuchMusic.com</NewsSource>

<NewsSourceUrl>http://www.muchmusic.com/news/</NewsSourceUrl>
    <Language>en</Language>
    <PublishDate>1106069393</PublishDate>
    <ModificationDate></ModificationDate>
  </Result>
  <Result>
    <Title>National : BBC Uncovers Madonna's Kabbalah
Center</Title>
    <Summary>The British Broadcasting Corporation has aired a
documentary that has put Madonna and other Hollywood celebs
who embrace a newly packaged version of Kabbalah in some hot
water.</Summary>

<Url>http://www.theconservativevoice.com/modules/news/article.
php?storyid=2079</Url>

<ClickUrl>http://www.theconservativevoice.com/modules/news/art
icle.php?storyid=2079</ClickUrl>
    <NewsSource>The Conservative Voice</NewsSource>

<NewsSourceUrl>http://www.theconservativevoice.com/</NewsSourc
eUrl>
    <Language>en</Language>
    <PublishDate>1106082108</PublishDate>
    <ModificationDate></ModificationDate>
  </Result>
```

Figure 2. 4 Response Of a Restful Service

2.5 Semantic Web

Current web technologies highly depend on human intervention, since the meaning of web content is not machine accessible. Tim Berners Lee, who is the inventor of the WWW, URIs, HTTP, and HTML, realized the deficiency of current web technologies. In order to decrease the role of human in web technologies, he came up with the decision that one needs machine understandable web pages and the use of ontologies for information integration. This resulted in the notion of “semantic web”.

Semantic web can be thought of as a web that is highly intelligent. It is known as intelligent web, because semantic web needs little or no human intervention to carry out tasks such as scheduling appointments and searching for complex documents. In semantic web, pages are understandable by machines that use ontology for information integration.

The Semantic Web is able to describe things in a way that computers can understand. The Semantic Web is not about links between web pages. The Semantic Web describes the relationships between things (like A is a part of B and Y is a member of Z) and the properties of things (like size, weight, age, and price).

2.5.1 Layers of Semantic Web

Semantic web is formed by layers as can be seen in the Figure 2.5. Next, we will examine what each layer is responsible for.

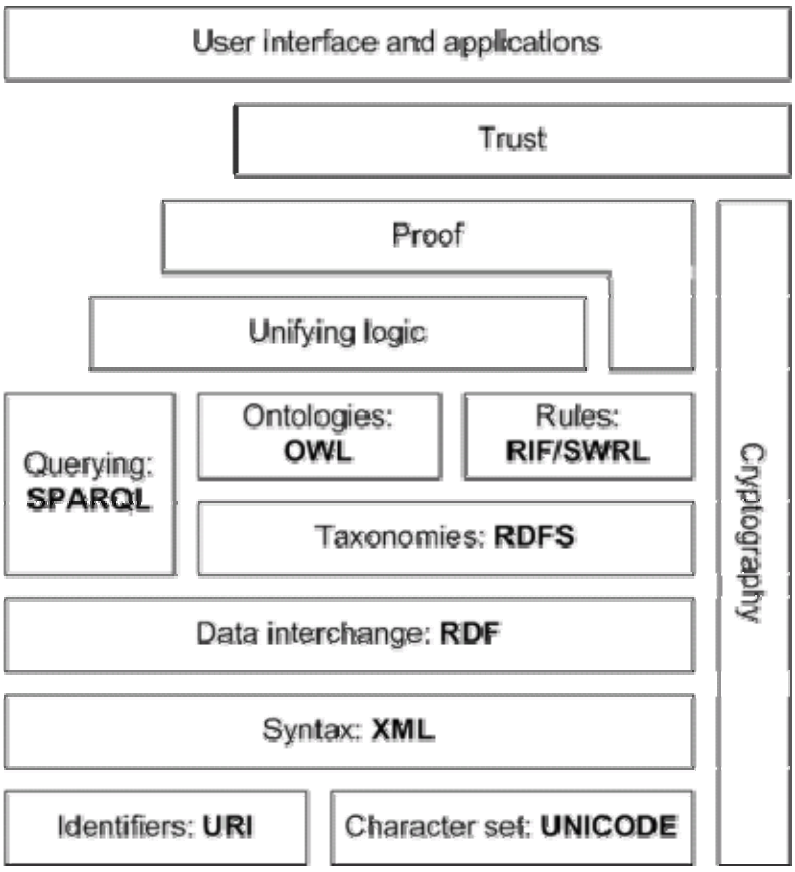


Figure 2. 5 Layers of Semantic Web

- URI and Unicode Layer: Unicode is a standard of encoding international character sets. Uniform Resource Identifier (URI) is a string of a standardized form that allows to uniquely identify a resource. URL is a subset of URI. The usage of URI is important for a distributed internet system as it provides understandable identification of all resources.
- XML Layer: It is the standard representation language for document exchange. If a document is not marked-up, then each machine may display the document in its own way. This makes document exchange difficult. XML is a markup language that has certain rules. XML Schema is a language for providing and restricting the structure and content of elements contained within XML documents. Thus, if all

documents are marked-up using XML, then there is uniform representation of documents.

- RDF Layer: This is the layer which is above XML layer. RDF is a new concept which is introduced with the semantic web concept. RDF stands for Resource Description Framework. A document could have different meanings at different sites. If we want to exchange information in a meaningful way on the web, we should use RDF. RDF is a framework for representing information about resources in a graph form. It is based on triples subject-predicate-object that form graph of data. RDF is a language for expressing data models, which refer to objects (resources) and their relationships. An RDF-based model can be represented in XML syntax.
- RDFS Layer: RDFS Schema is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes. RDFS can be used to describe taxonomies of classes and properties and use them to create lightweight ontologies.
- Ontology Layer: Ontology layer is necessary for information integration. Different communities develop their own ontology and they have to publish them on web. Using this ontology, different groups can communicate information. More detailed ontologies can be created with Web Ontology Language (OWL).
- RIF/ SWRL Layer: RDFS and OWL have semantics defined and this semantics can be used for reasoning within ontologies and knowledge bases described using these languages. To provide rules beyond the constructs available from these languages, rule languages such as RIF and SWRL are being standardized for the semantic web.
- SPARQL: SPARQL stands for Simple Protocol and RDF Query Language. It is a protocol and query language for semantic web data sources. SPARQL is SQL-like language, but uses RDF triples and resources for both matching part of the query and for returning results of the query. Since both RDFS and OWL are built on RDF, SPARQL

can be used for querying ontologies and knowledge bases directly as well.

- Proof Layer: It is expected that all the semantics and rules will be executed at the layers below Proof and the result will be used to prove deductions.
- Trust Layer: At the top level, trust layer lies. This layer mainly asks the questions “How do you trust information on the web?” and “How do you trust the resources?” It depends on where the information comes from, which means that trust depends on the author of the information. Interested parties have to communicate with each other and determine how to trust each other and how to trust the information obtained on the web.
- For reliable inputs, cryptography should be used, such as digital signatures for verification of the origin of the sources.
- On top of these layers, application with user interface can be built.

2.5.2 Semantic Web Services

Just like normal web services, semantic web services comprise the server end of a client-server system for machine-to-machine interaction via the world wide web. Semantic services are a component of semantic web because they use markup which makes data machine-readable in a detailed and sophisticated way.

Semantic web services address the following problems:

- Defining web services capabilities semantically to facilitate
    - Autonomous computing
    - Service discovery
    - Service invocation
    - Service composition
- Mediation between information and data resources

- Discovery of information resources
- QoS of web services and resources
- Semantically enriched user policies and security
    - Flexible authorization schemes

2.6 Resource Description Framework (RDF)

RDF is a framework for representing information about resources in a graph form. RDF is built around resources with URI, because it was primarily intended for representing metadata (data about data) about WWW resources. RDF is an infrastructure which enables the encoding, exchange and reuse of structured metadata. RDF uses XML as a common syntax for the exchange and processing of metadata.

2.6.1 RDF Data Model

Main purpose of RDF is to describe resources on the web by providing a model. RDF data model contains three concepts which are resource, properties and statements.

Resources are used to describe individual objects of any kind. An entire web page, a part of a web page or a collection of web pages can be given as an example of resources. According to RDF, a resource is any object that can be identified by a URI.

A property is used to define a certain characteristic, attribute or relation that is used for describing a resource. A Property is a resource that has a name, such as "author" or "homepage". A Property value is the value of a Property, such as "Jan Egil Refsnes" or http://www.w3.com.

Statements are composed of a specific resource, its named property and the value property of that property for that resource. These three parts of a

statement can be called as the subject, the predicate and the object. Together, they can be called as triple. Thus, sstatements are <subject, predicate, object> triples which can be shown by a graph. An example of a triple can be seen in Figure 2.6

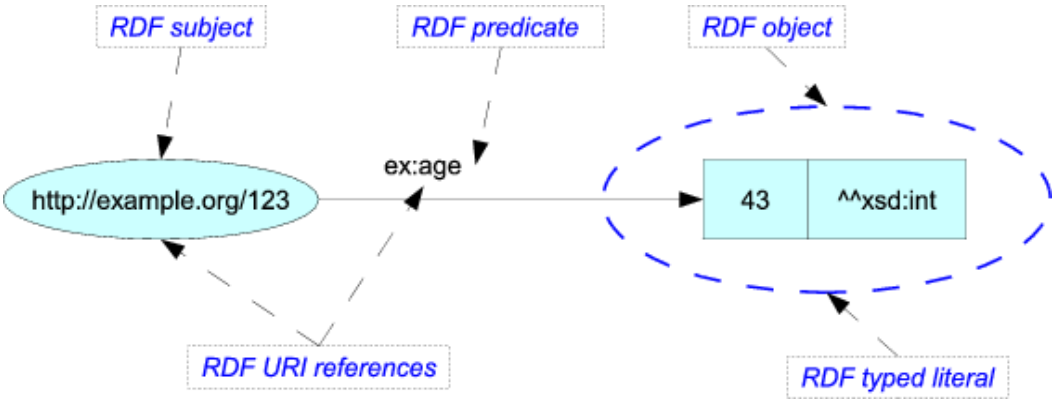


Figure 2. 6 RDF Triple/ RDF Statement

2.6.2 RDF Syntax

RDF models can also be represented by using XML, since RDF has an XML syntax that has a specific meaning.

- Every Description element that is shown by `<rdf:Description>` tag describes a resource
- Every attribute or nested element inside a Description is a property of that Resource.
- We can refer to resources by URIs

More details about RDF Syntax can be found in [7] .

```
<rdf:Description rdf:about="some.uri/person/sean_bechhofer">
  <o:hasColleague resource="some.uri/person/ian_horrocks"/>
  <o:hasName rdf:datatype="&xsd;string">Sean K.
Bechhofer</o:hasName>
</rdf:Description>
<rdf:Description rdf:about="some.uri/person/ian_horrocks">
  <o:hasHomePage>http://www.cs.mam.ac.uk/~horrocks</o:hasHomePage>
</rdf:Description>
<rdf:Description rdf:about="some.uri/person/carole_goble">
  <o:hasColleague resource="some.uri/person/ian_horrocks"/>
</rdf:Description>
```

Figure 2. 7 RDF/XML Example

2.6.3 RDF Schema

RDF Schema describes how to use RDF to describe RDF vocabularies. This specification defines a vocabulary for this purpose and defines other built-in RDF vocabulary initially specified in the RDF Model and Syntax Specification.

Main RDFS Constructs can be listed as follows:

rdfs:Class allows to declare a resource as a class for other resources.

rdfs:subClassOf allows to declare hierarchies of classes.

rdf:Property is the class of RDF properties. Each member of the class is an RDF predicate.

rdfs:domain of an rdf:predicate declares the class of the subject in a triple whose second component is the predicate.

rdfs:range of an rdf:predicate declares the class or datatype of the object in a triple whose second component is the predicate.

rdfs:subPropertyOf is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.

rdfs:seeAlso is an instance of rdf:Property that is used to indicate a resource that might provide additional information about the subject resource.

rdfs:isDefinedBy is an instance of rdf:Property that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described.

More details about RDF Schema go to [15] .

