A Mini-Project Report on

Automated Retrieval of Semantic Web Services

carried out as part of the course Web Services (*IT* 450)

Submitted by

Adarsh Mohata - 12IT03 - *V* Sem B.Tech

Ajith P S - 12IT04 - *V* Sem B.Tech

Ashish Kedia - 12IT14 - *V* Sem B.Tech

in partial fulfillment for the award of the degree

of

Bachelor of Technology

in

Information Technology

Department of Information Technology

National Institute of Technology Karnataka Surathkal

April 2015

Web Services Mini Project
Project Approval Certificate

Department of Information Technology

National Institute of Technology Karnataka Surathkal

This is to certify that the project entitled **Automated Retrieval of Semantic Web Services** is a bona-fide work carried out as part of the course Web Services (IT 450), under my guidance by **Adarsh Mohata**, **Ajith P S** and **Ashish Kedia** of VI^{th} Sem B.Tech (IT) at the Department of Information Technology, National Institute of Technology Karnataka, Surathkal, during the sixth academic semester in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, at NITK Surathkal.

Mrs Sowmya Kamath
Dept. of IT, NITK
Project Mentor

Place: NITK Surathkal, Mangalore
Date: May 6, 2015

Declaration

We hereby declare that the project entitled **Automated Retrieval of Semantic Web Services** submitted as part of the partial course requirements for the course **Web Services (IT 450)** for the award of the degree of Bachelor of Technology in Information Technology at National Institute of Technology Karnataka, Surathkal during the sixth academic semester (January, 2015 - April, 2015) has been carried out by us and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We declare that the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles elsewhere.
Further, We declare that we will not share, re-submit or publish the code, idea, framework and/or any publication that may arise out of this work for academic or profit purposes without obtaining the prior written consent of the course Faculty Mentor and Course Instructor

Mr. Adarsh Mohata
Department of Information Technology
National Institute of Technology Karnataka, Surathkal

Mr. Ajith P S
Department of Information Technology
National Institute of Technology Karnataka, Surathkal

Mr. Ashish Kedia
Department of Information Technology
National Institute of Technology Karnataka, Surathkal

Date: ________________________

Acknowledgements

We have put our best efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We are highly indebted to **Sowmya Kamath Ma'am** for his guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project during it's initial phase. We would specially like to thank her for providing us with sample data-set to carry out our experiments.

We would like to express our gratitude towards the members of Department of Information Technology for their kind co-operation and encouragement which helped us in the completion of this project. Our thanks and appreciations also go to our colleagues in other groups in developing the project and people who have willingly helped us out with their abilities.

Abstract

As Web services increasingly become important in area of distributed computing, some of the flaws and limitations of this technology become more and more obvious. One of these flaw is the discovery of Web Services through common methods. Research has been pursued in the field of ”Semantic Web services”. This research is driven by the idea, to describe the functionality of Web services as accurately as possible using natural language as well as ontologies and to create programs automatically out of already existing Web services.

Consideration of the web service interfaces(input/output) is also equally important. It is possible to index web services available on web and find other web services with similar interfaces. Web Service with similar interfaces are likely to replace each other.

This project aims at exploring the various ways of discovering, indexing and searching semantic web services based on their corresponding description documents. The project also aims to recommend possible compositions of available services to the user which can which can together meet the requirements whenever no suitable service can meet the requirements alone.

Keywords : Web Service Discovery, WSDL, OWL-S, Co-Sine Similarity, Word-Sense Disambiguation

List of Figures

List of Tables

Contents

Chapter 1

Introduction

A web service is a software system identified by a URI whose public interfaces are defined and described using XML based documents. It interacts with other systems in a manner prescribed by it's definition thereby satisfying one or more needs of the other system by performing required computation.

Web Service technologies have advanced the development of Internet-based applications by facilitating the integration and interoperability of services provided by different organizations. As a result, instead of developing monolithic applications, today, we build large-scale software applications by composing loosely coupled services which can scale with the size of the web-application. By doing so, we can reuse and select suitable services as various organizations can provide similar services for the development of different applications [2].

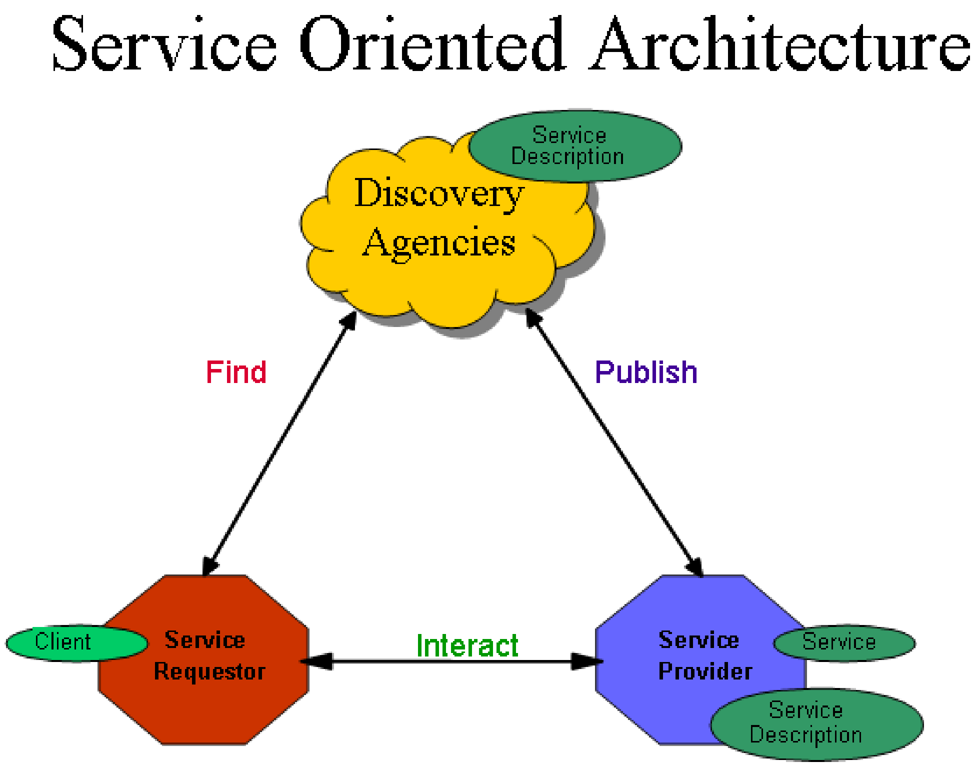


Figure 1.1: **Service Oriented Architecture [1]**

1.1 Web Service Discovery

When services are loosely coupled to provide functions for a particular application, it is not easy to search for the best available for a given task/application. Web Service Discovery and Retrieval often becomes a bottleneck.

Searching proper web services is the basic step to composite web services into applications. The requirement of automated web service discovery arises in many Service Oriented Computing applications. Recent years have seen a tremendous growth in number of web services over Internet and the various standards in which the services are being described which make automated service discovery challenging and cumbersome task.

