A Linguistic Approach for Semantic Web Service Discovery

Jordy Sangers
307370js
jordysangers@hotmail.com

Bachelor Thesis

Economics and Informatics
Erasmus School of Economics
Erasmus University Rotterdam

Supervisor: Dr. Flavius Frasincar
Co-reader: Frederik Hogenboom MSc.

13th July 2009

Abstract

This thesis presents a Semantic Web Service Discovery framework for finding semantically annotated Web services by making use of natural language processing techniques. The framework allows searching through a set of annotated Web services in order to find a match with a user query, consisting of keywords and their associated part-of-speech tags. For matching keywords with semantic Web service descriptions given in WSMO, techniques like part-of-speech tagging, lemmatization, and word sense disambiguation are used. After determining the senses of relevant words gathered from Web service descriptions and the user query, a matching process takes place. The Semantic Web Service Discovery framework and its implementation helps end-users to search for specific Web services based on the context of their needs. By specifying a query using keywords, end-users do not need to have knowledge about semantic languages, which makes it easy to express the desired Web services.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the emergence of Web services and the Service Oriented Architecture (SOA), processes are more and more being decoupled. Combining these technologies will create a wide network of services that collaborate in order to implement complex tasks.

Currently, Web services are commonly described via narrative Web pages containing information about their operations in natural languages. These Web pages contain plain text with no machine interpretable structure and therefore they cannot be used by machines to automatically process the descriptive information about a Web service.

To promote the automation of service discovery [1] and composition [2], different semantic languages have been created that allow describing the functionality of services in a machine interpretable form, while original Web service descriptions contained only information about the data types and bindings as a description of a Web service functionality. Although such semantic languages provide the functionality of automatic service discovery, end users might also want to discover services themselves. The problem is, however, that they do not have the knowledge about semantic Web service languages. Therefore, a discovery mechanism must exist to enable semantic Web service discovery based on keywords written in natural language.

Section 1.1 describes the problem of discovering Semantic Web services using

a keyword based approach. Based on this analysis, several questions are formulated. Section 1.2 defines for each research question a methodology to solve the problem. Section 1.3 lists the structure of this thesis.

1.1 Problem and Goal

Semantic Web services promote the automation of service discovery and composition. Though, service composition should be driven by people who knows business processes and not by technicians or machines. Therefore end users must be able to discover and compose these Web services themselves. Because a Semantic Web service is described in technical complex semantic languages and are therefore not suitable for end user to understand, a discovery engine must be created to enable discovery of Semantic Web services via simple keywords.

Creating a semantic Web service discovery engine using a keyword based approach can be a complex task. Several questions that may arise when building such engine are:

1. How to extract the most relevant information from a semantic Web service description?
2. How to match keywords from the user input with textual information from a semantic Web service description?
3. How to rank the Web services in such a way that the most relevant Web services can be provided to the user?

1.2 Methodology

For each question formulated in the problem statement in Sect. 1.1 a objective can be defined:

1. An implementation of a semantic Web service parser and reader.
2. An implementation of a series of Natural Language Processing (NLP) techniques to match user input with Web service information.

3. A matching algorithm providing matching scores between the user input and the Web service descriptions.

To extract information from semantic Web service descriptions, it is needed to know exactly what, where and how they describe Web services. Therefore first a literature review about different Web service languages and in particular semantic Web service languages is done. After that a parser and reader for a specific semantic Web service language will be implemented to extract relevant information from semantic Web service descriptions.

After this, several NLP techniques will be studied and applied to provide a matching between user input consisting of keywords and pieces of information extracted from a semantic Web service description. The semantic lexicon WordNet will be used to search for related words using its extensive Web of words. This will provide a more flexible matching system as related words like synonyms will also be taken into account.

Finally, several different algorithms for matching the user input with information from a semantic Web service description will be defined and implemented. They will be tested to see if they suggest a list of Web services in such a way that the most relevant Web services are at the top of the list. The different algorithms will be analyzed and compared with each other to see which performs best.

1.3 Structure

This thesis consists of six chapters and is structured as follows. Chapter 2 provides an overview of different Web service description languages and the related work. Chapter 3 describes the Semantic Web Service Discovery (SWSD) framework for discovery of Web services based on keywords. This is done in four sections. Section 3.1 describes the architecture of the SWSD framework. Section 3.2 focuses on parsing and reading of semantic Web service descriptions. Section 3.3 describes how to use Word Sense Disambiguation (WSD) for finding the correct meaning of words. Section 3.4 describes three different algorithms

to match the user input with the semantic Web service information. Chapter 4 exposes the SWSD engine, which is an implementation of the SWSD framework. This chapter is also divided into four sections where the last three sections correspond to the last three sections of Chapter 3. Section 4.1 shows and describes the SWSD engine. Section 4.2 explains what and how the SWSD engine extracts information from Web Service Modeling Ontology (WSMO) files. Section 4.3 describes how WSD is implemented in the SWSD engine and Sect. 4.4 describes how the different matching algorithms can be applied to match user input with Web service descriptions. Chapter 5 evaluates the different matching algorithms that were described in Sect. 3.4. The final chapter, Chapter 6.1, describes the conclusions and future work.

Chapter 2

Related Work

This section provides a state-of-the-art in the semantic languages for service discovery. First, a comparison is made between different types of Web service description languages in Sect. 2.1. Since this thesis covers the discovery of semantic Web services, three semantic Web service description languages are described and compared with each other in Sect. 2.2. Finally, several other approaches for discovering Web services are covered in Sect. 2.3.

2.1 Web Service Description Languages

For describing Web services, a wide variety of languages can be used. These languages differ in models and formalisms used for describing Web services and which properties of a Web service they cover. This range of languages can go from a simple piece of plain text describing the Web service, to large and complex semantic descriptions of the Web service's behavior by means of ontologies. Figure 2.1 shows an overview of the most widely used languages for describing a Web service. Distinction among four description layers is made in order to demonstrate an increasing role of semantics in such languages. Each layer can contribute to the process of matching a service with a given query. Thus, it is necessary to be able to make use of the information encoded into each format and layer in order to establish the most possibly complete view on the service.