2.7 Ontology

To help capture web services' semantic features, ontology concept is used. An ontology is a formal representation of a set of concepts within a domain and the relations between those concepts. Main purpose of ontologies are defining the domain. For this purpose it contains vocabulary, which involves objects their properties and relations. Ideally, ontology should capture a shared understanding of a domain of interest and provide a formal and machine manipulability model of the domain.

Usage areas of ontologies are e-Science for example, the gene ontology, the protein ontology (MGED), "in silico" investigations relating theory and data; medicine, databases for Integration and Query answering; user interfaces; linguistics and semantic web.

2.7.1 OWL

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies. OWL is a major technology for semantic web. For this reason, it has attracted both academic and commercial interest. OWL is a part of the semantic web where web information has exact meaning, web information can be processed by computers and computers can integrate information from the web. One of the advantages OWL ontologies is the availability of tools that can reason about them.

There are three kinds of OWL with different levels of expressiveness which are OWL-Lite, OWL-DL and OWL-Full. OWL Lite supports those users

primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. OWL DL supports those users who want the maximum expressiveness while retaining computational completeness. OWL-DL has a well defined semantics that tells us how to interpret expressions in the language. OWL DL is based on a well understood Description Logic (comprises concepts, roles, individual names, subsumption, and operators). OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. All the constraints of OWL-DL plus some more are included in OWL-Lite, and some constraints allowed in OWL-Full are permitted in OWL-DL. Because of these, we can reason about OWL ontologies, allowing us to draw inferences from the basic facts that we provide.

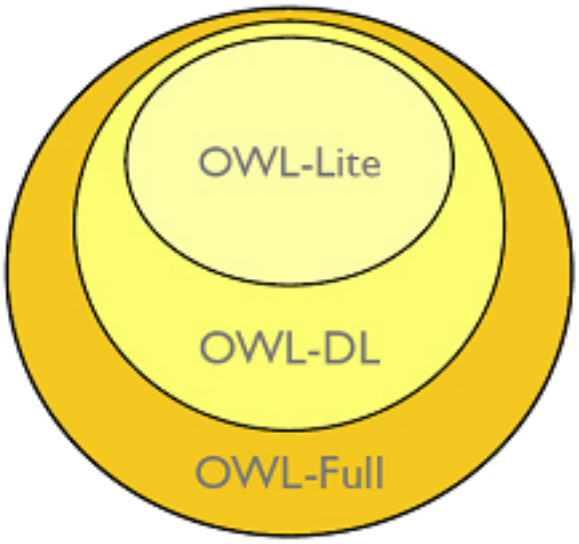


Figure 2. 8 OWL Types

2.7.2 OWL-S

OWL-S is an ontology built on top of Web Ontology Language (OWL) by the DARPA DAML Program. It is an ontology for describing semantic web services. OWL-S enable users and/or software agents to discover, invoke, compose and monitor services on the Web easily, with a high level of automation. Main motivation of OWL-S is to enable:

- Automatic Web Service Discovery: Finding the service that will satisfy the constrains automatically.
- Automatic Web Service Invocation: Execution of the web service by an agent.
- Automatic Web Service Composition & Interoperation: Automatic selection, composition and interoperation of web services.
- Automatic Web Service Execution Monitoring: The ability to find out where the request is in the process, the status of the request, and exceptions that occur.

OWL-S has three major parts which are:

- Service Profile: Describes what the servie does.
- Service Model: Describes how the service works.
- Service Grounding: Describes how to access the service

These parts will be explained in detail in the following sections.

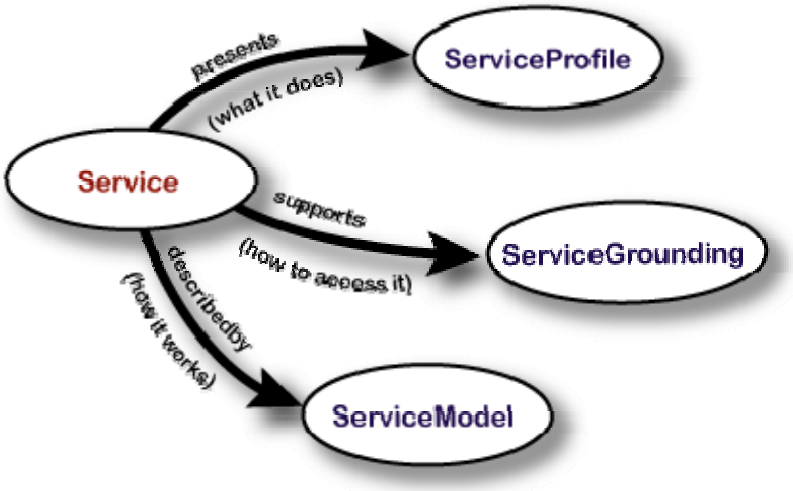


Figure 2. 9 OWL-S Model

2.7.2.1 Service Profile

Service Profile describes the capabilities and information of the web service. The purpose of service profile is advertising and discovering. It is mostly used by the clients in order to determine whether the service meets its needs or not. The most important information it contains are the inputs, outputs, preconditions and postconditions of the service.

It is formed by three types of information which are:

- A human readable description of the service,
- The functional behavior of the service that is represented as a transformation from the service inputs to the service outputs, and
- Several nonfunctional attributes that specify additional information about a service like the cost of the service.

2.7.2.2 Service Model

Service model specifies how a client can interact with a web service. Service model can be viewed as a process. However, in this case a

process is not an executable program. It contains descriptions about the ways of interacting with the service.

There can be an atomic process and a composite process. An atomic process is a service that takes a set of inputs and returns a set of output in a single step. On the other hand, a composite process is a set of services in which each message, which the client sends, is advanced through the process.

Process model has two aims. First of all, when an information and the world state is given to it, we mean inputs, it generates and return a new information which is the output. A process model can also produce a change in the world, which means a service can also have an effect on its domain after the execution. This change is described by preconditions and effects of the process. Inputs, outputs, preconditions and effects of the services are included in the process model.

2.7.2.3 Service Grounding

Service grounding gives details about how to access the service. This may include information about protocol and message formats, serialization, transport, and addressing. A grounding can be thought of as a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. In OWL-S, the ServiceProfile and the ServiceModel are the abstract representations. However, the ServiceGrounding deals with the concrete level of specification.

CHAPTER 3

SERVICE DISCOVERY WITH OWL

In this chapter, we explain how the application works and how it uses OWL while discovering services.

3.1 System Architecture

In our system, we aim to design a web based application in order to help companies or programmers to find the services they need easily and efficiently. We provide a system that can be an alternative to UDDI. Besides the capabilities of UDDI, our application is also able to make a semantic search. The results of the searches also contain a field that shows the relevance between the requested and the found service. This facilities the user's selection process by helping him/her to see how close is the proposed service to the needed service. Another advantage of our system is that clients can both search for web services and restful services. The architecture of our system can be seen in Figure 3.1.

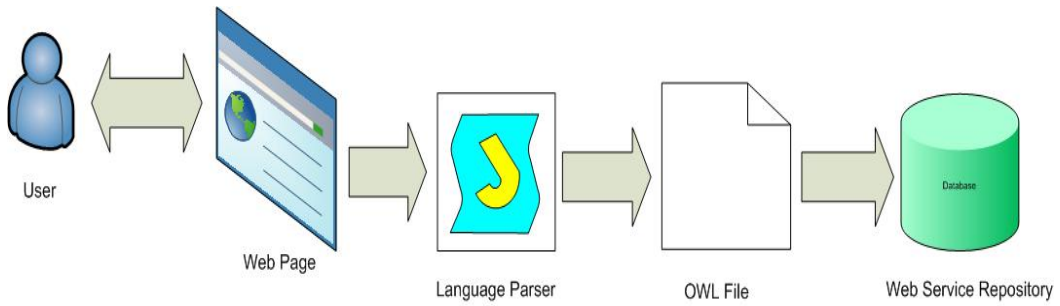


Figure 3. 1 System Architecture

As can be seen in the figure, the user goes to the web page that is implemented for searching web services. Then all he/she needs to do is entering the search criteria. The user can search services according to their functionality or/and expected inputs and outputs of the service with a combination of the functionality of the service needed. Also, at this step, user should specify whether he/she is looking for web service or restful service or both.

Then, the page sends the users input to the language parser. The job of the parser is to match the users inputs with the vocabulary that is defined in the owl file. Language Parser part of the system is out of the scope of this thesis. Thus, we implemented only a small parser. Our parser makes only synonym mapping for the words that are in the ontology. If a synonym can not be found then we look for the word that contains the maximum number of matching characters.

The parser returns a list of vocabulary that will be used to search web services. These vocabularies are sent to the OWL file. In this file, the vocabularies that are related with the user's inputs are found.

Finally, the exact words and the related words are all used in the query which is executed in order to select the list of web services that fulfill the users request. The returned list contains the relevance of the services as mentioned before.

The implementation of each of these parts are explained in Chapter 4. In this chapter, it is shown how OWL with an efficient algorithm can be used to find web services.

3.2 Semantic Web Service Discovery

Service discovery aims to find the best service that fits the requester's needs. Discovery is done based on the functionality, input and output of

the service. In these cases, in order to find the best web service, the functionality, input and/or output of the service request should match with the functionality input and/or output of the available service. However, this can be tricky in most conditions, since there can be ambiguity in the syntax and semantics of the searching criteria.

Let's explain the ambiguity in service discovery with an example [24]. Suppose there exists a web service for tax calculation, which has a single operation that is receiving an input message with floating point amount and string zip, and producing an output message with floating point rate and integer tax. (See Figure 3.2.)

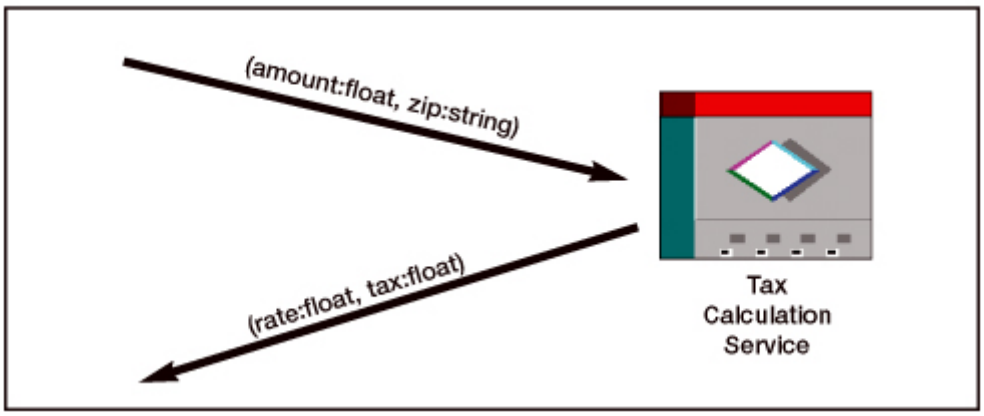


Figure 3. 2 Tax Calculation Service

Even with this simple service, there can be two types of confusions. First arises when there is a shared syntax but different semantics. By this, we mean that there can be confusions when differences in semantics are not specified. Thus, the client aims to use one service, but he/she ends up with another service which has a different semantic. Since the syntax is same, the service does not generate an error, but it produces an unexpected result. For our tax service, there can be two tax services with same syntax of inputs and outputs. However, one can calculate an income tax while the other one calculates sales tax.

The second type of confusion occurs when there is a shared semantics but different syntax. Two web service can have the same semantics with different syntax. Suppose there are two tax services which calculates sales tax. However, the first service rate is a multiplier, which means a 5.5% rate would be given as 0.55, while the other service rate is a percentage with the same value given as 5.5%. Since multiplier and percentage are semantically same, these two services share a common semantic with different syntax.

To overcome these problems in service discovery, there has been a considirable amount of work about the discovery of web services going on. One of the most common topic is discovering web services by UDDI. However, there are some drawbacks of this mechanism. First of all, UDDI is a centralized registry, so it suffers from single point of failure and bottlenecks. Also, UDDI has a limited search mechanism. It enables web services to discover their functionality using a classification schema and this may cause errors. Besides these drawbacks, UDDI does not allow us to add semantics to the web service discovery process, since it uses XML as its data model. Without semantics UDDI can not solve the ambiguity problem while matching the searching criteria with the published web service.