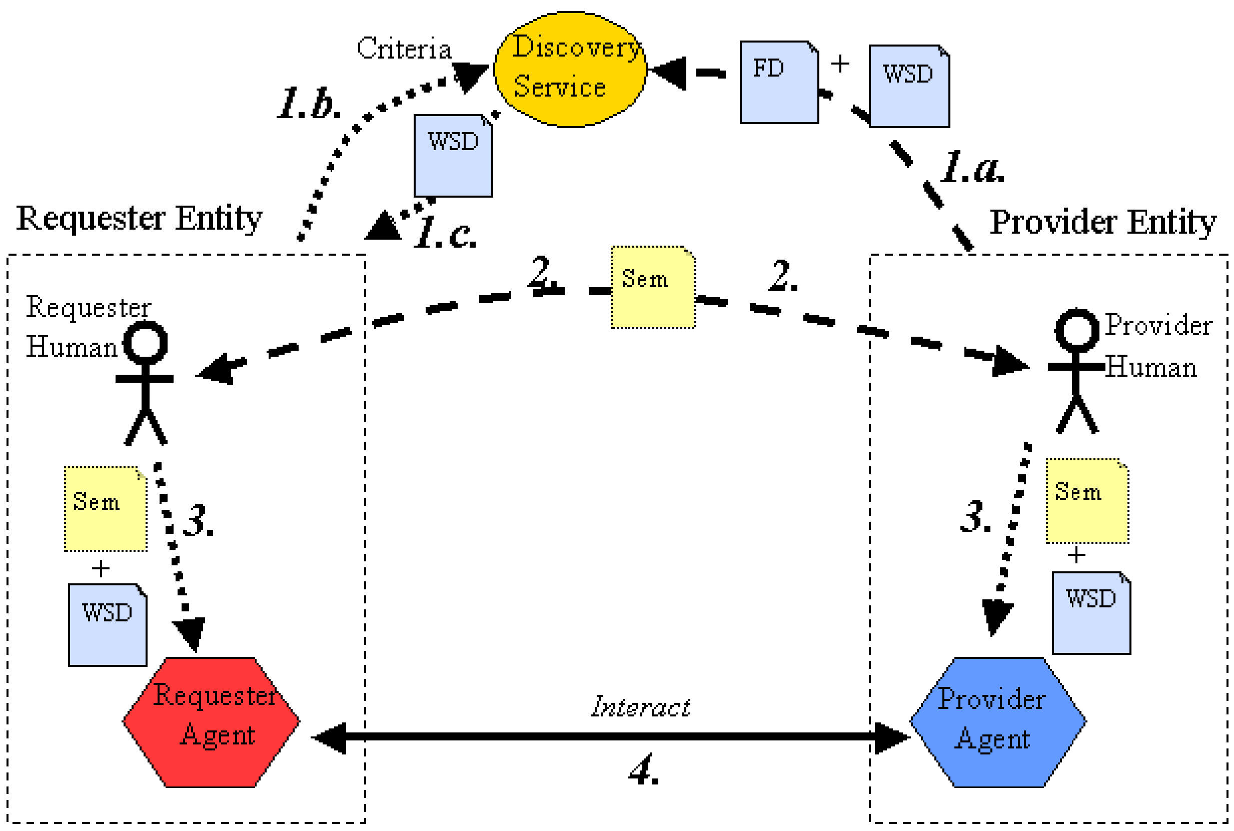


Figure 1.2: **Web Service Discovery Process**

1.2 Semantic Web Services

Semantic Web technology is a promising first step for automated service discovery. Most current approaches for web service discovery cater to semantic web services, i.e., web services that have semantic tagged descriptions. It is difficult, however, to expect all new services to have semantic descriptions associated. Furthermore, the descriptions of the vast majority of already existing services do not have explicitly associated semantics. There are also severe restrictions on the prospective conversion of existing, non-semantic, descriptions, e.g., Web Service Description Language (WSDL), to corresponding semantic descriptions, e.g., OWL-based Web Service Ontology (OWL-S).

1.3 Purpose of the Project

The purpose of this project is to study the existing methods of discovering web services and implement a prototype semantic web service search engine which should index a set of web services using their descriptive documents and discover appropriate services based on user's need.

The purpose of this document is to give an overview about the work done and challenges faced by us during the course of this project.

1.4 Abbreviations and Definitions

Table 1.1 lists all the common abbreviations used throughout the length of this document and their corresponding definition.

Table 1.1: **List of Abbreviations**

Abbv	**Definition**
PHP	PHP: Hypertext PreProcessor
HTML	Hyper Text Markup Language
JS	JavaScript
WSDL	Web Service Description Language
OWLS	Ontology Web Language Services
URI	Uniform Resource Identifier
UDDI	Universal Description, Discovery and Integration
RDF	Resource Description Framework
WN	Word Net
POS	Parts of Speech
DAG	Directed Acyclic Graph
DFS	Depth First Search

Chapter 2

Literary Survey

We carried out an extensive literary survey at the start of the project. We found out the existing methods of discovering web services and their shortcomings.

2.1 Background

Different approaches have been developed to solve the web service searching problem. Certain highlights of our literature survey are as follows :

Assignment Algorithm Method

Yanbin et. all [3] proposes an improved semantic web service discovery method based on assignment algorithm, which can convert service discovery problem into assignment problem using functional constraints. It uses 3 step match making - Service Library Matchmaking, Service Matchmaking and Operation Matchmaking(Interface Match Making and Concept Match Making) which has been illustrated in figure 2.2.

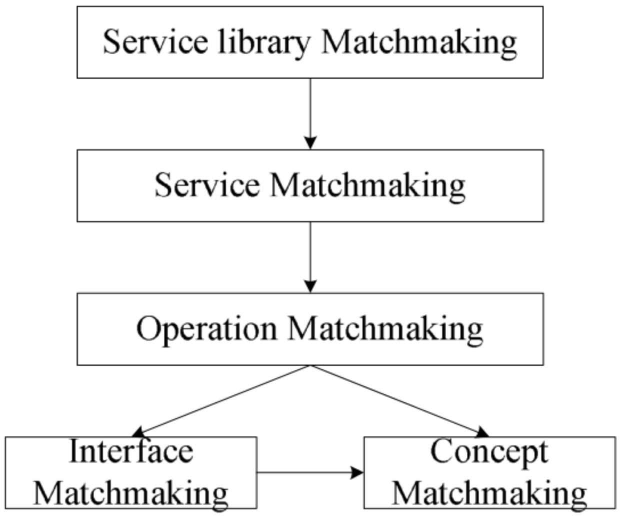


Figure 2.1: **Execution process of service discovery algorithm**

Vector Space Search Engine

Christian Platzer and Schahram Dustdar [4] have used a Vector Space Search Engine to index descriptions of already composed services. It is a combination of common information retrieval methods and existing standards for the description of Web services. A document can be seen as a vector within a "term space" where each dimension is a keyword. The position of this vector relative to other vectors within the same vector space describes their similarity to each other.

