Semantic Web Service Discovery Using Natural Language Processing Techniques

Jordy Sangers[a], Flavius Frasincar[a], Frederik Hogenboom[a,*], Vadim Chepegin[b]

[a]*Erasmus University Rotterdam, P.O. Box 1738, 3000 DR, Rotterdam, The Netherlands*
[b]*Tie Kinetix, P.O. Box 3053, 2130 KB, Hoofddorp, The Netherlands*

Abstract

This paper proposes a Semantic Web Service Discovery framework for finding Semantic Web services by making use of natural language processing techniques. The framework allows searching through a set of semantic Web services in order to find a match with a user query consisting of keywords. By specifying the search goal using keywords, end-users do not need to have knowledge about semantic languages, which makes it easy to express the desired semantic Web services. 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 performance evaluation shows that the three proposed matching algorithms are able to effectively perform matching and approximate matching.
Keywords: Natural language processing, Semantic Web services, WSMO, Web service discovery

1. Introduction

With the emergence of Web services and the Service Oriented Architecture (SOA), business process components are more and more being decoupled. Using Web services in SOA creates 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 cannot be used by machines to automatically process the descriptive information about a Web service.

*Corresponding author; tel: +31 (0)10 408 8907; fax: +31 (0)10 408 9031
Email addresses: jordysangers@hotmail.com (Jordy Sangers), frasincar@ese.eur.nl (Flavius Frasincar), fhogenboom@ese.eur.nl (Frederik Hogenboom), vadim.chepegin@tieglobal.com (Vadim Chepegin)

To promote the automation of Web service discovery (Keller et al., 2005) and composition (Hikimpour et al., 2005), a number of different semantic languages (Martin et al., 2004; de Bruijn et al., 2005; Vitvar et al., 2007) 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.

The semantic Web service descriptions use ontologies to describe the behavior of a Web service by applying reasoning over Web service semantics. In this way, the semantics described in ontologies enable systems to interpret what a Web service is doing, stimulating automatic Web service discovery and composition. The ontologies, however, are created by humans and therefore contain natural language. This allows humans to understand the concepts defined, but a system, in contrary to humans, can only understand ontology concepts and their relationships to a limited extend. Natural Language Processing (NLP) techniques can therefore help in better defining the context of a Web service.

When using one holistic ontology, machines can discover and compose Web services automatically based on the semantics defined. Using one holistic ontology is, however, hardly reachable and therefore it is impossible to reason based only on formal logic. NLP techniques can help overcome the ambiguity problems between different ontologies that are being used by semantic Web service descriptions.

Last, service composition should be driven by people who know business processes and not by technicians. Thus, end users must be able to discover these Web services based on keywords written in human language. Therefore, a discovery mechanism must be developed in such a way that a bridge between keywords written in a natural language, on one hand, and Web service descriptions provided using semantically enhanced languages, on the other hand, can be created.

In this paper, the Semantic Web Service Discovery (SWSD) framework is proposed, which enables end users to search, using keywords, for existing Web services described by means of a Semantic Web language for service annotation. This process consists of several steps including:

- Extraction of information from semantic descriptions to create the context of a Web service;
- NLP for disambiguating words' meanings and establishing a context for a set of words;
- Matching the users search context with a Web service context by means of a similarity measure.

The result of this process will be a ranked list of Web services that match the users search criteria. This context-based matchmaking mechanism provides flexibility by not only searching for exact word matches,

but also by looking for synonyms found in a popular lexical database as WordNet (Miller et al., 1990) and making use of its extensive network of relationships between different words (e.g., synonyms, hypernyms, etc.).

This work is based on our previous efforts (Sangers et al., 2012), in which we propose a linguistic approach for semantic Web service discovery, yet in our current endeavours we provide more details on the proposed approach and, additionally, we present a genetic algorithm-based method to learn the feature weights. The remainder of this paper is structured as follows. Section 2 discusses related technologies for describing and searching Web services. Next, Section 3 proposes the SWSD framework for discovery of Web services based on keywords. A possible implementation of the framework is described in Section 4. Subsequently, Section 5 evaluates different matching algorithms between the user input and a Web service description. Last, Section 6 concludes the paper and discusses future work.

2. Related Work

This section reviews state-of-the-art languages and tools for service discovery. First, a comparison is made between different types of Web service description languages in Section 2.1. Since this paper covers the discovery of semantic Web services, three semantic Web service description languages are described and compared with each other in Section 2.2. Last, several approaches for discovering Web services are discussed in Section 2.3.

2.1. Web Service Description Languages

Because Web service discovery depends to a large extent on how Web services are described, an overview of the different types of languages used for describing Web services is first given. These languages differ in models and formalisms used for describing Web services and which Web service properties 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 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 layer in order to have a good overview on the capability of a service.