In order to solve these problems, semantic approach is needed. Thus, semantic web should be used in order to find a solution to the service discovery problem. As discussed in section 2, semantic web is based on ontology which enables us to define words and their relations.

In this thesis, we propose a method to find services by using OWL. The client can make a search according to the functionality, input and/or output of the service. By using OWL, we can remove the misunderstanding of the semantics of the search criteria. Thus, we can provide a list of services in

which exists a match between the advertised properties and searched criteria.

3.2 OWL

As described earlier in Chapter 2, OWL is a language for describing ontologies in semantic web and it is based on RDF. Ontologies play an important role in semantic web, since they describe the terms and relationships between those terms and thus enable us to interpret the meanings of those terms.

3.2.1 Advantages Of Using OWL

RDF, which is described detail in Chapter 2, is a limited language. Thus, OWL is developed in order to overcome the drawbacks of RDF. OWL takes the basic abilities of RDF and extends those abilities in order to produce a more useful language. We can describe the capabilities and thus the advantages of using OWL as [27]:

- By using OWL, classes can be declared and these classes can be organized in a subclass hierarchy.
- Classes in OWL can be defined as the intersections, unions or complements of other classes.
- Bu using OWL, properties can be desclared and these properties can be organized in subproperty hierarchy. Also, domains and ranges of these properties can be defined. Domains of OWL properties are OWL classes and ranges of properties can be OWL classes or datatypes like string, integer.
- OWL allow us to define a property as a transitive, symmetric, functional or inverse of another property.
- OWL enable us to tell which individual or object is belong to which class and the properties of these individuals.

- By OWL, “equivalence” relation can be made for classes and properties; “disjointness” relation can be made for classes; “equality” and “inequality” relation can be made between individuals.
- OWL let us to describe restrictions on how properties behave. For example, with OWL we can say that at least one value for the property should come from a certain class.

OWL is chosen as the ontology language in our application, because of the advantages that are listed above.

In order to discover services semantically, the relation between the user’s request data and the advertised data should be understood. OWL has the flexibility that enables us to describe this relationship. It is possible to describe relations between data that is not possible with RDF. Also, OWL-S and DAML-S is not preferred, because they add semantic to web services. However, with our application, you can also make a search for restful services. Thus, we needed an ontology that can be used for both web services and restful services. OWL manages that.

3.2.2 OWL Basics

OWL is a language that is based on RDF which is described detailly in Section 2. Thus, it relies on triplet statements like RDF. Basic elements of OWL ontology are classes, properties, instances of classes, and relationships between these instances. In this section, we will explain these elements and some of the language components that are used to define these elements.

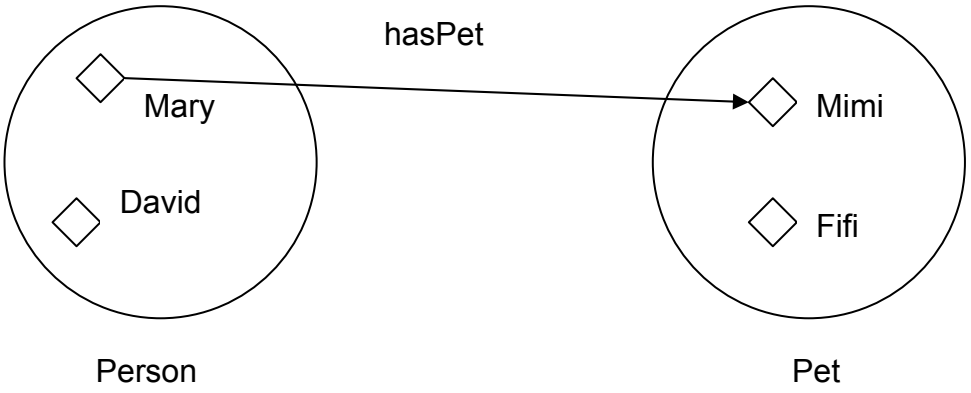


Figure 3. 3 Respresentation of Classes, Individuals and Properties

3.2.2.1 Classes

OWL classes can be thought as sets that contain individuals. An example representation of classes can be seen in Figure 3.3. “Person” and “Pet” are the classes.

Classes can be classified as subclasses and superclasses. These hierarchy is known as taxonomy and these are represented by `owl:SubClassOf` and `owl:SuperClassOf` respectively. Subclasses are the speciliazed version of their superclasses. Forexample, “Animal” class is the super class of “Pet” class and “Pet” class is the subclass of “Animal” class. `owl:Thing` is the most general class in OWL. Thus, every individual in the OWL world is a subclass of `owl:Thing`.

3.2.2.2 Individuals

Individuals are also known as instances, since they are the instances of classes. They represent the objects in the domain. Individuals are the members of the classes. In Figure 3.3, “Mary” and “David” are the individuals of “Person” class while “Mimi” and “Fifi” are the individuals of “Pet” class.

3.2.2.3 Properties

Properties represent the relation between two individuals. In Figure 3.3, “hasPet” shows the relation between individual “Mary” and individual “Mimi”.

OWL contains two types of properties which are Data Properties and Object Properties. Data Properties determine the data types and expressed as `xs:datatypes`, for example `xs:int`, `xs:string`, `xs:date`, etc. Object Properties represent the relation between individuals of two classes. “hasPet” relation is an example of an object property.

Properties can be restricted in several ways. They can be restricted by determining “Domain” and “Range”. Domain and Range shows the classes that the property can be assigned to. For example, in Figure 3.3, “Person” class is the domain and “Pet” class is the range of the “hasPet” property. Also, a property can be the a speciliazed version of the existing property which is called subproperty. Forexample, “hasPet” can be subproperty of “hasAnimal” property.

Properties can have charateristics. These characteristics are symmetric property, transitive property, functional property, inverseOf property and inverseFunctional property. Now, let us explain what are these characteristics are.

- Symmetric Property: If a property, say P, is a symmetric property, then P relates indivial A to individual B, and individual B is also related to individual A with property P.
- Transitive Property: If a property, say P, is a transitive property, then we can say that property P relates individual A to individual C knowing that individual A relates to individual B with property P and individual B relates to individual C with property P.

- Functional Property: If a property, say P, is a functional property, then only one individual can be related with the property. For example, “hasBirthMother” should be a functional property, since a person can only have one birth mother.
- InverseOf Property: If a property, say P1, is the inverse property of property P2, then if individual A relates to individual B with property P1, individual B relates to individual A with property P2. For example, “isBirthMotherOf” property is the inverse property of “hasBirthMother” property.
- InverseFunctional Property: If a property, say P, is a inverse functional property, then the inverse property is functional. Inverse property of “hasBirthMother” property is “isBirthMotherOf” property and since “hasBirthMother” property is functional, “isBirthMotherOf” property is functional, too.

More information about OWL can be found at http://www.w3.org/TR/2004/REC-owl-guide-20040210/

CHAPTER 4

A SEMANTIC SEARCH MECHANISM FOR WEB SERVICES

Our application uses Java language as a web based J2EE application. The web pages are developed by using HTML and JSP. Java is used as the programming language for the application. Eclipse framework is used as the platform. As the application server, GlassFish is selected. Microsoft SQL Server 2005 is used as a database for storing the web services. PROTEGE [51] tool is used for developing the ontology. Also, JENA [30] library is used in order to call ontology within the Java code.

Detailed explanation of each component can be found in following sections.

4.1 Web Interface

Our application is a web based application, thus it can be accessed by browsers like Internet Explorer, Firefox and Google Chrome. HTML and JSP technologies are used for developing the interfaces.

Figure 4.1 shows the screen for searching a web service. In this page, the user can search services according to their functionality, the inputs that should be given to a service and the outputs produced by the service. In this screen, the user specifies the type of the service that will be searched. It can be normal web service which is based on SOAP, WSDL or it can be a restful web service. The user can also conduct a search for both.

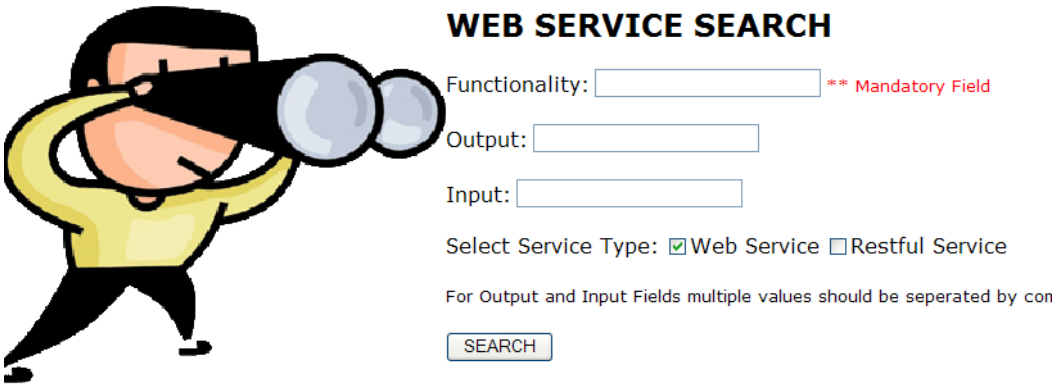


Figure 4. 1 Web Service Search Page

Figure 4.2 shows the result page of the application. When the user makes a search, the web services that matches with the search criteria are listed in this page. Appropriateness of the listed web services can be seen by looking at the relevance field of the result table.

WEB SERVICE SEARCH RESULT

Relevance (%)	Service Name	Method Name	Access Information	
80	Global Weather	GetCitiesByCountry	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=48	http://www.v
80	US Address verification	VerifyAddress	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=24	http://www.v wsdl
100	USA Zip code Information	GetInfoByZIP	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=35	http://www.v
100	USA Zip code Information	GetInfoByState	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=35	http://www.v
100	USA Zip code Information	GetInfoByAreaCode	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=35	http://www.v
90	UK Location	GetUKLocationByCounty	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=28	http://www.v
90	UK Location	GetUKLocationByPostCode	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=28	http://www.v
90	Australian PostCode	GetAustralianLocationByPostCode	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=29	http://www.v wsdl
70	UK Location	ValidateUKAddress	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=28	http://www.v

New Search

Figure 4. 2 Web Service Result Page

4.2 Web Service Database

In our application, we store some information related with the available web services in a database. These web services are developed by different companies and they are published via internet. We aim to collect the tourism domain related ones in a certain store in order to ease the accessibility and searchability of the services. For this purpose, Microsoft SQL Server 2005 is used as a database.

In the database, several properties of the web services are kept. These properties are ServiceName, MethodName, AccessInfo, Inputs, Outputs, Type and WSDLLocation. “ServiceName” property is for the name of the service. “MethodName” property keeps the method name of the service, since a service can have several methods. “Inputs” property holds the necessary input parameters for the service to work properly and lastly “Outputs” property holds the values that the service returns after execution. “Type” field is to distinguish RESTful services and web services. “WSDLLocation” holds the location of the WSDL document for web services. This field does not hold anything for RESTful services, since these services do not have a WSDL file. Figure 4.3 shows the database that our application uses.