To make Web services usable by machines, their interfaces are commonly

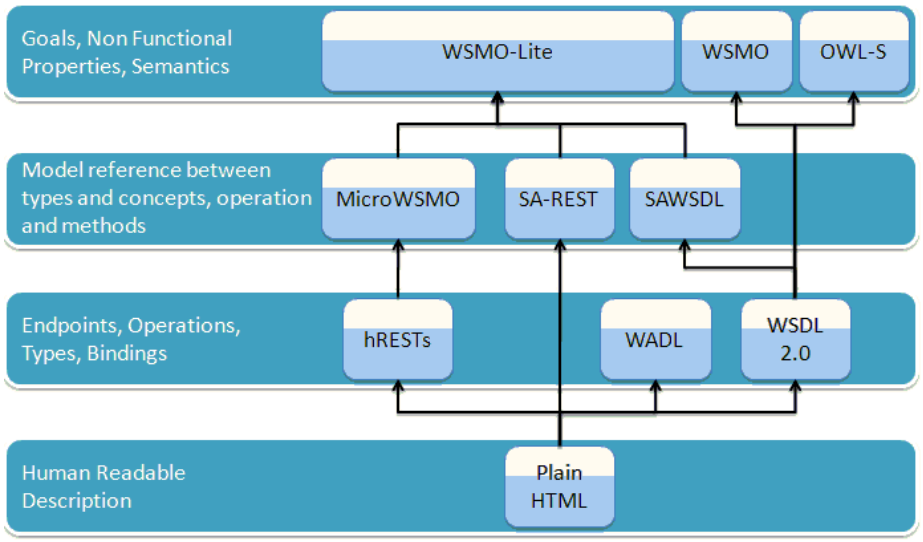

Figure 2.1: Web service description languages

defined in languages such as Web Service Description Layer (WSDL) [3], Web Application Description Layer (WADL) [4] or hRESTs [5]. These languages describe the bindings, the operations and their associated input and output data types, and the end points of a Web service. These descriptions enable humans and applications to understand where, when, and what a Web service is expecting from the user.

The top layer on Fig. 2.1 consists of semantic Web service description languages such as Web Service Modeling Ontology (WSMO) [6] and OWL-S [7]. These languages employs ontologies for describing the behavior of a Web service. Concepts, attributes and relations from existing ontologies and logical expressions can be used to state conditions and effects of a Web service.

To provide a bridge between the syntactical languages such as WSDL and hRESTs to the semantically enriched languages such as WSMO and OWL-S, middle layer languages were defined. MicroWSMO [5], SA-REST [8] and Semantic Annotations for WSDL (SAWSDL) [9] link the concepts from the semantic descriptions with the data types for the input and output of a Web service, or with its operations, and provide methods to transform data types to

semantic concepts and the other way around.

2.2 Semantic Web Service Description Languages

Language	OWL-S	WSMO	WSMO-Lite
Syntax	OWL	WSML	RDF/XML
Parts	Service Profile, Service Model, Service Grounding	Ontology, Webservice, Goal, Mediator	Ontology, Webservice
Reasoning	Logic language	Logic and rule language	Logic language

Table 2.1: Core differences between OWL-S, WSMO and WSMO-Lite

Table 2.1 lists the core differences between three semantic Web service description languages, OWL-S, WSMO and WSMO-Lite. They mainly differ in the syntax they have, the different parts they consist of, and which kind of reasoning they use to describe the logic behavior of a Web service.

OWL-S builds on Web Ontology Language (OWL) [10] and consists of different ontologies to describe a Web service. It is defined in order to enable three major tasks which could not be fulfilled with older technologies, e.g. WSDL: automatic Web service discovery, automatic Web service invocation, and automatic Web service composition and interoperation. Because OWL-S uses ontologies to describe Web services, Web services and their behavior become machine interpretable and thus tasks such as discovery and composition can be automated. OWL-S makes use of three different ontologies: a Service Profile, which states what the Web service does, a Service Model, which describe how the Web service performs the tasks, and a Service Grounding, which describes how to access the Web service.

WSMO is a framework for describing Web services and consists of four top-entities: Ontologies, Web services, Goals and Mediators. Ontologies provide the terminology used by other WSMO elements. Web services describe the capabilities, interfaces, and internal working of Web services. Goals represent

user desires, and Mediators provide bridges between different Ontologies, Web services or Goals to overcome interoperability problems. WSMO uses a specific designed language called WSML [11] and can contain powerful logical formulae to describe the different WSMO elements. It also contains of a grounding feature to reference concepts with WSDL data types so that automatic invocation can be achieved.

WSMO-Lite [12] was created because of the need for a simple semantic Web service description language. It is therefore a lightweight set of semantic service descriptions written in RDFS [13] that can be used for annotations of various WSDL elements using SAWSDL annotation mechanism. It only makes use of Ontologies and Web service descriptions and contains no grounding information, which makes it dependent of SAWSDL-like languages. The behavior of a Web service is only implicitly described by defining just preconditions and effects, so no specific how-questions can be answered.

2.3 Web Service Discovery Engines

Because the market for Web services is far from transparent, there are not many Web service search engines widely available. The ones that exist usually only search for Web services based on their UDDI registration and their WSDL description. One example is the search eSynaps [14] engine. Seekda! [15] tries to go further, by extracting semantics from the WSDL files, which enables runtime exchange of similar services and composition of services. Seekda! does not yet search through existing semantic Web service description files, but only makes use of the WSDL file of a Web service.

Service-Finder [16] is a platform for service discovery where information about services is gathered from different sources throughout a Web 2.0 environment. The information is automatically added to a semantic model so that flexible discovery of services can be realized.

GODO [17] is a Goal-Driven approach for searching WSMO Web services. It consists of a repository with WSMO Goals and lets users state their goal by

writing a sentence in plain English. A language analyzer will extract keywords from the user sentence and a WSMO Goal will be searched based on those keywords. The WSMO Goal with the highest match will be sent to WSMX [18], an execution environment for WSMO service discovery and composition. WSMX will then search for a WSMO Web service that is linked to the given WSMO Goal via some WSMO Mediators and return the WSMO Web service back to the user. This approach makes good use of the capabilities of the WSMO framework, but it cannot be applied for other semantic languages like OWL-S and WSMO-Lite, which do not have such goal representation elements.

Another approach for semantic Web service discovery is by searching for similarities among different service descriptions. Several mediation techniques to identify semantic similarities between ontologies, e.g., by using Mediation Spaces [19], have been developed. They mediate on data-level as well as semantic-level to discover related semantic Web services according to ontologies, other semantic Web services or WSMO Goals.

Chapter 3

Semantic Web Service Discovery Framework

The SWSD framework proposes a keyword-based discovery process for searching Web services which are described using a semantic language. This search mechanism incorporates natural language processing techniques to establish a match between a user search query, containing English keywords, and a semantic Web service description. It does not take into account any logic defined in the Web service descriptions yet, but uses the definitions of concepts stated in other imported ontologies. By making use of these definitions, the framework can establish a broader search field by also using related concepts from the ontologies to identify the context in which the Web service is operating.

This section will cover the global architecture of the SWSD framework (Sect. 3.1) as well as more in dept descriptions of its main components. Section 3.2 describes how the SWSD framework reads semantic Web service descriptions and which elements from those descriptions are being used for the discovery. How the context of a semantic Web service is established is described in Sect. 3.3. Finally, Sect. 3.4 describes three algorithms to match a user query with a semantic Web service using their contexts.