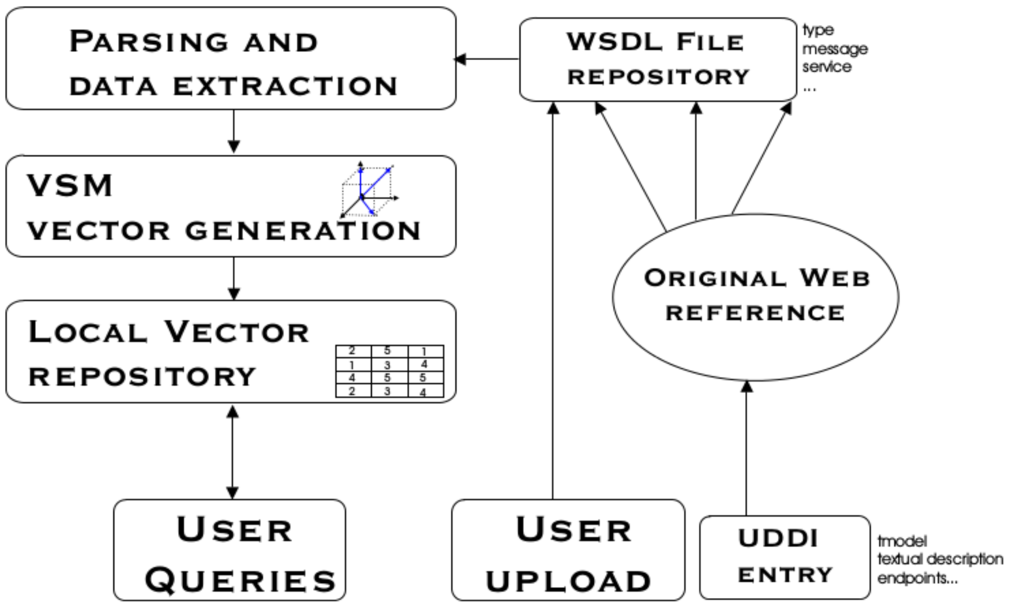


Figure 2.2: **Architecture of Vector Space Search Engine**

Advance Graph Based Matching

Cuzzocrea et. al [5] considers both internal structure and component services. They describe a method for using graph-based representation of composite OWL-S processes. They also proposed an algorithm that matches over such graph-based representations and computes their degree of matching by combining the similarity of the atomic services they comprise and the similarity of the control flow among them.

Natural Language Processing

Jordy Sangers et. al [6] have used techniques like part-of-speech tagging, lemmatization, and word sense disambiguation to determine the senses of the relevant words gathered from web service descriptions and user queries and then carry out a matching process between user's query and indexed web services. The architecture of such framework has been illustrated in figure 2.3. This method is promising in a sense that a context aware seach is performed i.e., actual user's need is matched with services that performs required computation.

Matching Based on Conceptual Indexation

Fethallah et. all [7] propose an approach which exploits the service interface (inputs/outputs) and the domain ontology, in order to index web services conceptually. After that, they compute a similarity score between the request and the indexed web services through the cosine measure. The related paper has been attached as an appendix.

2.2 Outcome of Literature Survey

After gaining an insight about the popular existing techniques used to discover web services we came to the following conclusions :

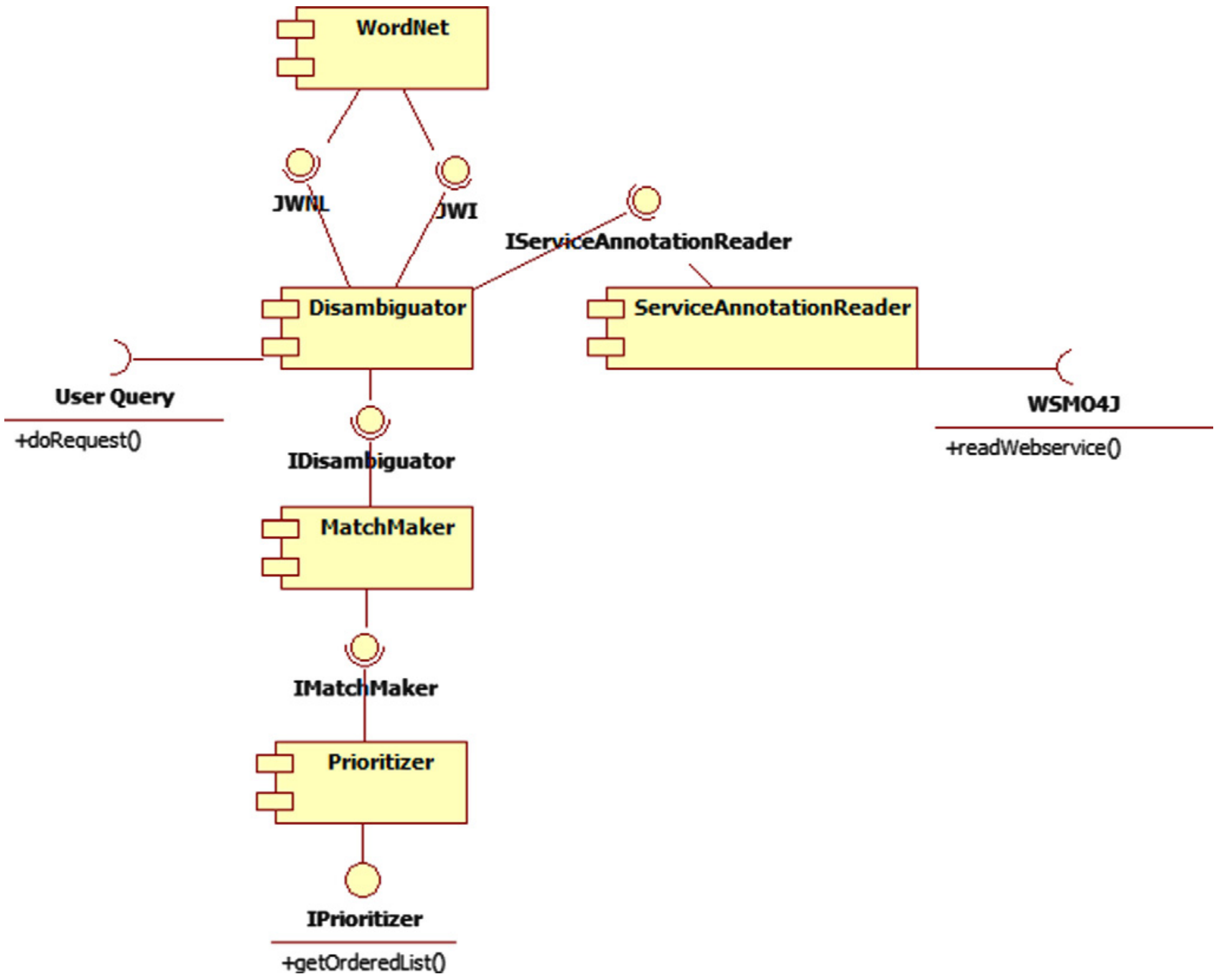


Figure 2.3: **NLP based Web Service Discovery Framework**

- Public repositories of UDDI registries are now obsolete. Modern web services are described using ontology based OWLS and RDF.
- Vector Space Search seem like a promising approach, however it does not consider service interfaces or service semantics
- Using techniques like Natural Language Processing was out of the project's scope given the time constraint
- We decided to build a prototype system using conceptual indexation of service interfaces
- We also had an intuition that vector space search along with cosine similarity would yield a better result

2.3 Problem Statement

Retrieve description documents of various web services - WSDL and OWLS, available on the Internet and then index them efficiently so as to discover appropriate web services or a group of web services based on the desired service interface or keywords specified by the user's query. The system should provide users with a feature that can tune or shortlist the results.