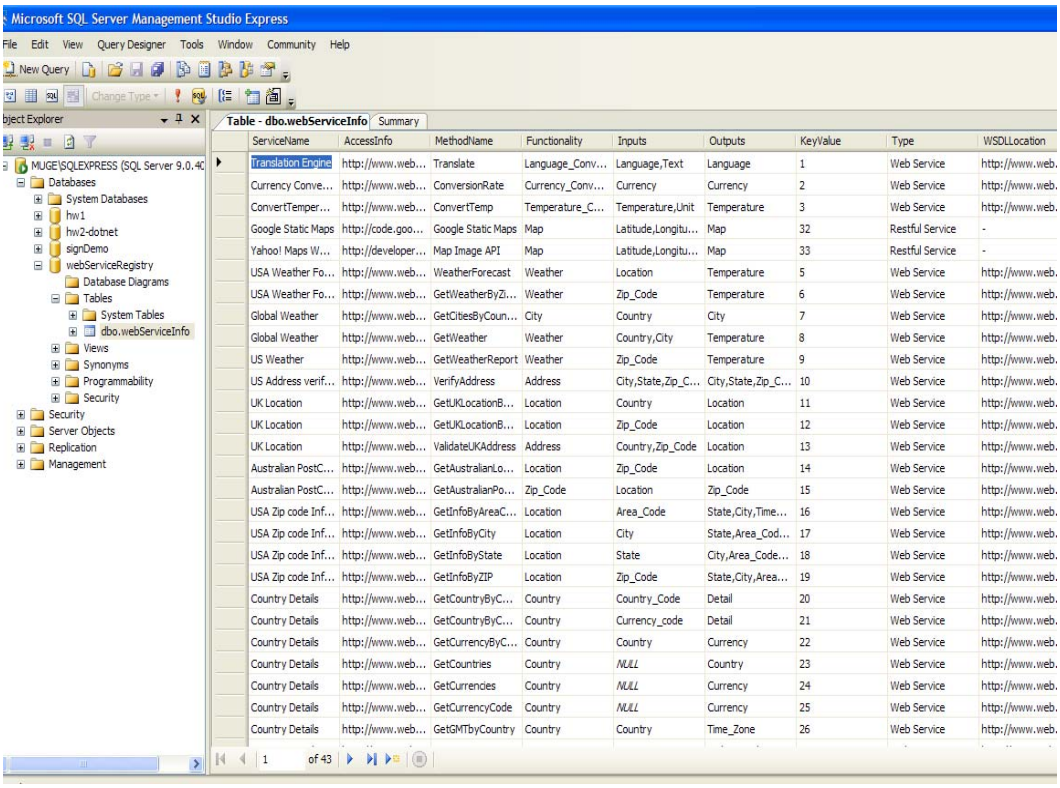


Figure 4. 3 Web Service Database

4.3 Ontology

Ontology has a major part in our application, since it provides the semantic. In order to obtain the relations between concepts, web ontology language (OWL) is used. Following sections show how the OWL is formed and how it is used with the application.

4.3.1 OWL Formation

Protege is an open source ontology editor that enables us to form ontologies easily. It provides an environment in which we can:

- Form and save OWL and RDF ontologies,
- Edit and visualize classes, properties,
- Execute reasoners and
- Edit OWL individuals.

While developing our ontology, we focused on the web services that are available in our database. The classes and relations between those classes are formed according to the inputs, outputs and functionalities of the existing web services.

The following figure shows the graphical representation part of our ontology:

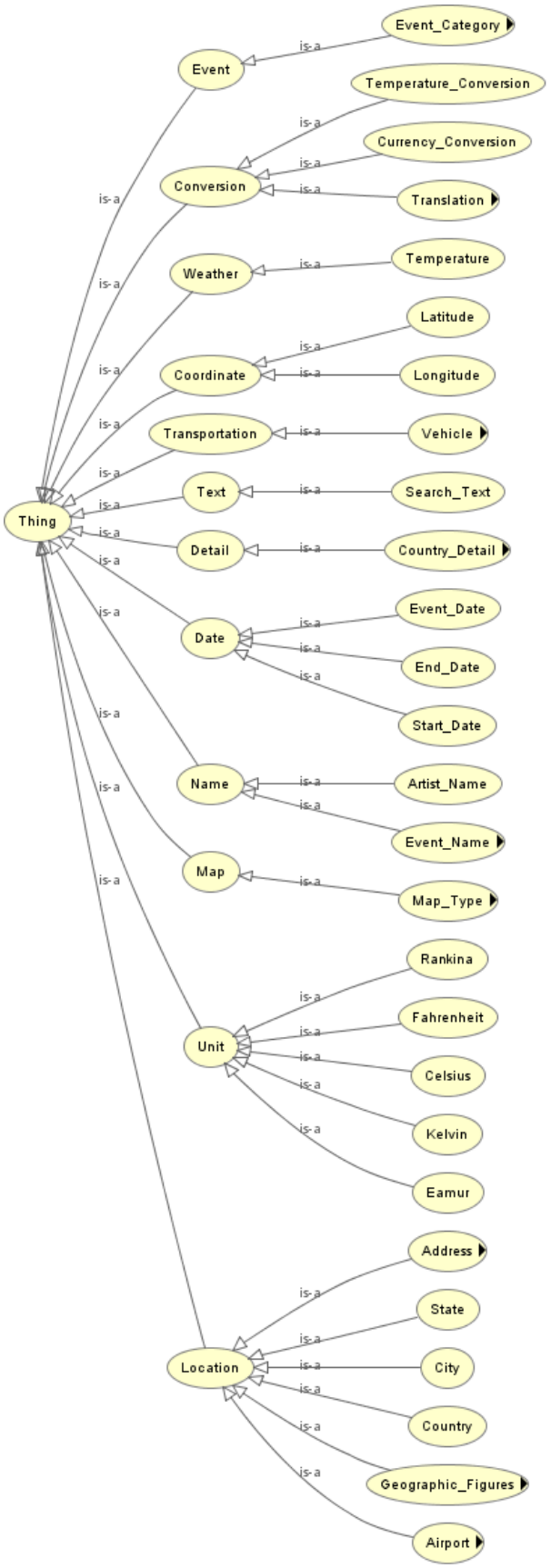


Figure 4. 4 System Ontology

4.3.2 OWL Parsing

Jena framework is used to call the OWL file that is formed by using the protege tool from our Java code. Jena is an open-source framework and it makes it easy to form semantic web applications. Besides containing RDF API and SPARQL query engine, it also contains an OWL API which is what we need.

When the user enters his/her search criteria and submits it, the code should parse the ontology in order to find the related classes with the user's search criteria. This is done easily by the help of the Jena Framework.

4.4 Searching Algorithm

In this section, the algorithm which is used for matching the request with the available web services are explained. The algorithm aims to find the most appropriate web services that satisfy the client's needs. The algorithm contains two important parts. First, it selects the web services according to the request, then it estimates a closeness degree by looking at the certain criteria and this closeness is shown in the relevance field of the result list. For this purpose, the relevance field of the matched web services that is returned by the application is very important for the requester. The matching of web services and calculation of relevance of these services is done by using a specific algorithm. Now we will investigate these algorithms closely.

4.4.1. Matching Part of the Algorithm

In our application, the searching can be done based on three fields which are functionality, input and output. Functionality is a mandatory field, for this reason, the client should enter the capability of the service that he/she is looking for. However, it is optional to enter input and output. According to

the client's search criteria, our algorithm is divided in four parts which will be investigated next.

The first case is when the client makes a search by only specifying the functionality of the service that is needed. In this case, if the requested functionality is same with the functionality of the available service, then we can say that an exact match is found. There can not be always an exact match. In this case, we look at the ontology and found the classes that are superclass and subclass of the requested functionality. Superclasses subsume the requested functionality which means it is the generalization of the requested functionality. Thus, this subsume relationship is also an exact match, since the returned web service contains the functionality of the requested service. If the returned functionality is the subclass of the requested functionlity, then this relation is a second degree relation.

To clarify this explanation, we can give an example. Suppose that we have a vehicle ontology [41] as shown in Figure 4.5. Suppose we have a web service whose functionality is mapped to "Car". If the client makes a request with "Car" functionality, then an exact match is returned. Instead of car, if the client searches with "SUV", then again an exact match is returned, since "Car" subsumes "SUV". However, if "Vehicle" is requested, then a second degree match is established, because the returned service does not contain all the capabilities that the client asks for. The client wants both the "Bus" and "Car" capabilities, but only "Car" is returned.

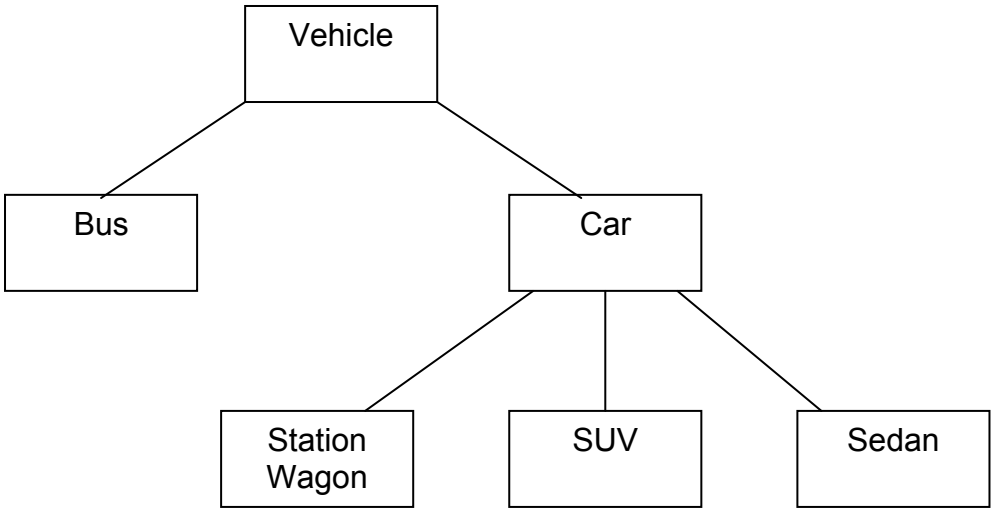


Figure 4. 5 Vehicle Ontology Example

The second case is when the client makes a search by specifying the functionality of the service that is needed and the inputs that the service is waiting for execute properly. First of all, the web services are selected according to the functionality property. The search, which is based on inputs, is done between these services. The user can enter one or more input variable(s). The input variables are iterated in order to find the exact match. If there exists a web service with all the variables requested, then an exact match is returned. If some part of the requested inputs match with the services inputs, then a second degree match is found. Then, the algorithm looks for the subclasses and superclasses of the inputs. Superclasses are again considered as the exact match and subclasses are considered as second degree match.

The third case is when the client makes a search by specifying the functionality of the service that is needed and the outputs that the service produced after proper execution. The algorithm logic is same with the previous case. The only thing that differs is that the client enters output parameters instead of input parameters.

The last case is when the client makes a search by specifying the functionality of the service that is needed as well as the outputs that the service produced after proper execution and the inputs that the service is waiting for execute properly. Again, the web services are selected according to the functionality property. The result list, which is based on inputs and outputs, is formed between these services. The web services in this result list is searched for matching outputs. Then web services based on input parameters are selected. If services exist that fits both input, output criteria, they form an exact match. For each unexisted output or input, we again list the service, but by reflecting this to the relevance of the service.

If there is no service that matches with any of the request, its subclasses or superclasses, then we can say that the search is a failure, since an empty list is returned.

4.4.2 Relevance Calculation Part of the Algorithm

The second part of the algorithm calculates a relevance degree for the matched web services. The purpose of this calculation is to help user to see how close is the found service to the requested one. The relevance is shown with numbers between 1 and 100. A value of 100 indicates the maximum relevance which is an exact match and 1 shows the minimum relevance.

If only functionality parameter is submitted by the user, relevance is calculated by looking only at this parameter. However, if input or output paramaters of the requested services are also submitted along with the functionality, first relevance is calculated according to the functionality and the result forms the base for the input or output calculation. The calculation is done for each input or output, if the user submits more than one input or output. When the user makes a search with all three criterias, then first functionlity, then output and lastly the input is considered.

Relevance calculation takes the following parameters into consideration:

- Generalization
- Specialization
- Language Match
- Number of Matched Inputs/ Outputs

Figure 4.6 shows the overall relevance algorithm that contains generalization, specification and language match. All the integer variables that should be omitted are decided by us.

```
int relevance = 100;
for each input data
{
     if (word is a generalization)
           then relevance = relevance -10
      if (word is a specialization)
           then relevance = relevance -20
      if (word is a substring match)
      {
           then relevance = relevance -5
           if (matching characters >5 )
                 then relevance = relevance -5
           if (matching characters ==5 )
                 then relevance = relevance -10
           if (matching characters ==4 )
                 then relevance = relevance -15
           if (matching characters ==3 )
                 then relevance = relevance -20
      }

      for each unmacthed output of the requested service
      {
           if(# of outputs user entered > # of outputs of the
service)
           relevance = relevance-5
      }
      for each unmacthed input of the requested service
      {
           if(# of inputs user entered != # of inputs of the
service)
           relevance = relevance-5
      }

}
```

Figure 4. 6 Relevance Algorithm

Now, we will explain each of the parameters that are considered during calculation.