3.1 Framework Architecture

The SWSD framework assumes that there is a set of Web services described in semantic languages such as WSMO [6], WSMO-Lite [12] or OWL-S [7]. These annotations can be read by the system and words that might represent the context of the Web services will be extracted (e.g. the names of the operations or nouns and verbs stated in non-functional descriptions of concepts or conditions). These words must then be disambiguated, because words can have different senses. If the system knows the sense of the words, they can be matched with the senses disambiguated from the search query. This will result in a ranked list of Web services according to the match with the user search.

Figure 3.1 describes the architecture of the SWSD framework. The process consists of three major steps: Service Reading, Word Sense Disambiguation (WSD), and Match Making. Service Reading consists of parsing a semantic Web service description, extracting names and non-functional descriptions of used concepts. WSD determines the senses of a set of words. During Match Making the similarity between the different sets of senses is determined, which will subsequently be used for ranking the Web services.

3.2 Semantic Web Service Reader

To enable a search engine to look through Web service descriptions written in different languages, it has to have several different Web service description readers, one for each language. In the case of semantically described Web services, the readers must be able to parse a description and extract concepts, attributes and relations from WSMO, WSMO-Lite, OWL-S, etc... Therefore the first step in the process of searching for semantically described Web services is to implement readers for different languages and formats.

A semantic Web service reader must be able to extract various elements out of a Web service description and its used ontologies. In the case of a WSMO Web service, names and non-functional descriptions of elements such as the capabilities, conditions and effects of the Web service help in understanding the

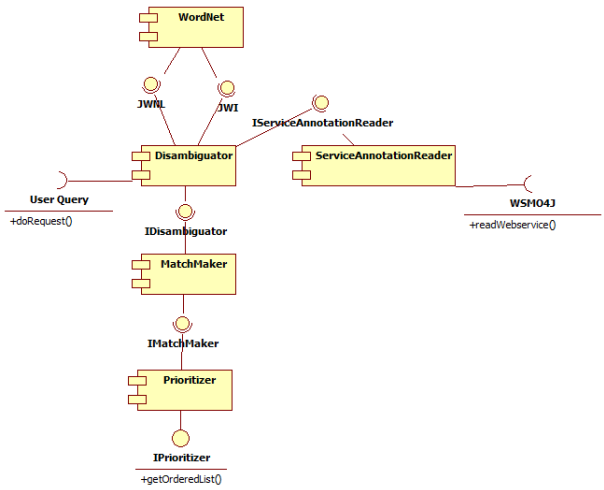

Figure 3.1: SWSD architecture

context of the Web service. Thereby, extracting concepts out of those elements and searching their non-functional descriptions in an ontology can result in a bigger change of establishing the right context. The non-functional descriptions are written in natural language and thus contain a human description of the specified element. By extracting these descriptions, one can thus establish the context of the Web services operations.

Before extracting words from a Web service description, the description has to be parsed. Different languages can mean different syntaxes and therefore different parsers are needed. For WSMO a WSML parser like WSMO4J [20] can help in this process and for OWL-S and WSMO-Lite, which are written using an OWL/RDF [10] language, a parser like Sesame [21] or Jena [22] can be used.

To find useful words for WSD, the element names and non-functional descriptions must be split into different words. In the case of element names, simply splitting the words when a case transition has occurred is enough, since in most cases they are written as camel words (e.g. HotelBookingWebService). Each word in the sentences found in the non-functional descriptions, must be tagged

with the right Part-of-Speech (POS) in order to find useful words. Using a POS-tagger, nouns and verbs from the sentences are extracted and used for the WSD.

3.3 Word Sense Disambiguation

A user can represent its goal by defining two different sets of words. One set contains only nouns and the other only verbs. By using the set of words gathered from the user input or from a semantic Web service description, WSD can be applied to establish the context of those words. This will result in a set of senses, each representing a single meaning of a word. Once a set of senses from the user query and a set of senses from a Web service are established, a matching between the two can be performed.

As non-supervised WSD allows disambiguation of words without user interference, we use a variant of the SSI algorithm [23] to get the senses out of a set of words (3.4). The algorithm disambiguates a word ($word$) based on a previously disambiguated set of words and their related senses. Per sense of the word (s_j), a similarity with the senses from the context (sc_i) is calculated and the sense with the highest similarity is chosen. After that, the word and its chosen sense will be added to the context (I) and another iteration will be done. This process continues until there are no ambiguous words left.

$$selectedSense(lex) = \arg \max_{s_j \in senses(word)} \sum_{sc_i \in I} sim(s_j, sc_i) \tag{3.1}$$

At the start of the process, a context is not yet established. In order to disambiguate meanings of the words that can have multiple senses, one first has to find the words that have only one sense (monosemous words) to initialize the context. If all the words in the set have multiple senses (polysemous words), the least ambiguous word is chosen and for each of its senses, the algorithm is simulated as if the sense was used as the starting context. Each time a new sense is added to the context, the similarity between the new sense and the context is stored. The sense which creates the highest sum of similarity measures during

its simulation is used for the context initialization.

Because studies have shown that the method of Jiang and Conrath [24] performs better than other Semantic distance measures [25], this method is chosen to compute the similarity between two senses. It uses the Information Content (IC), which depends on the probability of the occurrence of a particular concept in a large corpus.

$$IC(word) = \frac{1}{\log P(word)} \tag{3.2}$$

Formula 3.3 uses, beside the IC, also the Least Common Subsumer (LCS) between two concepts. This represents the first parent concept that subsumes both concepts when going in a bottom-up direction through a hypernym tree of words. The IC of the LCS is computed and compared with the IC of the two concepts. Then the similarity can be measured like this:

$$sim(s_i, s_j) = \frac{1}{IC(s_i) + IC(s_j) - 2 \times IC(LCS(s_i, s_j))} \tag{3.3}$$

3.4 Sense Matching

After disambiguating each word gathered from the user input or a semantic Web service description, one is left with several different sets of senses. The framework assumes each word in the user query being equally important for the matching process and therefore the user input will contain, after the WSD, one set of senses. However, a Web service description can contain words that represent the context of the Web service more than other words. Therefore, after the WSD, several sets of senses, each having a different weight for the matching process, are computed for a Web service.

This section will start with a high level view of the matching between the user input and the different levels of information extracted from a Web service description. Subsection 3.4.2 will describe how the Jaccard matcher performs the matching between the words or senses. Subsection 3.4.3 explains an approach for matching based on similarities between words or senses. Subsection 3.4.4

describes a matching approach that is similar to the Similarity matching, but applied in a slightly different way.