2.4 Objectives

A major part of this work is to come up with techniques that can enhance the existing web service discovery and retrieval methods. The objectives of the project are as follows :

- Retrieve XML (WSDL and OWL-S) documents that describes the web services
- Selection of Service Categories or Vector Fields
- Adding keywords related to each category / vector field
- Extracting interfaces of Web Services using their descriptive documents
- Extracting text description of the web service - service description in natural language
- Indexing the retrieved web services and it's interfaces conceptually
- Disambiguating the senses of the words in the text description and establishing a context
- Indexing services using established context
- Retrieving user's required interfaces and search parameters
- Disambiguating user's query and establishing the context of user's query
- Finding similar web services and returning the results to users
- Searching for multiple web services that can be composed together to serve user's need

We have focused on discovering an appropriate semantic web service(s) for a given application, given the required interfaces i.e., input and output format. We use both vector based method and semantic based context matching.

Chapter 3

Methodology

This chapter describes the methodology used for achieving the desired objectives of the project. The following sections gives a brief overview of the techniques and algorithms used as a part of our project.

3.1 Retrieving Web Service Description Files

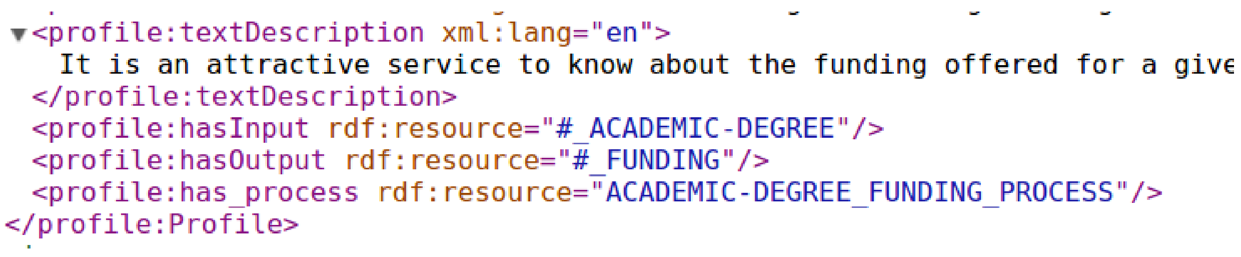
Any system which is based on indexing of files needs a data set. In our system the data set consist of valid WSDL and OWLS files. Initially we tried retrieving the WSDL files through a Web Crawler but it was very time consuming and the resulting number of WSDL file were not so impressive. We then searched for UDDI registries which would contain lots of WSDL files. Since all the major UDDI registries were down we could only find three registries which were still functional. We successfully retrieved 470 WSDL files. All the WSDL files were downloaded using Cross Domain Scripts. Many difficulties were faced during this period, the details of which can be found in the implementation section. Unfortunately, WSDL files are obsolete and are rarely used nowadays. The new method of describing Web Services is through OWLS files. We obtained an OWLS data set through our Project Mentor.

3.2 Indexing Description Files using Keywords

Each and every Web Service has, in their profile part, a set of elements named <<**profile:hasinput**>> and <<**profile:hasoutput**>>. These elements specify the respective interface of the corresponding web service. We use these interfaces to index the web services conceptually. A sample OWLS file defining the interfaces of it's corresponding web services has been shown in figure 3.1

In general any OWLS documents in our dataset can be segmented into one or more of the following classes:

1. Economy



```
▼<profile:textDescription xml:lang="en">
   It is an attractive service to know about the funding offered for a give
 </profile:textDescription>
 <profile:hasInput rdf:resource="#_ACADEMIC-DEGREE"/>
 <profile:hasOutput rdf:resource="#_FUNDING"/>
 <profile:has_process rdf:resource="ACADEMIC-DEGREE_FUNDING_PROCESS"/>
</profile:Profile>
```

Figure 3.1: **A sample OWLS file showing interface of the service**

2. Communication
3. Education
4. Food
5. Weapon
6. Medical
7. Travel

To index a single OWLS document, we created a OWLS Document Parser. It parses the OWLS document and extracts the required elements i.e <<**profile:hasinput**>> and <<**profile:hasoutput**>>.

We create 2 vectors namely V_i and V_o. Vector V_i contains the inputs and the concepts which sum them up in the ontology (each concept will have a number which represents its frequency of occurrence in the <<**profile:hasinput**>> element of the OWLS document to which it belongs).

We apply the same procedure to the <<**profile:hasoutput**>> element of the OWLS document and obtain the vector V_o. Thus we obtain a pair of vectors which represents an OWLS document in a vector space of 7 dimensions where each dimension is a concept. We repeated this procedure for each and every OWLS file in our data set and the resulting data contains vector representation of every OWLS document that we have in our data set. This concludes the indexing of OWLS documents and thus the corresponding web service.

3.3 Indexing Description Files using Word Senses

The overall functionality of a web-service is typically mentioned as a part of the text-description document in natural languages. They tend to give more insight about the actual functionality provided by the web service. We used techniques like lemmatization and parts of speech tagging to determine the multiple meaning of the words in those description document. We used a simple word sense disambiguation algorithm to choose the right meanings of all the words in the description and then establish a context of the web service which can be used for matching the service with user's requirements. The algorithm simply chooses the meaning which is conceptually most similar to the already established context (i.e., already disambiguated words). To find similarity between words we use the JCN Similarity algorithm [8]. As such the final context has the set of most similar lemmas. We index each web service with using it's corresponding set of Synset.

3.4 Searching Web Services

We intended to create a framework through which a user could retrieve an OWLS document based on his/her needs as provided through his/her inputs. This input is nothing but the interface of the web service. For that we needed to have the capability of searching our data set, given the user's query. We have used 2 approaches for two different kinds of indexing that we perform :

Co-Sine Similarity Search

Our system takes the user's input and index it to obtain two vectors S_i and S_o similar to the procedure mentioned previously in the case of indexation of OWLS documents.

After obtaining the two vectors we used the method of **Cosine Similarity** as given by the following formula to find similar services.

$$cos(A, B) = \frac{< A, B >}{||A|| * ||B||} \tag{3.1}$$

where, A and B are two vectors, $< A, B >$ denotes the scalar product of two vectors A and B and $||V||$ denotes magnitude of a vector V.

For illustration purpose, let us consider 2 vectors $V_1 = [1, 2, 0, 3, 4, 2, 1]$, $V_2 = [1, 1, 1, 2, 3, 1, 1]$. We can calculate the similarity between these two vector as :

$$cos(V_1, V_2) = \frac{V_1.V_2}{||V_1|| * ||V_2||}$$

$$= \frac{(1*1) + (2*1) + (0*1) + (3*2) + (4*3) + (2*1) + (1*1)}{\sqrt{1^2 + 2^2 + 0^2 + 3^2 + 4^2 + 2^2 + 1^2} * \sqrt{1^2 + 1^2 + 1^2 + 2^2 + 3^2 + 1^2 + 1^2}}$$

$$= \frac{1 + 2 + 0 + 6 + 12 + 2 + 1}{\sqrt{(}35) * \sqrt{(}18)} = \frac{24}{25.1} = 0.9561 \tag{3.2}$$

Once we have the vector representation of user's query(R_i, R_o), we compare the vector representation of user's query with every OWLS documents in our dataset. We calculate the results as follows:

$$Score1 = cos(R_i, S_i) \tag{3.3}$$

$$Score2 = cos(R_o, S_o) \tag{3.4}$$

Where, S_i and S_o are indexed vector in our dataset, R_i and R_o are the user's request vector (query), $Score1$ is the similarity between R_i and S_i and $Score2$ is the similarity between R_o and S_o.
Then the final $Score$ is calculated as the average of input and output similarity:

$$Score = \frac{Score1 + Score2}{2} \tag{3.5}$$

We calculate the similarity of Query(Request) Vector with all the vectors in our dataset. Thereafter we sort the results from the greatest to the weakest score and retrieve all the results which have a score greater than a given threshold. We have given the user an option using which the user can specify this threshold value.