4.4.2.1 Generalization

The user's input, which can be functionality, input data of the web service and/or output of the web service, is searched in the ontology in order to find a web service match. If a direct match can not be found, then we look for the matches for the generalization of the user's input. However, since a direct match can not be found, we omit 10 points from the overall relevance degree. For example, by looking again to the figure 4.5, if user searches for "car" input and a web service, which is tagged as "car", can not be found, then we search for the web services tagged as "vehicle". The web services that match to the "vehicle" criteria will have 10 points less than the overall relevance, since the user looks for car and we found vehicle.

4.4.2.2 Specialization

The user's input, which can be functionality, input data of the web service and/or output of the web service, is searched in the ontology in order to find a web service match. If a direct match can not be found, then we look for the matches for the specialization of the user's input. However, since a direct match can not be found, we omit points, 20 points, from the overall relevance degree. Web services that are found with the specification property has more point losses than the ones found with generalization property. This is because generalization can also contain the user match, but specification is the more specific version of the requested service. For example, by looking again to the figure 4.5, if user searches for "car" input and a web service, which is tagged as "car", can not be found, then we search for the web services tagged as "SUV". The web services that match

to the “SUV” criteria will have 20 points less than the overall relevance, since the user looks for car and we found SUV.

4.4.2.3 Language Match

When the user submits the input data, it is sent to the language parser in order to find a match in the ontology. First, language parser tries to find the synonym of the input data. If a match is found, there is no need to omit points, since the user’s search criteria exists in the ontology. However, if a match can not be found, then we look for the substring match of the input data. In this case, we omit points and we take the matching word that has the longest prefix.

We remove 5 points for not finding a match for the user’s criteria. Then we continue removing points according to the number of characters that match with the words in the ontology. If more than 5 characters match, then we omit 5 points, but if only 3 characters match then we omit 20 points.

For example, suppose that the user entered “context” as the search criteria. We look for word “context” in our ontology by language parser. However, there exists no match. In this case, we will omit 5 points for not finding a match. Then, we will continue looking according to the substring match. This searching will result with the word “convert”. These two words have 3 characters common. Thus, we will remove 20 points according to the algorithm. Overall, we will remove 25 points.

4.4.2.4 Number of Matched Inputs/ Outputs

User enters expected inputs and outputs of the web service as a search criteria in our application. There can be more than one input or output for the needed web service. In that case, the inputs and outputs should be deliminated by a comma to be recognized as different inputs/outputs.

Our search mechanism first looks for the services that contain all the expected data. If any service is found, there is not any point loss. However, all the services can not fit this criteria. Some web services accept only some of the inputs/outputs that the user has entered as matching criteria. While calculating the relevance, we also consider this situation. Our algorithm breaks 5 point for each input / output that the user wants but the service does not contain.

To clarify this, let's give an example. Suppose the user searches for a service that takes "car" and "bus" as an input. Our application starts making a search according to these inputs. If a service that takes both "car" and "bus" as input is found, then it is returned with relevance 100. However, if a service which accepts only the "car" input is found, then it is returned by relevance 95.

However, for outputs field, the user can expect only a single output where the returned service produces two or more outputs including the requested one. In that case, there is no need to break any points, since the user's need is met. This situation is not correct for inputs field. When the user wants to submit two or more inputs where only one input is needed for the service to work correctly or when the service is expecting two or more inputs while the user only has one, we break points, because the user's need is not met.

CHAPTER 5

A CASE STUDY: DEVELOPING A TOURISM WEB PAGE

In this chapter, we aim to show how to use our web service search application. This will be done by showing some examples and explaining the key features of our system. Since our application is designed for tourism domain, we will suppose that developers who are implementing a web page for a tourism company will use our application. We will not develop a tourism web page. Instead, we will show how to use our application while developing one.

5.1 Description of the System Needs

A tourism web site is a site that travelers visit in order to determine where to go, how to go and get detail information about the place that they will visit. Thus, web page should be capable of answering these needs of the travelers.

While developing these tourism sites, the developers will not prefer to implement each of the services and functionalities from the rough, since this will be a time and money consuming process. In order to make an efficient development, the developers will be willing to use existing web services and restful services. However, first of all, they should determine which services they can use. In this phase our application steps in.

For this study we will suppose the developers will need the following services:

- Weather forecast service
- Conversion service

- Event list service
- Map service

Next, we will see how to make a search to find the services that are needed.

5.2 Service Search Using the Application

In this section, we will show example usage of the search application for finding the services listed in the previous section. First, we will show a basic example and then we will show a more advanced one.

5.2.1 Weather Forecast Services

When a tourist wants to visit a certain city, he/she will most probably want to see the weather forecast of that city during his/her trip. For this reason, it is a good choice to add this capability to the tourism application. However, developing this utility is a waste of money and time since there are already available services for this. The problem is deciding which one of these available services should be used. This determination can be done according to the expected inputs and outputs of the service.

In this case, suppose the developer needs a web service. A RESTful service is not useful since for example, he/she wants the output data of the service to be generated dynamically. Also, the expected input of the service is the name of the city and the dates that the weather will be shown. Figure 5.1 shows how to make a request for this case.

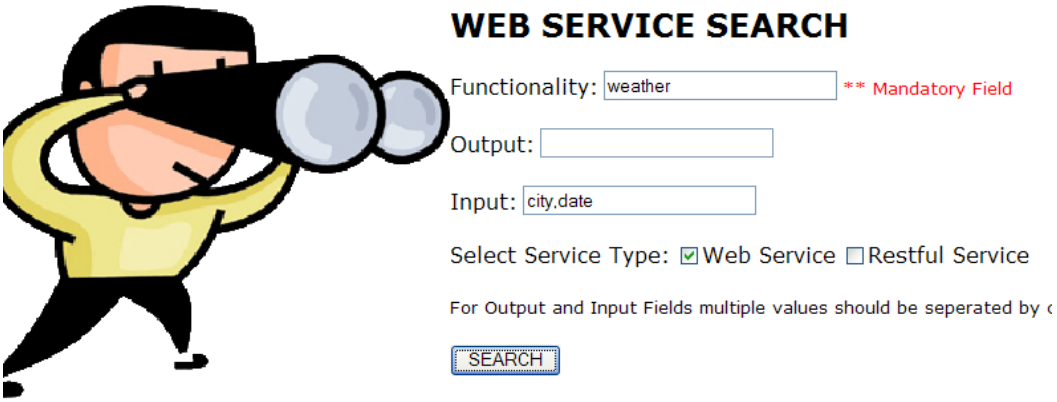


Figure 5. 1 Making a Request for Weather Service

The result of this search can be seen in Figure 5.2. In our ontology, “Temperature” is the specialization of the “Weather”. Thus, we look our database for services that have “Temperature” or “Weather” functionality and we found five services that match with the “Weather” functionality. One of them is a restful service, since the request asked for only web services, thus we have four services to look for input match. Two of these four services takes “Zip_Code” as input. In our ontology, “Zip_Code” is not related with city, since “Zip_Code” is neither generilization nor specificaiton of “City”. That’s why only two services are listed in the result list.

The first web service ensures all the requested criterias expect it takes only “City” as input but not the “Date”. That’s why it’s relevance is equal to 95. The other service takes “Location” which is a generalizaiton of “City” as input. This service does not also take “Date” as input. Thus, it’s relevance is 85, 10 points loss from generalization and 5 points loss from unmatched data.

WEB SERVICE SEARCH RESULT

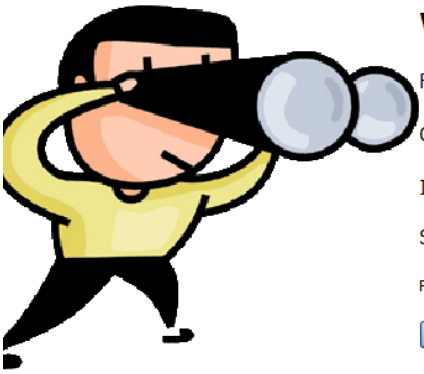
Relevance (%)	Service Name	Method Name	Access Information	WSDL Lo
95	Global Weather	GetWeather	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=48	http://www.webservicex.net wsdl
85	USA Weather Forecast	WeatherForecast	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=44	http://www.webservicex.net wsdl

New Search

Figure 5. 2 Result of Weather Service Search

5.2.2 Conversion Service

If you are going on a vacation, especially if you are going abroad, you want to learn detailed information about the country/city that you will visit. For example, you may need the currency conversion rates for that country, so that you can be prepared or you may want to learn some specific words in the language of the city that will be visited so that you won't be a complete stranger. Thus, developers may also need a service that makes translation. For this purpose, suppose the search in Figure 5.3 is done.



WEB SERVICE SEARCH

Functionality: conversation ** Mandatory Field

Output:

Input:

Select Service Type: ☑ Web Service ☐ Restful Service

For Output and Input Fields multiple values should be seperated by comma(,)

SEARCH

Figure 5. 3 Request for a Translation Service

As can be seen in the figure, developer searches for services with functionality "Conversation". This is not a word that exists in the ontology. Thus, we look for the closest substring for that input which is "Conversion". These words have 7 characters in common. According to our algorithm, if

the matched characters are bigger than 5, we only break 5 points, since the words are getting similar when the number of matched characters increase. Also, a web service whose functionality is “Conversion” does not exist in our database. Thus, we look for the specialization of this and we break 20 points for each service whose functionality is a specialization of “Conversion”. As can be seen in Figure 5.4, three services satisfy these criteria with relevance 70.

WEB SERVICE SEARCH RESULT

Relevance (%)	Service Name	Method Name	Access Information	WSDL I
70	Currency Converter	ConversionRate	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=18	http://www.webservicex.ne wsdl
70	ConvertTemperature	ConvertTemp	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=61	http://www.webservicex.ne wsdl
70	Translation Engine	Translate	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=47	http://www.webservicex.ne wsdl

New Search

Figure 5. 4 Response of the Translation Service Request

If the requester makes a more detailed search by entering the input or/and output criteria, then a more direct result may have been returned. For example, if the requester also submits an input as “language”, then the result will be only show the “Translation Engine” service, which is what the user is looking for. The requester will not be able to find this service by specifying “Conversation” input with a site that does not make ontology search.

WEB SERVICE SEARCH RESULT

Relevance (%)	Service Name	Method Name	Access Information	WSDL Location
70	Translation Engine	Translate	http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=47	http://www.webservicex.net/Transl wsdl

New Search

Figure 5. 5 Response of a Detailed Translation Service Request

5.2.3 Event and Map Services

Sometimes, for tourists, the purpose of a city visit is to join a special event that takes place in the visited city. For this reason, when deciding the place to see, they want to learn which activities take place in that city and they want to see how to reach the place that the activities take place. Thus, developers may also want to add this utility to their applications.

In order to find a web service that returns a map that shows the location of the event, the request shown in Figure 5.6 can be done. In this case, let's suppose the developers need restful services because of a limited bandwidth. However, no services are returned. Thus, the developer can decide to compose two web services in order to maintain this function. In composition of the web services, inputs and outputs have an important role, since the output of one of the service is the input for the other service.

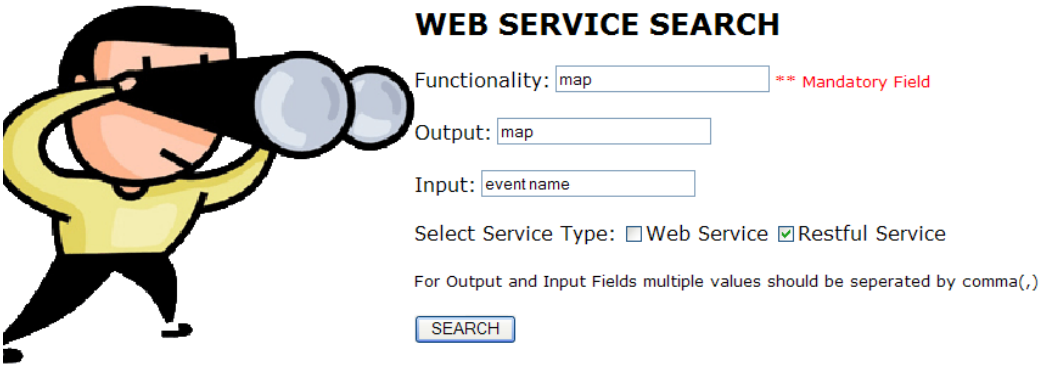


Figure 5. 6 Request for Map/ Event Service

Now, developers need two services. One will take the name of the event as input and will return the location of that event as output. The second service will take the location as the input and produce a map as an output. Figure 5.7 and Figure 5.8 shows these searches respectively.