3.4.1 Level Matching

Besides matching only disambiguated senses, words that do not appear in the used lexicon should also be taking into account. These words can represent important names or concepts for the discovery of Web services and must therefore also be used in the matching process. So for matching user input with a semantic Web service description, the user input contains a set of words that could not be disambiguated (ws_u) and a set of senses (ss_u), and the Web service description contains multiple sets of words (mws_w) and multiple set of senses (mss_w). Because the Web service description, presented in the next chapter, provides a number (n) of sets containing words and senses, each having a different importance for the matching process, the final similarity between the user input and the Web service input will be a weighted average of the similarities between each set of words ($mws_{wi} \in mws_w$) and senses ($mss_{wi} \in mss_w$) from the Web service description and the set of words and senses from the user input (Formula 3.4). The weights (w_i) are established by means of experiments with different rates and must sum up to 1 in order to make sure that the final similarity between the user query and a Web service description has a range between 0 and 1.

$$finalSim(ss_u, mss_w, ws_u, mws_w) = \sum_{i=1}^{n} w_i \times levelSim(ss_u, mss_{wi}, ws_u, mws_{wi}) \tag{3.4}$$

For each set of words and senses from the Web service description, the system performs two different types of measures; one for the sense matching (Formula 3.8) and one for the matching of words that could not be disambiguated (Formula 3.11). These two measures will have a range between 0 (no match) to 1 (exact match) and will be combined into a single measure using a weighted ave-

rage (Formula 3.5). These weights are, as with the final similarity, established by means of experiments with different rates and must sum up to 1.

$$\begin{aligned} levelSim(ss_u, ss_w, ws_u, ws_w) = \\ w_{sense} \times senseSim(ss_u, ss_w) + \\ w_{word} \times wordSim(ws_u, ws_w) \end{aligned} \tag{3.5}$$

3.4.2 Jaccard Matching

For matching the user set of senses with a set of senses from one of the levels of a Web service description, the Jaccard matcher uses the Jaccard Index. This method is often used for computing the similarity between two sets and can so compare the different sets of senses:

$$senseSim(ss_u, ss_w) = \frac{|ss_u \bigcap ss_w|}{|ss_u \bigcup ss_w|} \tag{3.6}$$

By dividing the number of senses which appear in both sets by the total number of senses in both sets, a similarity coefficient can be calculated. With this approach the Jaccard matcher calculates the percentage of exact matching items and can also be applied for matching the words that could not be disambiguated:

$$wordSim(ws_u, ws_w) = \frac{|ws_u \bigcap ws_w|}{|ws_u \bigcup ws_w|} \tag{3.7}$$

3.4.3 Similarity Matching

To overcome the fact that for calculating similarity values only perfect matching items are used, the similarity matcher uses a similarity based approach for matching different sets of senses or non-disambiguated words. Using this approach, words that are almost identical are not considered to be a mismatch, but an almost match. The same applies for sense matching. If two senses are closely related, a value near to 1 will be given for their similarity. This allows more flexible matching between different items.

For calculating a similarity between the different sets of senses, the same similarity function as in WSD is applied. Formula 3.8 describes how the similarity between the user set of senses (ss_u) and a Web service set of senses (ss_w) is computed. The average of the similarity between each sense (s_u) from the user set of senses, and the Web service set of senses is computed. The average of the similarity between each sense (s_w) from the Web service set of senses, and the user set of senses is added to that to provide a symmetric match.

$$senseSim(ss_u, ss_w) = \sum_{s_u \in ss_u} \frac{senseScore(s_u, ss_w)}{|ss_u| + |ss_w|} + \sum_{s_w \in ss_w} \frac{senseScore(s_w, ss_u)}{|ss_u| + |ss_w|} \tag{3.8}$$

The similarity between a sense (s_a) and a set of senses (ss_b) is done by searching for the maximum similarity between the sense and one of the senses (s_b) from the other set. Formula 3.9 shows how this is done.

$$senseScore(s_a, ss_b) = \arg \max_{s_b \in ss_b} senseNorm(s_a, s_b) \tag{3.9}$$

Because the similarity formula (Formula 3.3) can give any value between 0 and infinity as a result and a range between 0 and 1 is preferred for quantifying the degree of match, a logarithmic function must be used to transform the values of the similarity. Using Formula 3.10, exact similar senses will have 1 as resulting similarity and a total mismatch between senses will result in 0:

$$senseNorm(s_a, s_b) = 1 - e^{-sim(s_a, s_b)} \tag{3.10}$$

For matching the sets of non-disambiguated words, the Levenshtein Distance metric is used. This metric calculates the total number of operations that needs to be done in order to transform one word to another. The similarity between two sets of words is done in the same way as when comparing two sets of senses. The only difference is that instead of the similarity function from WSD, now the Levenshtein Distance is applied.

The similarity function for calculating the similarity between the user set of words (ws_u) and a Web service set of words (ws_w) is described in Formula 3.11. Formula 3.12 describes how the similarity between a word and a set of words is computed. Finally, Formula 3.13 describes how the Levenshtein Distance is used for comparing two words, where ($maxLength$) is the number of tokens of the longest word that is being compared. If this formula returns a negative value, which means that too many changes had to be done to change one word into another word, a value of 0 will be used to indicate a total mismatch.

$$wordSim(ws_u, ws_w) = \sum_{w_u \in ws_u} \frac{wordScore(w_u, ws_w)}{|ws_u| + |ws_w|} + \sum_{w_w \in ws_w} \frac{wordScore(w_w, ws_u)}{|ws_u| + |ws_w|} \quad (3.11)$$

$$wordScore(w_a, ws_b) = \arg \max_{w_b \in ws_b} wordNorm(w_a, w_b) \quad (3.12)$$

$$wordNorm(w_a, w_b) = 1 - 2 \times levenshtein(w_a, w_b)/maxLength(w_a, w_b) \quad (3.13)$$

3.4.4 Average-Maximum Similarity Matching

When comparing a small set with a very large set, the similarity matcher presented in the previous section, can result in a very low similarity value if the large set subsumes the small set. This is caused by the fact that many items from the large set are not represented in the small set, so that there similarity values will be low. It could be better to use the highest similarity from one of the two sets instead. This results in the case if one set is subsumed by the other in a high similarity for the small set and a lower similarity for the large set. The high similarity from the small set will then be used to represent the similarity between the two sets.

Formula 3.14 shows how the Average-Maximum Similarity matching algorithm calculates the similarity between two sets of senses. The highest similarity

from either the user set of senses (ss_u) or the Web service set of senses (ss_w) will be used to represent the similarity.

$$
\begin{aligned}
&senseSim(ss_a, ss_b) = \\
&\quad \max\Big(\sum_{s_u \in ss_u} \frac{senseScore(s_u, ss_w)}{|ss_u|}, \\
&\quad \sum_{s_w \in ss_w} \frac{senseScore(s_w, ss_u)}{|ss_w|} \Big)
\end{aligned} \tag{3.14}
$$

The Average-Maximum Similarity formula can also be applied to the similarity calculation between two sets of words. The formula will then look like in Formula 3.15.

$$
\begin{aligned}
&wordSim(ws_a, ws_b) = \\
&\quad \max\Big(\sum_{w_u \in ws_u} \frac{wordScore(w_u, ws_w)}{|ws_u|}, \\
&\quad \sum_{w_w \in ws_w} \frac{wordScore(w_w, ws_u)}{|ws_w|} \Big)
\end{aligned} \tag{3.15}
$$

Chapter 4

Semantic Web Service Discovery Engine

This section describes the SWSD engine, which is an implementation of the SWSD approach and allows users to search for Web services on an existing repository by defining a set of keywords. The steps required for this implementation are closely related to the steps stated for the SWSD framework stated in the previous section. However, a possible implementation of the framework is presented, using specific languages and external libraries to provide a discovery engine.