Semantic Linear Search

The main idea is basically iterating over the index data-set of all the services and selecting the most similar service i.e., where the similarity of context of the search query given by the user and the context of the web service is maximum. We define similarity between two given context as :

$$\text{Semantic Similarity} = \frac{\sum sim(u_i, v_i)}{m \times n} \tag{3.6}$$

where, $u_i\epsilon$ User's Query Context$\forall$ i = 0, 1, . . . n, $v_i\epsilon$ Service's Indexed Context$\forall i = 0, 1, ...m$, $sim()$ denotes similarity between 2 given lemmas.

The semantic similarity of user's query and each of the web service description is computed following which the services are sorted according to maximum similarity.

3.5 Searching Composite Services

It is often the case that a single service in the database is unable to satisfy user's query complete. In such cases we need to find multiple services that can work together in a given sequence so as to provide the required functionality to the user. In essence the services have to be automatically composed. Service composition has following steps :

- Searching for suitable web services that can be composed together to act as a single service
- Arrange the different web services in a particular sequence that yields the desired output
- Conversion of data-formats so that output of one service matches the input format expected by the next service in the sequence

Several solutions to this service composition problem has been proposed based on graphical model of web services [9], [10]. In this section a methodology to search for multiple web services that can be composed together to serve to user's using a graph of interconnected web services has been discussed.

Constructing Service Interface Graph

A DAG (Directed Acyclic Graph) is constructed to model services and the relation between their interfaces. Each node in this graph represents a web service. A node has several incoming and outgoing edges. An edge from node 'A' to node 'B' signifies that the output yielded by service 'A' is similar to the input accepted by service 'B' i.e., service 'A' and 'B' can be composed together. To construct the graph we first compute the equivalent input and output vector of all the service in the database. Then we match the output of each service to the input of every other service i.e., determine the co-sine similarity between the output vector of first service and input vector of second service. If the similarity is found to greater than a pre-determined cut-off the the two services are connected via an edge from first one to the second. After addition of each edge, the graph is checked for cycles. If any cycles are found to exist in the graph, the recently added edge edge is discarded. The main objective to model the services such that service composition problem can be converted into a simple graph traversal problem. Thus, it is essential to have an acyclic graph. The similarity cut-off for adding edges is chosen to be 0.9. A high value is chosen to ensure that their is almost a perfect match between the interfaces, This cutoff can be determined dynamically based on the average similarity of interfaces and several other domain-dependent factors.

Executing Service Composition Query

Once the graph is constructed, it has to be traversed for each query. Firstly, an input and output vector corresponding to the user's query is computed. Then a keyword based query as described in previous section is executed. The resultant services are sorted according to the input similarity - and top results (top 10) results are filtered. This gives the top 10 start-point of potential composition. For each of these input, the graph is traversed using the well known DFS (Depth First Search) Algorithm starting from the input service as source. For every node visited, the similarity between the node's output vector and user's query output vector is determined. Among all the nodes visited, the node with best output vector is selected. The path between the corresponding source node and the node with best output yields the best composition possible for the corresponding source service.

Chapter 4

Implementation Details

This chapter outlines the implementation details of the system.

4.1 Development Environment

We implemented the Search Engine using python and the Web front-end using Python and PHP frameworks. We used NetBeans IDE 8.0.1 for the development of the User Interface and business logic layers of our application. This IDE is a good tool to create a fast Web-based implementation without worrying about circumferential problems like deployment or compatibility. We have also used Sublime text and PyCharm for creating python scripts and Chrome Browser v37.0 for testing and writing JavaScript. Memory requirements and Processor speed are negligible for our sample repository of 1076 OWL-S files. The test machine is a 2.50GHz Intel® Core^{TM} i5-3210M CPU with 4 GB of main memory. The operating system used is Debian based Linux - Ubuntu 14.04 LTS x86_64. We used a standard Apache2 Server for deployment of the search engine.

4.2 Sample Data-Set

We had a sample data set of 1076 OWLS files. However, owing to our system's limitation of working with foreign language we could only index 1070 of those 1076 files. Keywords for the vector generation was added manually. All these keywords were collected from these 1076 files itself and were classified into categories manually.

4.3 Work Done

Extracting Description Files

The initial motive of the project was to extract information from WSDL files. We found out few public UDDI repositories which contain a few hundred WSDL files. Following tools were used to download those WSDL files:

- **Python Scripts** - We wrote customized Python scripts for gathering the destination and meta-files of all those web services for each repositories. Python scripts were also used to parse the various URLs related to meta-files.

- **Mechanize** - It is a very useful python module for navigating through web pages. It has been used extensively in our project to navigate through the destination web services.

- **Beautiful Soup** - It is a Python package for parsing HTML and XML documents, is used to extract download links of WSDL files from the web pages of the repositories. Beautiful Soup creates a parse tree for parsed pages that can be used to extract data from HTML - a very useful tool for web scraping.

Challenges Faced - Several problems were encountered during this phase. These were as follows:

1. Too many duplicates and bad meta-files
2. UDDI repositories are obsolete. Major UDDI registry vendors like Microsoft, IBM, etc. are no longer providing this service.
3. Most of the Web Services files on public repositories refers to the local host (i.e., dummy services created by programmers to test).
4. The names of the WSDL were in different international languages which makes it difficult to write a generalized program.
5. Many Web Services do not exist although their description file exist.
6. Most of the servers do not allow cross-domain scripting which is essential for automating the process of collecting web service meta-files

Ontology based OWLS

As mentioned in the methodology section, the new method of describing Web Services is through OWLS files. Ontologies play an important role in semantic descriptions of Web Services.They are used to describe the semantics of terms in different domain. Our next motive was to build a search engine for web services by extracting information from the corresponding OWL-S files of Web Services. There are not many public OWL-S file repository available on the Internet. However, our Project Mentor helped us with a data-set of 1000+ OWLS documents. We have used the said dataset throughout the course of this project.