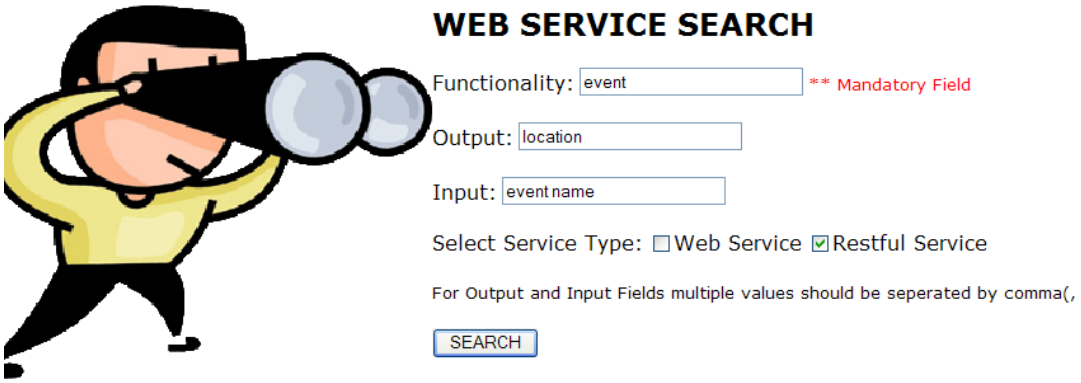


Figure 5. 7 Event Location Service Search

The request, which is made in order to find the location of the event, returns two services. One of these services has 100 relevance, which tells us that it does exactly what the user is need. It takes the event name as input and produces location as the output.

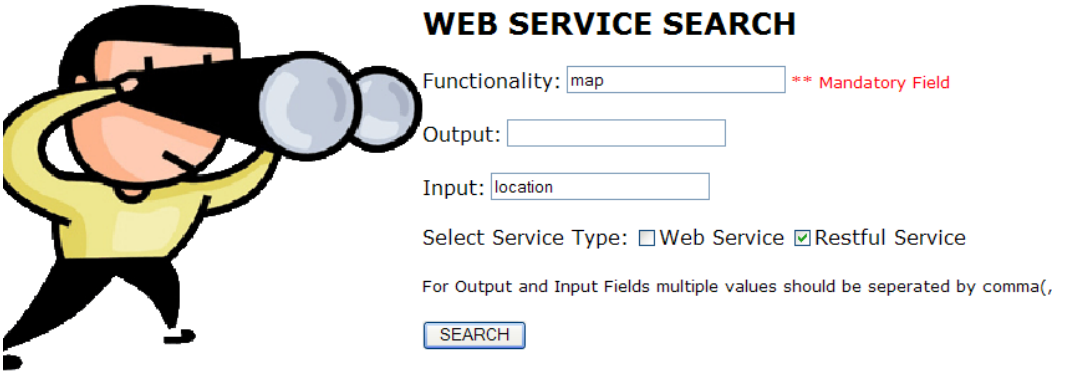


Figure 5. 8 Map Service Search

Now, the developer needs a service that takes the location as the input, since the output of the event location service will be used as the input of the map service. The request which is shown in Figure 5.8 returns the result that is shown in Figure 5.10.

WEB SERVICE SEARCH RESULT

Relevance (%)	Service Name	Method Name	Access Information	WSDL Location
)0	5GIGs Event	artist.getEvents	http://www.5gig.com/api/	-
100	Yahoo! Upcoming Events	event.getInfo	http://upcoming.yahoo.com/services/api/event.getInfo.php	-

New Search

Figure 5. 9 Response for Event Location Service Search

The closest match of the map service is the one with relevance 90. This service takes latitude and longitude of the location as well as the location as input. Thus, the develop now should also find a service which finds the coordinates of a location. For each unmatched input 5 points are cut off. The second service takes the longitude, latitude, address and map type as the input. Again three unmatched inputs exist that cause the developer to search for another service. 15 points are broken from unmatched data. Address data is a specialization of the location data, thus 20 points are taken out. The resulting t relevance is 65.

WEB SERVICE SEARCH RESULT

Relevance (%)	Service Name	Method Name	Access Information	WSD Locati
90	Yahoo! Maps Web Services	Map Image API	http://developer.yahoo.com/maps/rest/V1/	-
65	Google Static Maps	Google Static Maps	http://code.google.com/apis/maps/documentation/staticmaps/	-

New Search

Figure 5. 10 Response for Map Service Request

CHAPTER 6

SEMANTIC RELEVANCE CALCULATION

This chapter explains existing methods for semantic similarity and semantic relevance calculation. Also, we will compare the results some of these methods with the results returned from algorithm. This comparison will be done according to the searches that can be done in our application.

6.1 Available Relevance Calculation Algorithms

Semantic relevance calculation has a major part in research area, since it is widely used in informaiton retrieval. For this reason, there have been many papers published about the algorithms that can be used to calculate semantic similarity. Next, we will investigate some of the methods that are used during semantic service discovery.

In [1], a novel approach is presented in order to discover web services. This approach combines semantic and statistical association metrics. Semantic metrics are based on semantic aspects of ontology. Statistical association metrics are based on the association aspects of web services instances, specifically their inputs and outputs. In order to establish a semantic relevance, semantic relationship ranking and a hyperclique pattern discovery approach which groups web service parameters into meaningful associations are combined.

In [28], Hongen Lu is presented an algorithm that can be used to rank web services.
In this paper, first a way to match the user's request with the advertised service is presented. Then, in order to reduce the burden of finding the most

relevant web service, a ranking algorithm that considers the semantics of each service is proposed.

Natenapa Sriharee proposes an ontology based rating model for service quality for efficient semantic service discovery in [44]. According to this paper, services are rated for quality by a third-party organisation which gives scores in terms of ontological values. The services will be matched by using a matching algorithm and they will be ranked based on service consumers' preference criteria.

Another ranking mechanism that can be used in semantic web service discovery is proposed in [19]. In this paper, while determining the degree of match between the request and the advertisement, the use of recall and precision is proposed as suitable measures.

In [31], a semantic web service ranking approach is described which is based on the semantic descriptions of non-functional properties of services and user preferences.

A new approach for semantic web service ranking is introduced in [48]. This approach calculates ranking based on Vector Space Model. In this model, the user query and each of its related results will be modeled as a vector.

Besides these approaches that are specific for web services ranking, there also exists methods for ranking semantic similarity that can be used in finding similarity of textes or information retrieval etc. These methods can also be used while ranking web service during discovery. Next, we will explain some of these methods.

The method for semantic relevance measure that is proposed in [55], aims to calculate the relevance between resources as numeric values based on the semantic relations between concepts. This paper extends the similarity to relevance so that even the relations between completely different

resources can also be considered, and explore the possibility of measuring such semantic relevance without being limited to a specific structure.

In [82], a novel semantic similarity algorithm based on OWL ontologies is introduced. This algorithm first parses OWL ontologies, and then translates OWL ontologies into RDF triples, and finally, utilizes an improved dynamic adjustment of the semantic weight of OWL constructors to calculate the semantic similarity.

The work in [29], presents an approach for measuring semantic similarity between words and concepts. It combines a lexical taxonomy structure with corpus statistical information so that the semantic distance between nodes in the semantic space constructed by the taxonomy can be better quantified with the computational evidence derived from a distributional analysis of corpus data.

Jaccard index [71], Dice's coefficient [69], Cosine similarity [68] are also methods that are used in order to calculate the semantic similarity of contents.

6.2 Evaluation of Our Semantic Relevance Calculation Algorithm

Some of the techniques explained in the previous part will be compared with our relevance calculation approach in this part. This will be done based on some experimental searches.

Dice’s Coefficient [69] is a similarity measure. For sets X and Y of keywords used in information retrieval, the coefficient may be defined as:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

Suppose we make a service search by specifying “convert” as our functionality parameter. Our relevance calculation algorithm returns “80%” as the relevance degree, because the closest word in the ontology is “conversion”. However, if the same search is made by using Dice’s coefficient, the result will be close to “0.70”. The set of bigrams in each word (convert, conversion) will be found first. These are {co, on, nv, ve, er, rt} and {co, on, nv, ve, er, rs, si, io, on} respectively. First set has six elements, where the second set has nine elements and the intersection of these two sets has only five elements which are { co, on, nv, ve, er }. Inserting these numbers into the formula, we calculate, $s = (2 \cdot 5) / (6 + 9) = 0.70$. Our algorithm and Dice Coefficient calculated close numbers, but according to our algorithm these words are more related.

Requested Functionality	**Advertised Functionality**	**Dice Coefficicent**	**Our Relevance Algorithm**
convert	conversion	~70	80
conversation	conversion	50	70

Figure 6. 1 Dice’s Coeffiecent vs Our Algorithm

Another evaluation of our algorithm can be done by using the approach in [19]. This approach is sampled by a simple scenario where the user is interested in watching a movie and is searching for a Web service in which, some details about the movie are given as the

input and available cinemas and showtimes are returned as the output. To semantically describe the request and the service capabilities, a sample ontology is considered, as shown in Figure 6.2. [19]

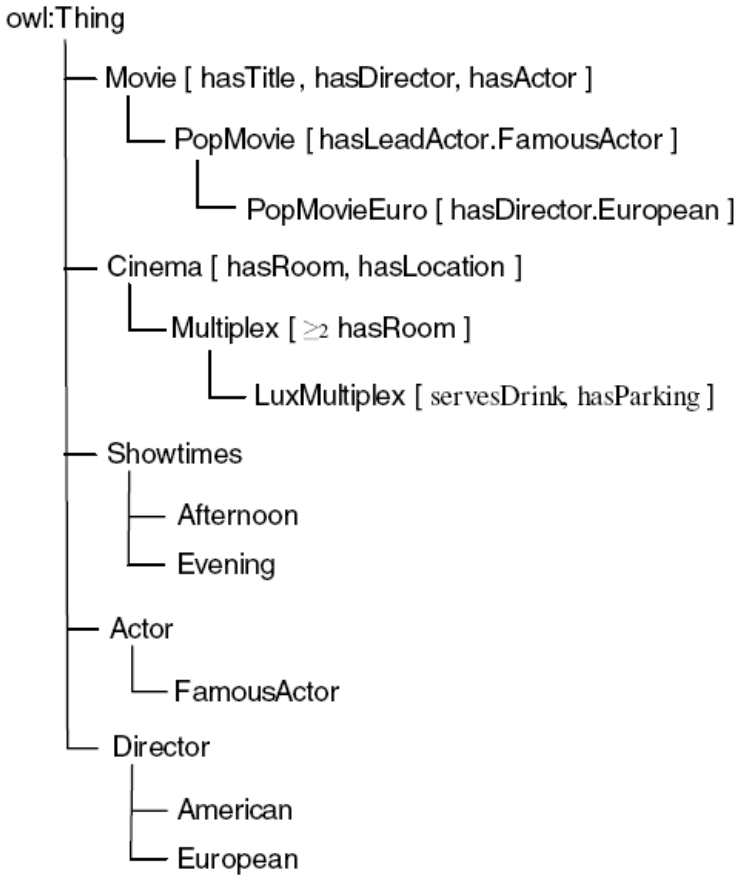


Figure 6. 2 Sample Movie Ontology

The requested search criterias, advertised web services, the relevance degrees calculated by the approach in [19] and by our algorithm can be seen in Figure 6.3. The relevance degrees returned from the relevance calculation approach in [19] are bigger then the degrees from our algorithm. This is because they make a more detailed and sensitive calculation according to ours. However, the results from tese two methods can be considered as close.

	Input	Output 1	Output 2	Relevance approach in [19]	Relevance our approach
Request	PopMovie	Multiplex	Showtimes		
Advertisement1	PopMovieEuro	LuxMultiplex	Evening	0.79	60%
Advertisement 2	Movie	Cinema	Showtimes	0.96	80%
Advertisement 3	Movie	Multiplex	Showtimes	0.95	90%
Advertisement 4	Movie	Multiplex	Evening	0.87	70%

Figure 6. 3 Our Relevance Approach vs Approach in [19]

CHAPTER 7

COMPARISON OF WEB SERVICE SEARCH MECHANISMS

In this chapter, we will compare some existing web service search mechanisms with the search mechanism that we developed. The comparison will be built on certain parameters.