Section 4.1 introduces the SWSD engine as an implementation of the SWSD framework. Section 4.2 describes how the SWSD engine reads WSMO descriptions and which elements from those descriptions are being used for the discovery. How the context of a semantic Web service is established is described in Sect. 4.3. Finally, Sect. 4.4 describes how matching algorithms can be used for matching user input with semantic Web service information.

4.1 SWSD Engine

At the moment, the SWSD engine can only apply a search on Web services which are annotated using the WSMO [6] framework as WSMO is a powerful framework and can contain very wide information about Web services. So for

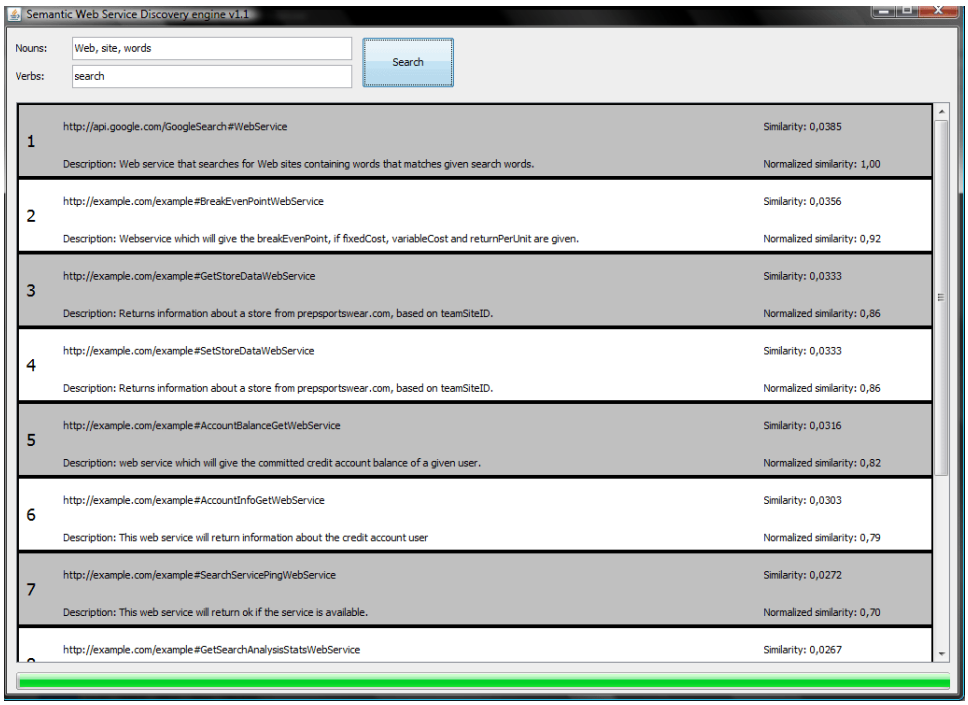

Figure 4.1: Result presentation example

an initial implementation only a WSMO-Web service reader and a WSMO-Ontology reader have been implemented, but based on the modularity of the implementation, the engine can be extended with readers that can parse other Semantic Web service languages.

To read the WSMO-files WSMO4J [20] is used. WSMO4J is an API and a reference implementation for building Semantic Web Services and Semantic Business Process applications based on WSMO. By using WSMO4J, WSMO files can be parsed, read and written. For the word sense disambiguation WordNet [26] is used through two WordNet API's, to find senses belonging to words. JWordNetSim [27] is used to calculate the similarity between two WordNet senses because it contains an implementation of the Jiang and Conrath formula, which we proposed in the SWSD framework. For the part-of-speech tagging the Stanford parser [28] is used. The overall implementation is made in Java, due

to the availability of external packages written for WSD and WSMO parsing and reading.

Figure 4.1 shows the user interface of the SWSD engine. The user can fill in a comma-separated list of nouns and verbs representing its goal. The system will propose a list of Web services, which are ranked by their similarity with the user input. Each item in the list contains the name of the Web service, its non-functional description, its similarity score and a normalized similarity score that has a range from 0 to 1 and is based on the Web service with the highest similarity score.

4.2 Semantic Web Service Reader

For reading WSMO files, an implementation of the WSMO4J API is used, resulting in an engine that can extract different types of information from WSMO Web services and WSMO ontologies. Because WSMO Web services and ontologies use different structures, two different readers must be used.

Figure 4.2 shows an example WSMO-Web service describing the Google Search Web service. The parts surrounded by rectangles are extracted by the system to establish the context of the Web service. First the name and non-functional description of the Web service are analyzed for extracting relevant words (nouns/verbs). Then the words from the non-functional descriptions of the properties of the capabilities are extracted and the reader will search in the logical formulae, written after each definedBy statement, for concepts used from external ontologies, which are stated after the importsOntology statement.

After reading the WSMO Web service file, the found concepts must be used to search in ontologies to find their description. Figure 4.3 shows an example WSMO Ontology containing some concepts used in the Web service description from Fig. 4.2. Based on a full identifier, the reader can search for a concept. If a concept is found, the non-functional definition, attributes and related concepts can be used for WSD.