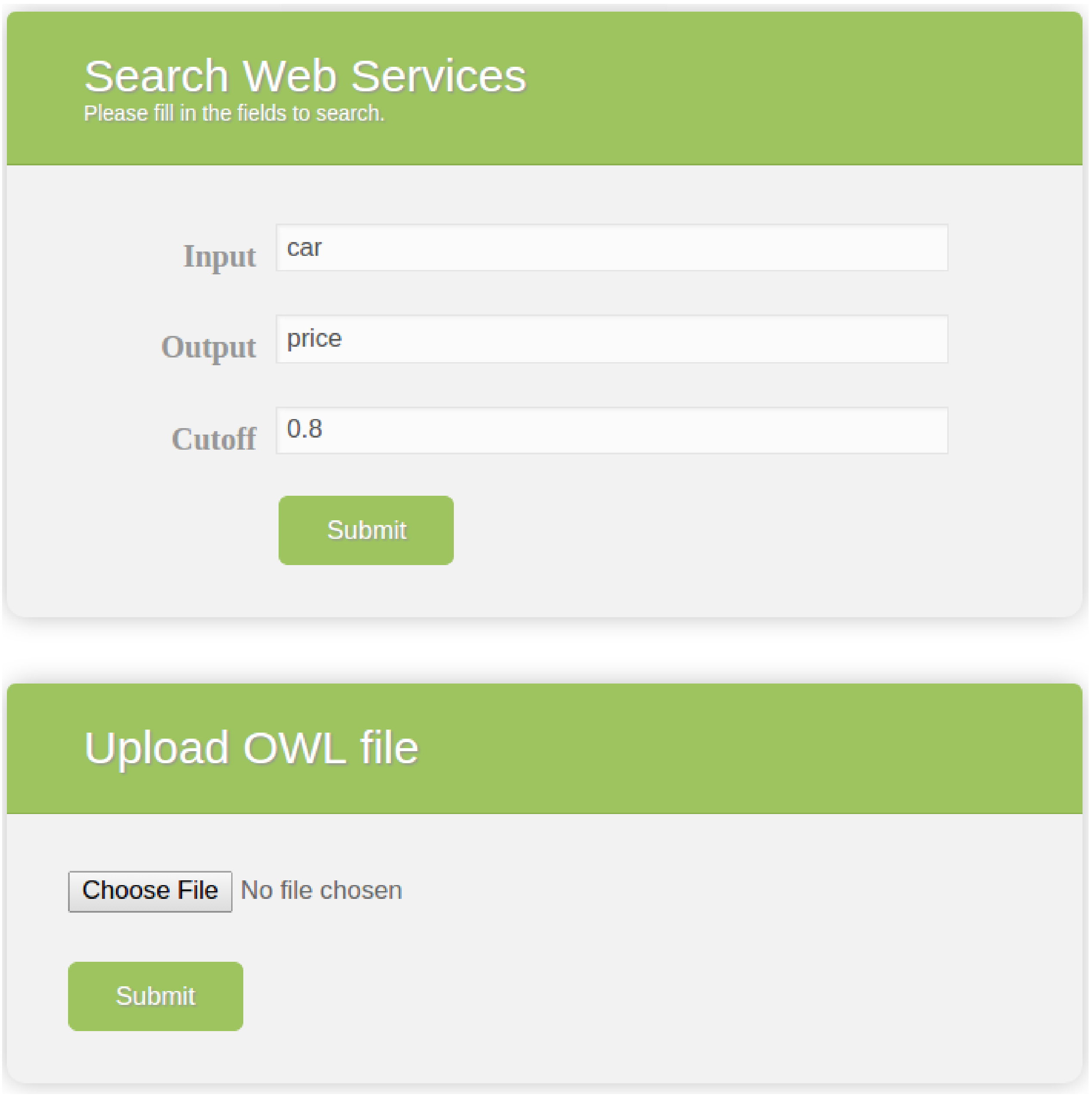
Prototype System

To demonstrate the reliability of our concept and to show how an application for the presented architecture may look like, we implemented a prototype search engine. We designed our application with a Web front-end to offer the possibility of using its functions and evaluate the produced results. We allowed the user to either enter input/output for a desired service or upload a corresponding OWL-S file. We search for web services similar to the uploaded OWLS files or services that have similar interfaces (input/output). A sample screen-shot of the search form has been illustrated in Figure 4.1

Keyword based Indexing of OWLS

For indexation purpose, we define 7 categories (or dimensions) of Web Services as discussed in the preceding chapters. Each Input or Output vector is represented in this 7-dimension vector space. We assign a list of keywords on each of the seven categories. For our prototype system, we have have done this manually. We have some assigned some common input keywords for each category and some common output keywords to each category. This has been illustrated in Table 4.2 and 4.1.

Following a step-wise process of what we have implemented for indexing web services :


Search Web Services
Please fill in the fields to search.

Input: car
Output: price
Cutoff: 0.8

Upload OWL file

Figure 4.1: **Screen-shot of Search Page**

1. **Document Parsing** - We implemented a OWLS Document Parser using Python. It extracted the required elements i.e <<**profile:hasinput**>> and <<**profile:hasoutput**>> from all the OWLS files in our repository using Beautiful Soup package.

2. **Algorithm for Constructing Input Vector** - After extraction we do the following for <<**profile:hasinput**>> element of each repository

```
Vi = [0, 0, 0 ,0, 0, 0, 0]
for concept in concept-list #concept list is the list of 7 concept
    for keyword in concept.input_keylist #list of keywords
        if keyword is substring of <<profile:hasinput>>
            Vi[concept.no] += 1
```

3. **Algorithm for Constructing Output Vector** - Similarly, we do the following for <<**profile:hasout**>> element of each repository

```
Vo = [0, 0, 0 ,0, 0, 0, 0]
```

Table 4.1: **Sample Input Keywords Used for each Categories**

Category	**Sample Input Keywords**
Communication	Address, Code, Record, ZIP, Linguistic, Map, Media, Source, Analog, Digital
Economy	Dollar, Euro, Expense, Credit, Liability, Retail, Profit, Sale
Education	Academic, Book, Author, Research, ISBN, Science, Reader, Radius
Food	Butter, Bread, Beverages, Apple, Biscuit, Meat, Lemon, Honey, Pea, Tea
Medical	Medical, Doctor, Hospital, Drug, Bed, Care, Visit, Clinic, Center
Travel	Car, Bicycle, Longitude, Activity, Hill, Beach, Hike, Hotel, Capital
Weapon	Ballistic, Destruct, Device, Missile, Projectile, Weapon, State, Power

Table 4.2: **Sample Output Keywords Used for each Categories**

Category	**Sample Output Keywords**
Communication	Address, Interval, East, Phone, Time, Zone, URL, Information, Motion, Post
Economy	Firm, Intentional, Tax, Duty, Cost, Price, Bank, Fund, Income, Country, Total, Sold
Education	Density, Scholarship, Article, Item, Liquid, Edit, Internal, Journal, Software
Food	Beer, Price, Grain, Whiskey, Diet, Cereal, Breakfast, Lunch, Taste, Dough, Cola, Flavor
Medical	Physical, Cost, Doctor, Test, Quality, Occupation, Specialty, Type, Bed, Internal
Travel	Direction, Distance, Latitude, Longitude, Code, City, Area, Land, Hotel, Driver
Weapon	Arrow, Price, Ship, Battle, Projectile, Weapon, State, Power, Technology

```
for concept in concept-list #concept list is the list of 7 concept
    for keyword in concept.output_keylist #list of keywords
        if keyword is substring of <<profile:hasoutput>>
            Vo[concept.no] += 1
```

4. **Implementing Vector Construction** - UNIX Shell Script was used for construction of vectors. Powerful UNIX commands like **grep** were handy and had low-overhead.