7.1 Comparison Parameters

We will examine existing search mechanisms according to certain criteria. These criteria can be explained as follows:

- Accessibility: By accessibility, we mean how easy or hard it is to reach the registry. Is it a free registry that everyone can access easily or is it an expensive registry that you should be a member and pay some amount to use the registry. The values that accessibility parameter can take are “Free”, “Membership”, “Expensive”.
- Efficiency: Efficiency parameter shows how easy to locate web services correctly. This parameter can take “Low”, “Medium” and “High” values.
- Web Registry Type: This parameter shows the architecture of the registry. Registries can be a central or distributed. A central registry means that there is only one location that controls all the tasks. These registries have a single point of failure. On the other hand, a distributed registry consisting of many computers that can be located in different areas. If one of them fails, otherscontinue to serve. Thus, we can set “Central” and “Distributed” values to this parameter.
- Web Service Structure: Web services that are used in the registry can be developed by the developers of the registry. These services are “Local” services. If the service registry is formed by using the existing

services that are developed by different companies, then we label it as "Distributed". Also, some registries use both the services that are developed by themselves and other companies. These registries will be labeled as "Both".

- Service Type: This parameter is about the types of services that we can search through the registries, in other words service types that the registries support. For this thesis, we are interested in web services and RESTful services. Thus, the values of this field can be "Only Web Services", "Only RESTful Services" and "Both".
- Ease Of Use: This parameter shows how friendly is the searching mechanism to the user. Is it a user friendly mechanism or not? For example, how many clicks are needed to form the search query is a criterion to decide the ease of use parameter. This parameter can take "Easy", "Medium" and "Hard" values.
- Use of Semantics: This parameter shows whether the matching algorithm of the service discovery process includes semantic matching or not. The values can be "Yes" or "No". We can also specify how the semantic is achieved by using this parameter. Semantic capability can be added to the algorithm by using "OWL", "OWL-S", "DAML" or"DAML-S".

7.2 Existing Search Mechanisms

We will compare the following web service search mechanisms with our semantic based web service search application:

- UDDI: As described in detail in chapter 2, UDDI is a registry in which web services are listed.
- WebserviceX.NET: This is an internet site that provides web services for certain fields like financial, distribution, retail, health care, manufacturing, telecom, government, and educational industry [66].

- Yahoo! API: This is an internet site that contains restful services for developers to use while developing an application. Detailed information about this site can be reached from [84].
- UDDI with DAML-S: This is an approach for semantic UDDI registry and it is based on [40] and [41]. This registry uses DAML-S in order to add semantics to the UDDI. The Service Profile part of the DAML-S is mapped with the Business Service part that contains information about the company which develops the service and Business Entity part that contains Tmodels, which enables them to map capabilities of the service in DAML-S profile with UDDI, of the UDDI. The detailed explanation can be found in [40]. Then, the services are discovered by using the matching algorithm as described in [41]. The matching algorithm conducts a four degree match. These degrees are “exact”, “plugin”, “subsume” and “fail”.
- UDDI with OWL-S: This is also another approach for semantic UDDI registry. This approach is very similar to the one called “UDDI with DAML-S”. One of the differences is that in this approach an OWL-S is used to add semantic to UDDI. In this case the OWL-S Profile is mapped with the Business Service part that contains information about the company which develops the service and Business Entity part that contains Tmodels, which enables them to map capabilities of the service in the OWL-S profile with UDDI, of the UDDI. A more detailed explanation of this registry can be found in [46].

7.3 Benchmarking of Search Mechanisms

You can see the benchmarking of existing service search mechanisms in Figure 7.1.

Nowadays, it is hard to find a UDDI that can be accessed easily, because most of them are closed and the one that are not closed need a payment to be used. For that reason, accessibility of UDDI is “Expensive”. Since

UDDI with DAML-S and UDDI with OWL-S approaches are based on UDDI, their accesibility is also “Expensive”. However, WebServicesX.NET and our application can be reached by anyone easily, since they are “Free” applications. On the other hand, in order to use the Yahoo! API, you should be a member and get an id. Thus, it’s accesility depend on “Membership”.

As explained earlier, efficiency shows how hard to locate the services to the registry. In order to locate your services to UDDI, the owner of the UDDI should accept you. Also, you have to categorize your services. Thus, it is not enough wanting to publish your services. You also have to be accepted. For semantic UDDIs, you also should make a semantic matching. For these reasons, efficiency for UDDI, UDDI with OWL-S and UDDI with DAML-S is “Hard”. Since the locating process and the tagging of services is done manually, it is not easy to locate services into our application, too. Thus, the effiiency for our application is also “Hard”. On the other hand, WebServicesX.Net and Yahoo! API publish the services that they developed, it is easy for them to locate their services.

UDDI, UDDI with OWL-S, UDDI with DAML-S and our application is centrilized registries. They have only one storage. On the other hand, WebServicesX.Net and Yahoo! API are distributed systems.

UDDI is a standart for web service discovery. Companies, publish their services to UDDI in order to be discovered. Since, UDDI contains web services from different companies, their web service structure is “Distributed”. WebServicesX.Net and Yahoo! API contain only services that they developed. Thus, their service structure is “Local”. Our application stores services that are developed by both other companies and by us.

UDDI, UDDI with OWL-S, UDDI with DAML-S, WebServicesX.Net store only web services in their registry. Yahoo! API stores only restful services. Our application stores both web services and restful services.

We determined ease of use criteria according to how friendly is the user interface of the registry. WebServicesX.Net and Yahoo! API applications have very simple interface, since they only list services according to their criterias. The user does not need to enter any criteria, all he needs to do is selecting the criteria and determining which service to use. The user interface of our application is a little complex than these applications, since the user should enter search criteria according to the service he needs. UDDI is way difficult than our application, because you need to have a basic knowledge about UDDI, since some terminology is used in UDDI interfaces.

UDDI, WebServicesX.Net, Yahoo! API does not make semantic search. UDDI with OWL-S, UDDI with DAML-S and our application make semantic search using OWL-S, DAML-S and OWL ontologies respectively.

UDDI enables web services to be searched according to their service characteristics. However, this search does not include any semantic. It is a key word based matching search mechanism. For example, if a search is made by typing “Transport”, the services that are labeled as “Car Rental” would not be returnd, since the searching mechanism does not look for the relation between the transport and car words. Also, you need to be a member and pay some money to be able to use the UDDI registry. This is also true for publishing your service. Our semantic service discovery approach overcome these difficulties of UDDI by making a semantic search and being free. Also, making a search is more simpler.

WebServiceX.NET and Yahoo! API are applications that publish the services that they developed. In these applications, the services are listed according to their categories. Thus, there is neither a searching mechanisms nor semantic. Our application can be considered as a union

of these two, since it makes both web service and restful service discovery. As a matter of fact, our application also contains the services that are discovered by these two applications.

UDDI with DAML-S and UDDI with OWL-S also makes semantic search. However, they only search for web services. Besides web services, our application also discover restful services. Also, the matching algorithm of UDDI with DAML-S and UDDI with OWL-S differs from our algorithm. They look whether they find a match and if there is a match what kind of match is it (exact, subsume or plugin). However, our application calculates a relevance degree by considering certain parameters.

Service Registry Name	Accessibility	Efficiency	Web Registry Type	Web Service Structure	Service Type	Ease Of Use	Use of Semantics
UDDI	Expensive	High	Centralized	Distributed	Web Service	Hard	No
WebServicesX.NET	Free	Low	Distributed	Local	Web Service	Easy	No
Yahoo! API	Membership	Low	Distributed	Local	Restful Servcie	Easy	No
UDDI with DAML-S	Expensive	High	Centralized	Distributed	Web Service	Hard	Yes/ DAML-S
UDDI with OWL-S	Expensive	High	Centralized	Distributed	Web Service	Hard	Yes/ OWL-S
Our Application	Free	High	Centralized	Both	Both	Medium	Yes/ OWL

Figure 7. 1 Benchmarking of Search Mechanisms

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Summary and Conclusions

In this thesis, we have proposed an approach and its implementation in order to be used in web service discovery problem. We developed a system that discovers services semantically using OWL and calculates a heuristic relevance degree for the services discovered. Then, we make a comparison of our relevance calculation method with the existing ranking methods and realize that the results of our algorithm is very close to the results of other mechanisms. The results of our algorithm realize words as more semantically related than the Dice's coeffient. On the other hand, the results of approach in [19] makes a more sensitive calculation than ours.

We have observed the difficulty of finding web services, since the usage of most of the services demand fee and discovery mechanisms like UDDI have some drawbacks like being lack of semantics. Thus, we try to suggest a mechanism that solves some of the existing problems of discovery mechanism.

Our approach aims to find the most satisfactory web service and/or RESTful service that the user has requested for. The user of our application submits the inputs, which are necessary for finding the most suitable service, to the system. According to the submitted data our application returns a list of services that are closest to the user's search criteria. The user can make search according to the functionality of the web service, the inputs and the outputs of the web service.

OWL is used while trying to find a match between the users input and the advertised data. We have developed an OWL file for tourism domain. Our system uses OWL in order to find the relations between the advertised services. Thus, we can provide a more efficient and correct list of services.

Our application provides the first semantic service registry that both web services and restful services can be discovered. It is shown that by using OWL as the ontology, we can discover services semantically. Our application is easy to use, since it has a user friendly interface, and anyone can use this application since it is fee. Also, our application proves that it is possible to make a registry that calculates relevance while discovering web and restful services. Thus, by this way, our tool helps users to select from the resulted services if more than one service is discovered.

During this thesis study, we saw that our approach achieved the goal of being an alternate to UDDI, since it is semantically aided, user friendly and free. However, one drawback of our system is that we can not provide a place to keep the WSDL definitions of web services. Thus, if web services do not have an WSDL address, then we can not store these services. Also, the relevance calculation algorithm that we developed is able to correspond our needs. Although it is not based on sensitive values, we saw that it produces results that are close to the existing ranking researches.

8.2 Future Work

The ontology that is used in this work contains only tourism related data. Thus, this work can be mainly used by developers that are searching web services while developing a tourism related web page. In the future, the ontology and the web service list can be extended to include more services and thus domains. Thus, it can be used by more people.

Our work focuses on the functional properties of web services such as its inputs and outputs while searching an appropriate service. However, it would be more efficient if non-functional properties of a web service, such as response time, or maintainance, Quality of a Service is added as a searching criteria. Research about the QoS of web services [3] and reputation of web services [22] exist.

Language parsing has a very important place in our work, because the user of the interface can enter a word which is a synonym of a word in the ontology. Our application should understand what the user is trying to search in order to find the most appropriate service. For now, we have just included a class that maps the input words with their synonyms. It would be more efficient if opposite words, closest words and roots of words also contribute to the match. Research on language parsing still continues, with the involvement of an efficient language parser, our application will be more effective.

The algorithm for calculating the relevance of the web services can be improved so that it can return a more detailed result. Currently, we are only using the generalization or specification of the user's input. However, in the future, we can also add the degree of generalization and specification to our relevance calculation.

Another point is that, we build the web service database manually. This means that the internet is browsed and the appropriate services are selected manually. In the future, it would be good to have a web service crawler so that the web service storage can be formed automatically. Also, with this crawler, we can control the availability of the service. Thus, if the service is no longer available it can be removed from the storage.

Besides adding the existing web services and RESTful services to our service database, in the future the services that are developed and implemented by us can be added to the database. If the number of

available and free services increase the search will be affected less in the situations like when a web service does no longer exist.

Lastly, we can improve our application so that if no matching service is returned by the matching algorithm, our mechanism can try to form a new service that is capable of meeting the user's request, by combining the existing services.

REFERENCES

1. Aabhas V. Paliwal,Nabil R. Adam,Hui Xiong,Christof Bornhövd, Web Service Discovery via Semantic Association Ranking and Hyperclique Pattern Discovery, IEEE/WIC/ACM International Conference on Web Intelligence, 2006

2. Akkiraju Rama, Goodwin Richard, Doshi Prashant, Roeder Sascha, A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI.