In total the engine creates seven different levels of information about the

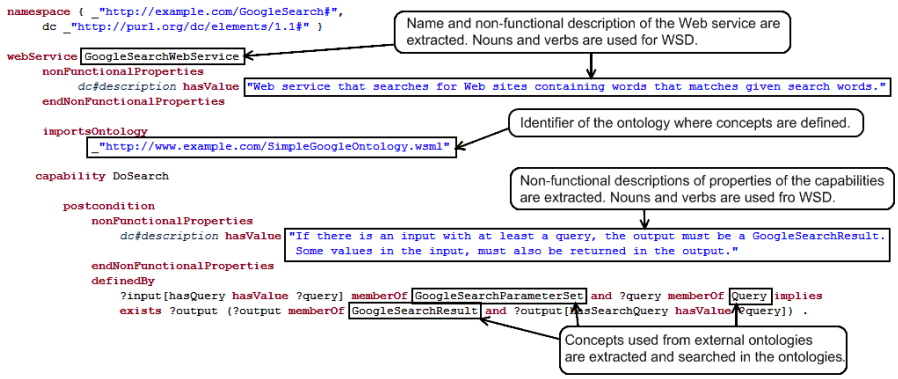

Figure 4.2: A WSMO Web service information extraction example

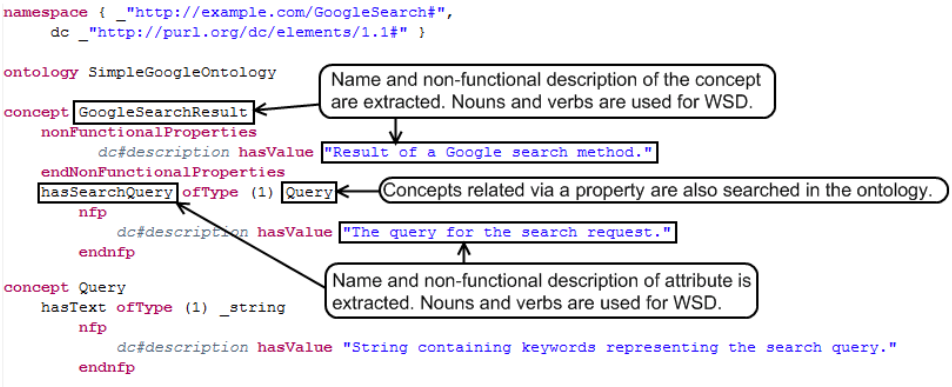

Figure 4.3: A WSMO ontology information extraction example

Web service. Each of these levels has a different amount of importance for the matching process. Those amounts are expressed in weights, summing up to 1. The different levels and their associated weights are:

- Non-functional description and name of the Web service, 7/27;
- Non-functional descriptions of properties of capabilities of the Web service, 4/27;
- Non-functional descriptions and names of concepts used from ontologies, 5/27;

- Non-functional descriptions and names of attributes of concepts used by the Web service, 1/27;
- Non-functional descriptions and names of concepts from ontologies directly related via attributes of concepts used by the Web service, 3/27;
- Non-functional descriptions and names of superconcepts of the concepts used by the Web service, 4/27;
- Non-functional descriptions and names of subconcepts of the concepts used by the Web service, 3/27;

The names and non-functional descriptions of the entities returned from these two readers will then go through a natural language processing step. Nouns and verbs are extracted from the non-functional descriptions using the Stanford POS-tagger and words are split if they consist of case-transitions.

A sentence like the non-functional description of the Google Search Web service presented in Fig. 4.2 (*"Web service that searches for Web sites containing words that matches given search words."*) will generate a set of nouns {*Web, service, sites, words*} and verbs {*searches, containing, matches, given*}. The name of the Google Search Web service, *GoogleSearchWebService*, will be split into the set of nouns {*Google, Web, Service*} and verbs {*Search*}. So instead of using compound words and whole sentences, the system only uses nouns and verbs extracted from them.

4.3 Word Sense Disambiguation

For WSD the system makes use of two different API's of WordNet. By using them, words can be found from within WordNet by a lexical representation of the word and its POS. These words will have different senses and each sense has its own identifier, a WordNet synset. The system has to find the synsets based on a set of lexical representations of words employed by users or a Web service description.

For establishing a starting context, WordNet will be used to find monosemous words. If no monosemous word is found, the word with the least synsets will be used to simulate the best starting context. For each synset, the similarity of the other words will be computed as if this synset was the starting context. Each time a new synset is added to the context, the similarity between the new synset and the context is stored. The synset which creates the highest sum of similarity measures during its simulation is used as the real starting context.

If there is a context, the SSI algorithm will be used to find the synsets of the other words. The similarity is measured using a native implementation of the Jiang and Conrath method. It can handle pairs of WordNet synsets as input and will return the similarity between them. For each word, the synset with the highest similarity to the context, will be added to the context.

4.4 Sense Matching

The synsets resulting from the disambiguation process must be matched in order to get a final similarity measure. Because the information in a Web service description has different levels of usability in the matching process, several sets of synsets belonging to a Web service will come out after the disambiguation phase. Thus one set of synsets coming from the user input must be matched with several sets of synsets coming from a Web service.

Each set from the Web service will have a weight representing the value of its information for the matching process. For example, the synsets found from the non-functional description of the Web service are more important during the matching, then the names of related concepts. These weights are calculated using experiments and must sum up to 1 in order to get a final match with the range $[0 \ldots 1]$.

To overcome the fact that words that are not present in WordNet are not being used in the matching process, every lexical representation of the extracted words from the user input, which are not in WordNet, is being compared to the lexical representations of the extracted words from a Web service description.

To provide a flexible matching, the words that are not in WordNet are first being lemmatized before they are matched. This means that for example the word *searching* will also match the word *searched*.

By combining the synset similarity value and the lexical representation similarity, by using a weighted average, we determine a final similarity value. Based on experiments the weights are determined and set to 7/10 for the synset similarity and 3/10 for the lexical representation similarity. The value of the synset similarity is higher than the lexical representation similarity as they provide more information.

Chapter 5

Evaluation

This chapter covers the evaluation of different matching algorithms that can be used for semantic Web service discovery. The algorithms described in Sect. 3.4 are implemented in the SWSD engine and evaluated using a set of predefined queries and sets of preferred Web services related to one of the queries.

Section 5.1 explains how the testing is done. In Sect. 5.2 the results of those tests are presented. Section 5.3 concludes which algorithms provides the best matching for discovery of semantic Web services.

5.1 Introduction

For testing the algorithms provided in Sect. 3.4, 61 test queries have been defined to check the outcomes of each of the algorithms. These queries represent possible sets of keywords a user might use to search for a Web services. For each query, a list of preferred Web services that are present in the repository of the SWSD engine, that should be returned as high as possible in the list of Web services provided to the user, are defined. Each of these Web services have a significant relation with the query and is therefore preferred to be visible for the user as early as possible.

Beside the algorithms described in Sect. 3.4, a simple matching algorithm that makes no use of natural language processing is added for the evaluation. The algorithm simply uses the Jaccard algorithm, but only for the lexical re-

presentations of the keywords.

The testing is done with usage of precision and recall metrics [29]. For every query, the precision and recall values for each of the algorithms are computed according to the list of Web services they provide using that particularly query as an input. This list is a ranked list, based on the similarity values calculated by the used matching algorithm, of all the Web services that are in the repository and will be compared with the list of preferred Web services stated for the query.

The precision and recall values are computed by traversing the provided list of Web services, according to a query and matching algorithm, from top to bottom. If a Web service is stated as preferred, then the Web service is classified (y_j) as 1. If it is not preferred, the Web service is classified as 0. Using this classification, the precision (p_i) and recall (r_i) for a position (i) in the list of provided Web services can be calculated as follows, where (k) is the number of preferred Web services:

$$p_i = \frac{\sum_{j=1}^{i} y_j}{i} \tag{5.1}$$

$$r_i = \frac{\sum_{j=1}^{i} y_j}{k} \tag{5.2}$$

Figure 5.1 shows how the precision and recall values are calculated for a given list of classified Web services. For each preferred Web service, which is classified as 1 and visualized with a blue dot, the precision and recall values are calculated. Each time a Web service that is not stated as preferred (classified as 0 and visualized with a yellow dot) is found, the precision will drop. This creates the fact that the PR-graph can visualize when it takes a long time before another preferred Web service is found in the list. This is the case when the precision drops heavily.