5. **Iterating through Dataset** - We wrote a python script to iterate through all the OWLS files in our dataset and apply all the above steps. After that we save the indexed

Web Services in a file using space separated format.

Indexing using Semantics Techniques

Word Net [11] was used to determine the synsets of the words used in the description documents of the services and to determine the similarity between various synsets. Word Net provides as easy to use interface for python programming language via nltk package. The main functions that are relevant for this project are **synsets**() and **jcn_similarity** () . Again, Python's Beautiful Soup was used to parse OWLS document and then extract the text description of the web service. After that each word in the test description is mapped to a set of it's lemmas using wordnet. Then the word's meaning is disambiguated using a simple algorithm :

$$\text{selected sense} = max \sum sim(s_j, sc_i) \tag{4.1}$$

where, s_j is the word to be disambiguated and sc_i is the set of already disambiguated words.

We sort the words according to the number of synset in their mapped list. If no word is disambiguated trivially (i.e., only one possible meaning), then the context is established used first two words - select the pair of synset which has maximum mutual matching. If only one word is given, the most frequently used lemma of that word is taken as the context of the service. Once all the words in the service's text description is disambiguated, the selected senses are stored in a file. All the synset chosen are of same POS (Parts of Speech) Tag.

Querying using Cosine Similarity

We obtain the desired service interface from the user. For the requested interface (input and output), we create a request vector R_i and R_o of concepts using the same vector construction algorithm described in the previous section. Cosine similarity is found between R_i, S_i and R_o, S_o using a simple Python script by iterating through all the vectors of concepts in our dataset. We filter the results above a certain threshold and store the result in a temporary file. Threshold value can be set by the user. But, threshold value greater than 0.7 is recommended for the retrieval of relevant services. A default value of 0.8 is used. Ranks are assigned to each of the sorted service description document.

Querying Using Semantic Approach

From the user's given query, a query context is established by disambiguating the words in the query similar to the indexing process using wordnet. Once a context is established, The list of all the indexed documents is traversed and the query context is matched with each. The similarity score is recorded and the OWLS documents are sorted accordingly. After that the service is sorted according to it's similarity score and a rank is assigned to each service. Once a rank is assigned the final list of services is computed by considering both the lists. Equal weightage is given to each rank (which can be tuned) and thus average is taken of both rank to compute the final rank.

Graph Construction based on Service Interface

An in-memory graph of services and their interface dependency is constructed. Adjacency List representation of graph has been used. Each Node contains following data:

- Service Name or Service Identifier
- Vector representation of service interface - input and output

- A list of nodes to which the current node has an outgoing edge

The graph is constructed as discussed in the previous chapter. Graph Construction takes place only once when the server starts for the first time. After that the graph resides in main memory and same graph is used for each query.

Composition Query

When a composition query is received - an iterative version of DFS is implemented to traverse the constructed graph. The graph is traversed multiple times - (10 times in our implementation) one for each possible source (first service in composition). A simple python list is used as a stack. Once the best output service is determined using DFS it is also essential to determine the path from the source node to the node with best output similarity. This is done by keeping a record of node hierarchy while traversing the graph. For the said purpose, we store the parent node (the node that discovered the child node during DFS traversal) for each node in a separate list. The parent of the source node is assigned to be the source node itself. Thus once the best output service is determined, find the composition is as simple as iterating over the list of parent node recursively until source is reached.

4.4 Results and Analysis

The results that we obtain are very encouraging. We are able to obtain very relevant web services given an interface. We have obtained this results with a very small set of keywords in each field (average 40 each). As such, with a large set of keywords in each category we should be able obtain a much better results.

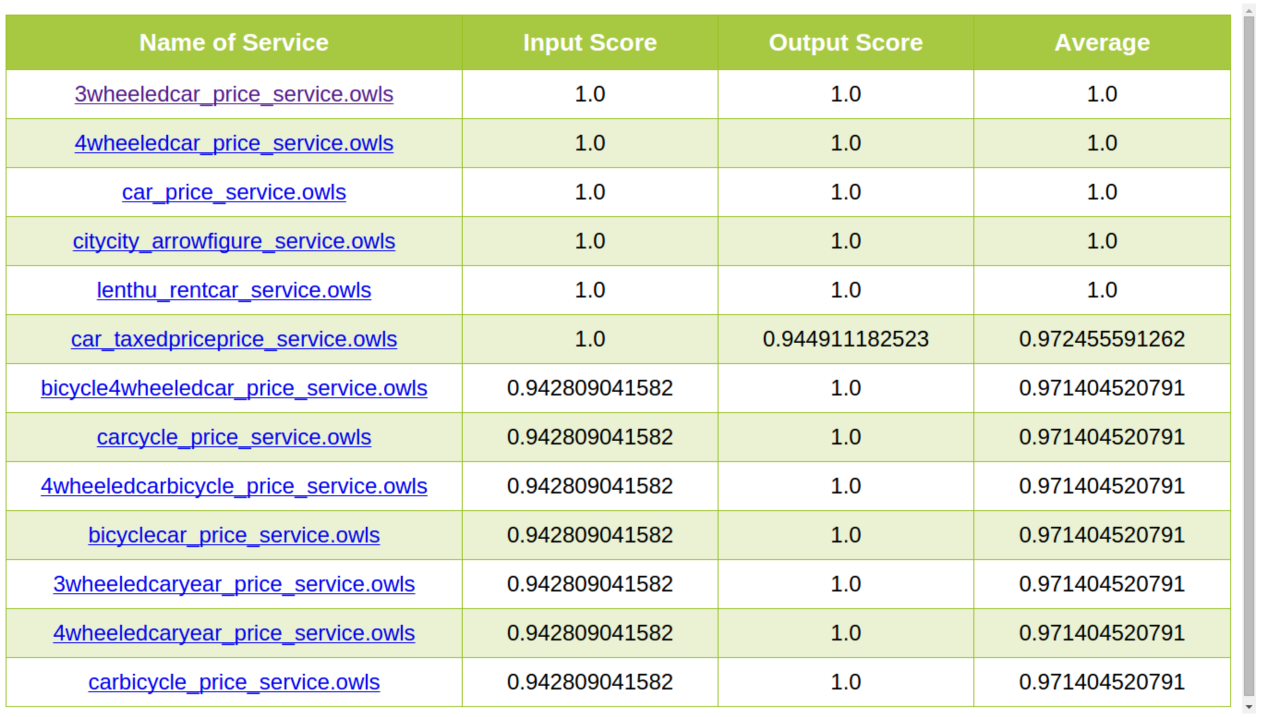
Sample Result

A sample result set is illustrated in Table 4.3.

Table 4.3: **Sample Result of some Queries**

Input	Output	Theta	Services and Similarity
Car	Price	0.95	3wheeledcar_price_service (1.0), car_price_service (1.0), citycity_arrowfigure_service (1.0), lenthu_rentcar_service (0.972)
Missile	Range	0.9	missile_lendingrange_service(0.971), missile_givingrange(0.933), ballistic_range_service(0.918)
Location	Distance	0.9	sightseeing_service(1.0), DistanceInMiles(0.908), calculate_betwee_Location(0.903), surfing_service(0.901)
Medical	Bed	0.5	hospital_investigatingaddress_service(0.789), medicalclinic_service(0.670), SeePatientMedicalRecords_service(0.640)

A screen shot of the sample result is also added. See Figure 4.2.