3. Al-Masri E., and Q. H. Mahmoud, Discovering the best web service, 16th International World Wide Web Conference (WWW), pp. 1257-1258, 2007.

4. Al-Masri Eyhab, Mahmoud Qusay H., Investigating Web Services on the World Wide Web, WWW 2008 / Refereed Track: Web Engineering - Web Service Deployment, 2008

5. Bachlechner, D., Lausen, H., Siorpaes, K., Fensel, D., Web Service Discovery-A Reality Check, Third Annual European Semantic Web Conference ESWC'06, 2006.

6. Based Semantic Search in UDDI, SWSWPC 2004 UDDI, Universal Description, Discovery and Integration, The UDDI Technical White Paper, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, August 2008

7. Beckett Dave, RDF/XML Syntax Specification, W3C Recommendation 10 February 2004, W3C Technical Reports and Publications, http://www.w3.org/TR/rdf-syntax-grammar/

8. Benatallah Boualem, Hacid Mohand-Said, Leger Alain, Rey Christophe, Toumani Farouk, On automating Web services discovery, VLDB Journal, 2005.

9. Benatallah Boualem, Hacid Mohand-Said, Rey Christophe, Toumani Farouk, Request Rewriting-Based Web Service Discovery, LNCS 2870, pp. 242–257, 2003.

10. Benatallah Boualem, Hacid Mohand-Said, Rey Christophe, Toumani Farouk, Semantic Reasoning for Web Services Discovery, Budapest – ESSW, 2003.

11. Benatallah B., Dumas M., Fauvet, Rabhi F.A. Towards Pattern of Web Services Compositon, November 2001.

12. Bhavani Thuraisingham, Security standards for the semantic web, Computer Standards & Interfaces, Volume 27, Issue 3. March 2005.

13. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., Web Services Architecture, W3C Working Group Note 11 February 2004, W3C Technical Reports and Publications, http://www.w3.org/TR/ws-arch/ , August 2008.

14. Brahim Medjahed, Athman Bouguettaya, Ahmed K. Elmagarmid. Composing Web Services on the Semantic Web, September 2003.

15. Brickley Dan, Guha R. V., RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, W3C Technical Reports and Publications, http://www.w3.org/TR/rdf-schema/

16. Casati, F., Ilnicki, S., and Jin, L., Adaptive and Dynamic Service Composition in eFlow, Proceedings of 12th Int. Conference on Advanced Information Systems Engineering(CAiSE), 2000.

17. Chakraborty Dipanjan, Perich Filip, Avancha Sasikanth, Joshi Anupam, DReggie: Semantic Service Discovery for M-Commerce Applications

18. Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, W3C Technical Reports and Publications, http://www.w3.org/TR/wsdl/ , August 2008.

19. Dimitrios Skoutas, Alkis Simitsis, Timos Sellis,A Ranking Mechanism for SemanticWeb Service Discovery

20. Di Noia Tommaso, Di Sciascio Eugenio, Donini F. M., Mongiello Marina, Semantic Matchmaking In a P-2-P Electronic Marketplace, ACM, 2003.

21. Dustdar, S., and Schreiner, W., A Survey on Web Services Composition, Int. J. Web Grid Serv. 1 (1), pp. 1–30, 2005.

22. E. Maximilien and M. Singh, "Conceptual model of Web service reputation," ACM SIGMOD Record, 31(4), 2002.

23. Fensel Dieter, Lausen Holger, Polleres Axel, Jos de Bruijn, Stollberg Michael, Roman Dumitru, Domingue John, Enabling Semantic Web Services – The Web Service Modeling Ontology.

24. Fox Joshua, Borenstein Joram, Semantic Discovery for Web Services, Web Services Journal, March 2003.

25. Garofalakis John, Panagis Yannis, Sakkopoulos Evangelos, Tsakalidis Athanasios, Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?, 2004

26. Hakimpour Farshad, Cong Suo, Damm Daniela E. , A Practical Tutorial on Semantic Web Services.

27. Horrocks Ian, Patel-Schneider Peter, Van Harmelen Frank, From SHIQ and RDF to OWL: the making of a Web Ontology Language, Web Semantics: Science, Services and Agents on the World Wide Web Volume 1, Issue 1, Pages 7-26, December 2003

28. Hongen Lu, Semantic Web Services Discovery and Ranking, IEEE/WIC/ACM International Conference on Web Intelligence, 2005

29. Jay J. Jiang, David W. Conrath, Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy, International Conference Research on Computational Linguistics, 1997

30. Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/, Last Access Date on 14 November 2009

31. Jose Marıa Garcıa, Ioan Toma, David Ruiz, Antonio Ruiz-Cortes, Ying Ding, and Juan Miguel Gomez, Ranking Semantic Web Services Using Rules Evaluation and Constraint Programming

32. Kırcı E., Automatic Compositon of Semantic Web Services With the Abductive Event Calculus, M.S. Thesis, METU, 2008.

33. Kifer Michael, Lara Rub´en, Polleres Axel, Zhao Chang, Keller Uwe, Lausen Holger, Fensel Dieter, A Logical Framework for Web Service Discovery.

34. Klusch Matthias, Fries Benedikt, Sycara Katia. Automated Semantic Web Service Discovery with OWLS-MX. AAMAS 2006, May 2006.

35. McIlraith S., Son T.C., Zeng H., Semantic web services, IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2):46/53, March/April 2001.

36. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K., OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004, Acknowledged Member Submissions to W3C.

37. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D.,McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K., OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004, Acknowledged Member Submissions to W3C, http://www.w3.org/Submission/OWL-S/ , August 2008

38. Martin David,Burstein Mark,Lassila Ora, Paolucci Massimo, Payne Terry, McIlraith Sheila, Describing Web Services using OWL-S and WSDL, From http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html

39. Martin David, Paolucci Massimo, McIlraith Sheila, Burstein Mark, McDermott Drew, McGuinness Deborah, Parsia Bijan, Payne Terry, Sabou Marta, Solanki Monika, Srinivasan Naveen, Sycara Katia, Bringing Semantics to Web Services: The OWL-S Approach, LNCS 3387, pp. 26 – 42, 2005.

40. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Importing the Semantic Web in UDDI. In Proceedings of Web Services, E-business and Semantic Web Workshop, 2002

41. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In ISWC2002, 2002

42. McGuinness Deborah L., van Harmelen Frank, OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/

43. Moreau Luc, Miles Simon, Papay Juri, Decker Keith, Payne Terry, Publishing Semantic Descriptions of Services, From http://www.semanticgrid.org/GGF/ggf9/luc/

44. Natenapa Sriharee, Semantic Web Services Discovery Using Ontology-Based Rating Model

45. Naveen Srinivasan, Massimo Paolucci , Katia Sycara. Adding OWL-S to UDDI, implementation and throughput. First International Workshop on Semantic Web Services, 2004

46. Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. An Efficient Algorithm for OWL-S Based Seamntic Search in UDDI.

47. Oh, S.C., Lee, D., and Kumara, S., A Comparative Illustration of AI Planning-based Web Service Composition, ACM SIGecom Exchanges, 5(5), pp. 1-10, 2006.

48. Omid Mola, Peyman Emamian, Mohammadreza Razzazi, A Vector Based Algorithm for Semantic Web Services Ranking, IEEE

49. Papazoglou Micheal P., Web Services: Principles And Technology, Pearson Education Limited, 2008

50. Pathak Jyotishman, Koul Neeraj, Caragea Doina, Honavar Vasant G., A Framework for Semantic Web Services Discovery, WIDM'05, November 5, 2005.

51. Protege, http://protege.stanford.edu/, Last Access Date on 14 November 2009

52. Rao, J., and Su, X., A Survey of Automated Web Service Composition Methods, Proceedings of First International Workshop on Semantic Web Services and Web Process Composition, pp 43-54, 2004.

53. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. Proceedings of ACM SIGCOMM`01, San Diego, September (2001)

54. Rowstron, A., Druschel, P., Pastry: scalable, decentralized object location and routing for largescale peer-to-peer systems. Proc. of the 18th IFIP/ACM Middleware,Germany. (2001)

55. Sang Keun Rhee, Jihye Lee, and Myon-Woong Park, Ontology-based Semantic Relevance Measure

56. Semantic Web Architecture, http://obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html , Last Access Date on 13 November 2009

57. Sheth Amit, Miller John A., Web Services: Technical Evolution yet Practical Revolution?, IEEE INTELLIGENT SYSTEMS, 2003.

58. Sivashanmugam K., Verma K., Sheth A., Discovery of Web Services in a Federated Registry Environment, 2004

59. Sivashanmugam K., Verma K., Sheth A., Miller J.. Adding semantics to web services standards.In 1st Int. Conf. on Web Services (ICWS'03), pp. 395–401, June 2003.

60. Srivastava Biplav, Koehler Jana, Web Service Composition - Current Solutions and Open Problems.

61. Srinivasan Naveen, Paolucci Massimo, Sycara Katia, An Efficient Algorithm for OWL-S Based Semantic Search in UDDI, LNCS 3387, pp. 96 – 110, 2005.

62. Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, Proceedings of ACM SIGCOMM'01 Conference, pp.149–160, 2001.

63. StrikeIron, http://www.strikeiron.com/Home.aspx, Last Access Date on 15 November 2009

64. Sycara Katia, Paolucci Massimo , Ankolekar Anupriya, Srinivasan Naveen, Automated discovery, interaction and composition of Semantic Web services, Web Semantics: Science, Services and Agents on the World Wide Web 27–46, 2003.

65. Thuraisingham Bhavani, Security standards for the semantic web, Computer Standards & Interfaces 27 pg: 257–268, 2005.

66. WebServiceX.NET, http://www.webservicex.net/, Last Access Date on 15 February 2010

67. Web Service List, http://www.webservicelist.com/, Last Access Date on 15 November 2009

68. Wikipedia, Cosine similarity, http://en.wikipedia.org/wiki/Cosine_similarity , Last Access Date on 10 February 2010

69. Wikipedia, Dice's coefficient, http://en.wikipedia.org/wiki/Dice's_coefficient , Last Access Date on 10 February 2010

70. Wikipedia, Extensible Markup Language, http://en.wikipedia.org/wiki/XML, Last Access Date on 14 November 2009

71. Wikipedia, Jaccard index, http://en.wikipedia.org/wiki/Jaccard_index, Last Access Date on 10 February 2010

72. Wikipedia, Ontology, http://en.wikipedia.org/wiki/Ontology_(computer_science), Last Access Date on 15 November 2009

73. Wikipedia, OWL-S, http://en.wikipedia.org/wiki/OWL-S, Last Access Date on 20 October 2009

74. Wikipedia, Resource Description Framework, http://en.wikipedia.org/wiki/Resource_Description_Framework, Last Access Date on 14 November 2009

75. Wikipedia, Representational State Transfer, http://en.wikipedia.org/wiki/Representational_State_Transfer#Key_goals_of_REST, Last Access Date on 14 November 2009

76. Wikipedia, Semantic Web, http://en.wikipedia.org/wiki/Semantic_Web, Last Access Date on 15 November 2009

77. Wikipedia, Semantic Web Services, http://en.wikipedia.org/wiki/Semantic_Web_Services , Last Access Date on 20 October 2009

78. Wikipedia, Service- oriented Architecture, http://en.wikipedia.org/wiki/Service-oriented_architecture, Last Access Date on 15 November 2009

79. Wikipedia, Universal Description Discovery and Integration, http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration, Last Access Date on 20 October 2009

80. Wikipedia, Web Services Description Language , http://en.wikipedia.org/wiki/Web_Services_Description_Language , Last Access Date on 15 November 2009

81. Wikipedia, Web Service Discovery , http://en.wikipedia.org/wiki/Web_Services_Discovery, Last Access Date on 15 November 2009,

82. Xiao Min, Zhong Luo, Xiong Qianxing, Semantic Similarity between Concepts based on OWL Ontologies, IEEE, 2009

83. Xmethods, http://www.xmethods.com/, Last Access Date on 20 November 2009

84. Yahoo! API, http://developer.yahoo.com/everything.html, Last Access Date on 15 November 2009