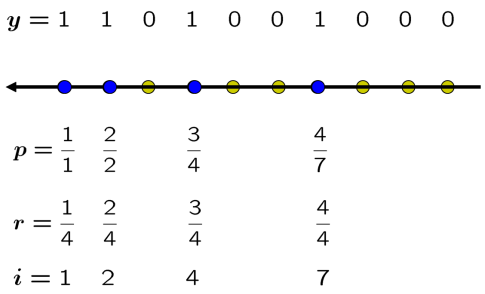

Figure 5.1: Precision and recall calculation example

5.2 Experiment Results

To test the performances of the four matching algorithms, 61 tests have been done. The tests can be divided into two types. 33 tests have been done to measure the matching performance of the algorithms using queries that have been designed to search for Web services that are present in the repository. 28 tests have been done to measure the performance using queries that have been designed to search for Web services that are not present in the repository. In the latter case, a number of similar Web services from the repository have been used to test how good the algorithms can discover similar Web services.

Testing with 61 queries and four matching algorithms, provide 244 PR-graphs. Because comparing so many graphs is impossible, PR-graphs consisting of average precision values for the recall points are created. This enables comparing all the different algorithms at once. However, the testing is done with lists of preferred Web services that can vary in the number of Web services they consists of. For testing, lists that contain two to four preferred Web services have been used. Because these variations in number of Web services cause different recall values, average precision values could only be calculated for queries that have the same amount of preferred Web services.

In the end 8 different PR-graphs can visualize the performances of the different matching algorithms. The four PR-graphs that are shown in Fig. 5.2, show the average results for the exact matching tests. For each of the four dif-

ferent numbers of preferred returned Web services (n) a PR-graph is created. The four PR-graphs that are shown in Fig. 5.3, show the average results for the approximate matching tests.

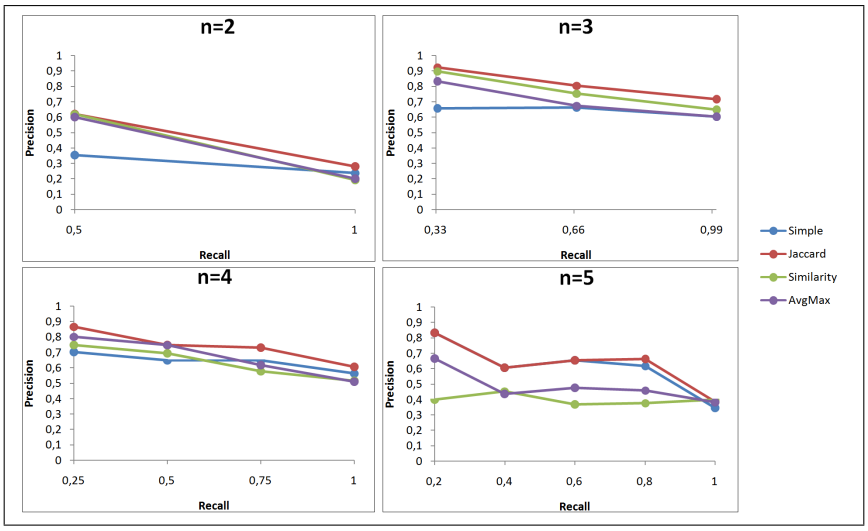

Figure 5.2: PR-Graphs for discovery of exact matching services

From the different PR-graphs that are shown in Fig. 5.2, we can make two observations. First, the Jaccard algorithm has in most cases a higher precision for the first half of the graph. Second, all algorithms have about the same precision to provide a full recall. This means that to provide all the preferred Web services to the user, they need about the same amount of Web services to be displayed. So, the user has to scroll down an amount of services, that is the same for every algorithm, to find the last preferred service. However, according to the fact that the Jaccard algorithm provides a higher precision for a lower recall, the Jaccard algorithm provides at least some of the preferred Web services in an earlier stage to the user then the others. It can therefore be seen as the best algorithm to discover exact matching Web services.

From the different PR-graphs that are shown in Fig. 5.3, we can make the observation that the similarity algorithms perform overall better for discovery of similar Web services than the Jaccard and the simple matching algorithm, as

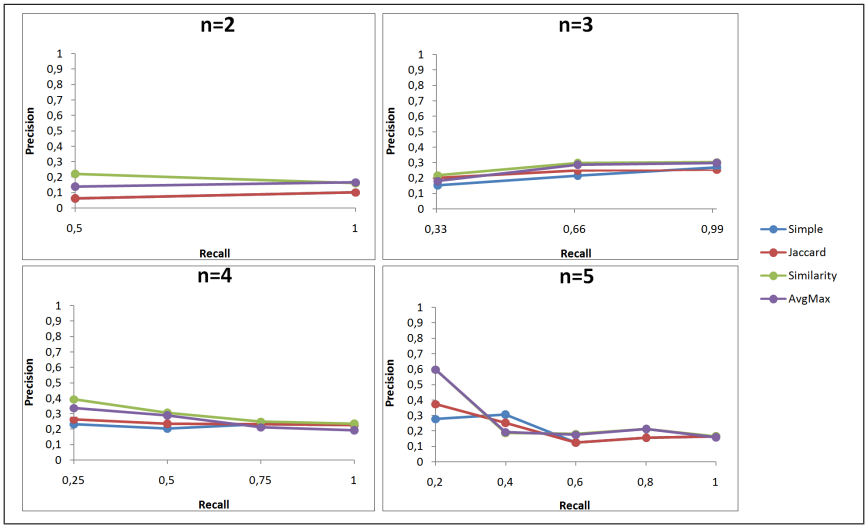

Figure 5.3: PR-Graphs for discovery of similar services

in most of the cases the precision lines of the Similarity algorithm and Average-Maximum Similarity algorithm are above the line of the Jaccard algorithm. The Similarity algorithm seems to be better than the Average-Maximum Similarity algorithm according to the discovery of similar Web services, though their performances are very close to each other.

5.3 Conclusion

To test the performance of the three matching algorithms explained in Sect. 3.4, 61 tests have been done. 31 tests were done to measure the performance of the algorithms according to discovery of exact matching Web services. 28 tests were done to measure the performance of the algorithms according to discovery of similar Web services.