Name of Service	Input Score	Output Score	Average
3wheeledcar_price_service.owls	1.0	1.0	1.0
4wheeledcar_price_service.owls	1.0	1.0	1.0
car_price_service.owls	1.0	1.0	1.0
citycity_arrowfigure_service.owls	1.0	1.0	1.0
lenthu_rentcar_service.owls	1.0	1.0	1.0
car_taxedpriceprice_service.owls	1.0	0.944911182523	0.972455591262
bicycle4wheeledcar_price_service.owls	0.942809041582	1.0	0.971404520791
carcycle_price_service.owls	0.942809041582	1.0	0.971404520791
4wheeledcarbicycle_price_service.owls	0.942809041582	1.0	0.971404520791
bicyclecar_price_service.owls	0.942809041582	1.0	0.971404520791
3wheeledcaryear_price_service.owls	0.942809041582	1.0	0.971404520791
4wheeledcaryear_price_service.owls	0.942809041582	1.0	0.971404520791
carbicycle_price_service.owls	0.942809041582	1.0	0.971404520791

Figure 4.2: **Screen-shot of Result Page**

Analysis

The complexity analysis of the system for different events has been illustrated as follows:

- **Indexing a single OWLS document** - The steps followed while indexing a single OWLS document and their complexity are:

  1. Extract the input and output interface descriptions
  2. Compare the input keyword list and the output keyword list of all categories with the input and output interface description and create a vector from the comparison

  Step 1 is dependent on the length of the OWLS file and can be assumed to constant. Thus Step 1 takes constant time.

  Let the number of keywords in each category, on an average be C. Since there are 7 categories, the number of keywords in the data set is $7C$. Our algorithm will compare each keyword in our data set with the input and output description. Thus the time taken will be proportional to $7C$. Thus Step 2 has $\mathcal{O}(C)$ time complexity.

  Thus the whole process of indexing a single OWLS document takes $\mathcal{O}(C)$ time.

- **Indexing the whole OWLS dataset** - To index the whole repository of web services we need to index each and every OWLS document by applying the algorithm mentioned before. Let the number of OWLS documents that the system has in it's repository be N. Since the algorithm to index a single OWLS document takes $\mathcal{O}(C)$ time (As shown in the previously), the algorithm to index the whole repository will take $\mathcal{O}(NC)$ time complexity.

- **Constructing Composition Graph** : The vectors corresponding to service interfaces are already available from indexing. The input vector of each service is matched with output vector of every other service. Thus the complexity of graph construction is

$\mathcal{O}(N^2)$, where N is the number of service. The check for cycle formation is linear in time and thus it doesn't add anything to the complexity. However, the graph is constructed only once when the server is started and thus we can afford it to be slower. Whenever a new service is added to the database i.e., A new node is appended to the graph, it's input and output has to be matched with every other service and thus the complexity of adding new node will be $\mathcal{O}(N)$.

- **Complexity of searching** - To search for a similar web service, the user will supply the input and the output he/she expects. Our algorithm will work in the following way :

  1. Create a pair of vectors representing a pseudo OWLS file document the input and output supplied represents.
  2. Find the Cosine similarity of these vectors with the previously indexed values of all the OWLS documents in our repository and accept all those which are above a certain threshold value.
  3. Establish Context of the User's Query
  4. Find Context Similarity Score for each document.
  5. Sort the result and display it to the user.

  Step 1 is equivalent to indexing a single OWLS document and hence takes $\mathcal{O}(C)$ time complexity.

  Finding Cosine similarity between 2 OWLS documents takes constant time. In Step 2 we find the Cosine similarity of the pseudo OWLS document with each and every OWLS document. Let the number of OWLS documents in the system repository be N. Then Step 2 takes time of complexity $\mathcal{O}(N)$

  Step 3 again requires constant time to implement as the number of keyword will have a constant upper-bound.

  Step 4 It takes $\mathcal{O}(N)$ time as we have to traverse through the whole dataset

  Step 5 is simple sorting that takes $\mathcal{O}(K \log K)$ where K is the number of results after filtering using the given threshold. The maximum value of K can be N. Thus the time taken by this step at most $\mathcal{O}(N \log N)$.

  Thus the complexity of searching for a similar web service is $\mathcal{O}(N)+\mathcal{O}(C)+\mathcal{O}(N \log N)$.

Thus we can observe that the system is very efficient and scalable. It can easily handle large number of Web Services and still return result in a reasonable amount of time.

Chapter 5

Conclusion and Future Work

We have developed a basic search engine that uses only service interfaces. Our result looks promising. However, the current system still has a lot of scope for improvements which have been discussed in the following sections.

5.1 Automatic Keyword Retrieval

We manually indexed a set of keywords relevant to each category namely - economy, education, food, medical, travel, weapon and communication. However for a very large and diverse set of web services, it is not feasible to manually index appropriate keywords. One can make use of freely available repositories like Wordlon to get the keywords related to a specific category. Moreover the keyword list should be dynamic i.e., the system should automatically learn new keywords and update the list. This can be achieved using machine learning techniques. A part of our problem can be modeled as a well known classification problem in the area of Machine Learning.

5.2 Considering Other Parameters

Currently our system constructs the appropriate vectors using only the words specified in the interface (input / output) of the web service. Better results could be obtained if we consider other parameters like functions and meta-data of a web services.

5.3 Hybrid Approach with Graph Matching

It is our intuition that better results can be obtained using a graph based assignment and matching algorithms. However, the current system only uses co-sine similarity approach. Both methods can be implemented and best one or an average value of similarity from both methods can be considered. We can also specify weight-age to the output of each algorithm and tune the weight-age depending on the need of the user.

References

[1] "Web services architecture - w3c." `http://www.w3.org/TR/2002/WD-ws-arch-20021114/`. Accessed: 7th November, 2014.

[2] H. L. Truong and S. Dustdar, "A survey on context aware web service systems," *International Journal of Web Information Systems*, vol. 5, no. 1, pp. 5–31, 2009.

[3] Y. Peng and C. Wu, "Automatic semantic web service discovery based on assignment algorithm," in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 6, 2010.

[4] C. Platzer and S. Dustdar, "A vector space search engine for web services," *Third European Conference on Web Services*, 2005.

[5] A. Cuzzocrea and M. Fisichella, "Discovering semantic web services via advanced graph-based matching," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pp. 608–615, IEEE, 2011.

[6] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin, "Semantic web service discovery using natural language processing techniques," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4660 – 4671, 2013.

[7] H. Fethallah and A. Chikh, "Automated retrieval of semantic web services: a matching based on conceptual indexation.," *Int. Arab J. Inf. Technol.*, vol. 10, no. 1, pp. 61–66, 2013.

[8] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," *CoRR*, vol. cmp-lg/9709008, 1997.

[9] S.-C. Oh, B.-W. On, E. Larson, and D. Lee, "Bf*: Web services discovery and composition as graph search problem," in *e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on*, pp. 784–786, March 2005.

[10] S. Hashemian and F. Mavaddat, "A graph-based approach to web services composition," in *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*, pp. 183–189, Jan 2005.

[11] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.