Each test consists of a set of keywords used as a search query and a set of Web services that are preferred to be provided to the user as early as possible. To check how early the Web services are provided to the user, PR-graphs have been created. Out of these graphs can be concluded that the Jaccard matching

algorithm is the best method for discovery of exact matching Web services and that the Similarity matching algorithms is the best method for discovery of similar Web services

Chapter 6

Conclusions and Future Work

With the emergence of semantic Web services, new ways for Web service discovery have to be created. The Semantic Web Service Discovery framework is a framework for searching semantically annotated Web services based on keywords. With this approach, end users that do not have the knowledge of semantic languages can still discover semantic Web services. Section 6.1 presents the conclusions of this thesis. Section 6.2 describes future work that could be done.

6.1 Conclusions

In Chapter 1 several questions were posed that may arise when building a semantic Web service discovery engine. These questions can now be answered using the techniques that are used for the SWSD framework and engine.

1. **How to extract the most relevant information from a semantic Web service description?**
   As described in Sect. 3.2 several different parsers exists for parsing WSMO or OWL-S documents. In Sect. 4.2 it is stated that the SWSD engine uses WSMO4J to read WSMO descriptions. The SWSD engine extracts information such as names of elements and non-functional descriptions

that characterize the Web service using natural language.

2. **How to match keywords from the user input with textual information from a semantic Web service description?**
   For matching keywords with textual information extracted form a semantic Web service, different natural language processing techniques are used. This is described in Sect. 3.3 and Sect. 4.3. Techniques like lemmatization, part-of-speech tagging and word sense disambiguation are used to build up several sets of senses or lexical representations of words. These can then be matched with each other.

3. **How to rank the Web services in such a way that the most relevant Web services can be provided to the user?**
   In Sect. 3.4 different algorithms are proposed to match different sets of senses and lexical representations of words. These can be used for describing the similarity between the user input and a semantic Web service description. Chapter 5 described how these algorithms are tested and concludes that the Jaccard algorithm, described in Subsect. 3.4.2, can be used best for exact matching and that the Similarity algorithm, described in Subsect. 3.4.3, can be used best for approximate matching.

The SWSD engine is an implementation of the SWSD framework and can discover semantic Web services stored in a repository based on a set of keywords the user provides as a search query. This enables end users to simply describe the service they are looking for and to discover them. The contributions to the field of Web services and the Semantic Web are:

- A description of a framework for keyword based discovery of semantic Web services;
- Different matching algorithms evaluated for matching a user query with information from Web services;
- Natural language processing techniques that improve the performance of the discovery;

6.2 Future Work

As a future work, the SWSD framework and the SWSD engine can be extended in several ways. First, the SWSD engine could be extended with the ability to search for goals in WSMO annotation. This allows discovery of goals that can be used for finding Web services using the WSMO Mediators that link the goals with Web services. Besides that, the goals can also be used within WSMX to find related Web services. This is the approach that GODO [17] applies.

Second, the SWSD engine could be extended in such a way that it has the ability to read more annotation formats, e.g., WSMO-Lite. In this case, not only WSMO Web services can be discovered but also Web services described using other semantic languages than WSMO.

Finally, the weights that are used by the matching algorithms within the SWSD engine could be trained by making use of Artificial Intelligence. Currently, the weights are established by means of experiments, but they can improve the Web service discovery if they are trained using methods such as neural networks or Bayesian networks.

References

[1] Keller, U., Lara, R., Lause, H., Polleres, A., Predoiu, L., Toma, I.: Semantic Web Service Discovery. WSMX Working Draft - October 3, 2005 (2005)

[2] Hikimpour, F., Sell, D., Cabral, L., Domingue, J., Motta, E.: Semantic Web Service Composition in IRS-III: The Structured Approach. In: Seventh IEEE International Conference on E-Commerce Technology, IEEE Computer Society (2005) 484–487

[3] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). W3C Note 15 March 2001 (2001)

[4] Hadley, M.J.: Web Application Description Language (WADL) (2006)

[5] Kopecky, J., Vitvar, T., Fensel, D., Gomadam, K.: hRESTS and MicroWSMO (2009)

[6] de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., Konig-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman, D., Scicluna, J., Stollberg, M.: Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June 2005 (2005)

[7] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S. W3C Member Submission 22 November 2004 (2004)

[8] Lathem, J., Gomadam, K., Sheth, A.P.: SA-REST and (S)mashups: Adding Semantics to RESTful Services. In: International Conference on Semantic Computing, IEEE Computer Society (2007) 469–476

[9] Farrell, J., Lausen, H.: Semantic Annotations for WSDL and XML Schema. W3C Recommendation 28 August 2007 (2007)

[10] McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language. W3C Recommendation 10 February 2004 (2004)

[11] de Bruijn, J.: The WSML Specification. WSML Working Draft 2008-08-08 (2008)

[12] Vitvar, T., Kopecky, J., Fensel, D.: WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. In: 5th IEEE European Conference on Web Services, IEEE Computer Society (2007) 77–86

[13] Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004 (2004)

[14] eSynaps: eSynaps Web Service Search (2009)

[15] Semantic Technology Institute: Seekda! (2009)

[16] Cefriel, Seekda, Ontoprise, University of Sheffield: Service-finder (2009)

[17] Gomez, J.M., Rico, M., Garcia-Sanchez, F., Bejar, R.M., Bussler, C.: GODO: Goal driven orchestration for Semantic Web Services. In: Workshop on Web Services Modeling Ontology Implementations. Volume 113., CEUR Workshop Proceedings (2004)

[18] DERI Galway: Web Service Execution Environment (2008)

[19] Dietze, S., Gugliotta, A., Domingue, J.: Exploiting metrics for similarity-based semantic web service discovery. In: IEEE 7th International Conference on Web Services (ICWS), IEEE Computer Society (2009)

[20] EU IST, FIT-IT: WSMO4J API (2008)

[21] openRDF.org: Sesame (2009)

[22] HP Labs Semantic Web: Jena (2008)

[23] Navigli, R., Velardi, P.: Structural Semantic Interconnections: a Knowledge-Based Approach to Word Sense Disambiguation. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 27., IEEE Computer Society (2005) 1075 – 1086

[24] Jiang, J., Conrath, D.: Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In: International Conference Research on Computational Linguistics (ROCLING X). (1997)

[25] Budanitsky, A., Hirst, G.: Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In: Proceedings of WordNet and Other Lexical Resources: Applications, Extensions and Customizations, NAACL 2001 Workshop (2001)

[26] Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: Wordnet: An on-line lexical database. International Journal of Lexicography **3** (1990) 235–244

[27] The University of Sheffield: Pure Java WordNet Similarity Library (2009)

[28] The Stanford Natural Language Processing Group: Stanford Log-linear Part-Of-Speech Tagger (2009)

[29] Herbrich, R., Zaragoza, H., Hill, S.: A Statistical Analysis of the Precision-Recall Graph. In: NIPS*2002 Beyond Classification and Regression: Learning Rankings, Preferences, Equality Predicates, and Other Structures. (2002)