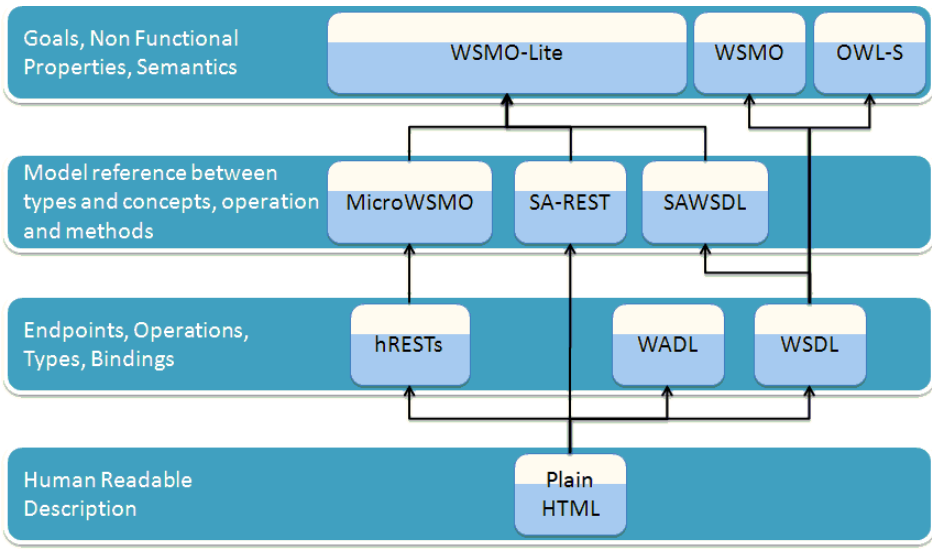

Figure 1: Web service description languages.

To make machines use Web services automatically, their interfaces are commonly defined in languages such as Web Service Description Layer (WSDL) (Christensen et al., 2001), Web Application Description Layer (WADL) (Hadley, 2009), or hRESTs (Kopecký et al., 2009). 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 on a syntactical level.

The top layer depicted in Figure 1 consists of semantic Web service description languages such as Web Service Modeling Ontology (WSMO) (de Bruijn et al., 2005) and OWL-S (Martin et al., 2004). These languages employ 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.

In order 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 (Kopecký et al., 2009), SA-REST (Lathem et al., 2007), and Semantic Annotations for WSDL (SAWSDL) (Farrell and Lausen, 2007) 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.

The latter two layers are not widely used yet. Therefore most Web service discovery systems use only the descriptions written in languages like WSDL. However, since semantics provide more information about

Web services, they can easily be used for Web service discovery. In our view, both types of languages can contribute to the discovery of Web services.

2.2. Semantic Web Service Description Languages

Table 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) (Bechhofer et al., 2004) and consists of different ontologies to describe a Web service. The language is defined in order to enable three major tasks which could not be fulfilled with older technologies (WSDL). These are automatic Web service discovery, automatic Web service invocation, and automatic Web service composition and interoperation. Because OWL-S uses ontologies to describe Web services, these 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. In recent work, Farrag et al. (2012) introduced an algorithm for mapping WSDL descriptions to OWL-S descriptions by making use of an ontology-based approach.

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 (de Bruijn, 2008) and can contain powerful logical formulae to describe the different WSMO elements. It also contains a grounding feature to link concepts with WSDL data types so that automatic invocation can be achieved.

Table 1: Core differences between OWL-S, WSMO and WSMO-Lite.

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

WSMO-Lite (Vitvar et al., 2007) 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 (Brickley and Guha, 2004) that can be used for annotations of various WSDL elements using SAWSDL annotation mechanism. WSMO-Lite only makes use of Ontologies and Web service descriptions and contains no grounding information, which makes it dependent of SAWSDL. The behavior of a Web service is only implicitly described by defining just preconditions and effects, so no specific how-questions can be answered.

Although these three languages have different properties, they all describe the semantics of a Web service. They provide information about the context domain, behavior and usage of a Web service. Because they use elements from predefined ontologies, those ontologies have to be used together with the semantic Web service descriptions for the discovery and matching of the services.

2.3. Web Service Discovery Engines

We can distinguish between two types of approaches for Web service discovery, i.e., discovery based on clustering operation parameters on the one hand, and rich semantics on the other hand. This section continues by elaborating on existing or proposed Web service discovery systems for each of these two different approaches.

2.3.1. Web service discovery based on clustering operation parameters

One approach for Web service discovery is by searching for similarities among different WSDL service descriptions, enabling searching for substitutable and composable Web services as similar operations and services can be discovered based on operation parameters. Web service operation semantics can be extracted and employed for discovery purposes.

Woogle (Dong et al., 2004) is a Web service search engine that employs clustering techniques for grouping operation parameters, and for a given query, it searches for similar and/or composable Web service operations. For this, Woogle automatically defines the underlying semantics of WSDL descriptions based on the operation parameters and uses these semantics to match operations. However, if independent ontologies which define the Web service semantics exist, the behavior of a Web service can be known without investigating parameter names and is therefore preferable to use.

Operation parameter clustering techniques are also employed in Seekda! (Semantic Technology Institute, 2009), which extracts service semantics from WSDL files, enabling runtime exchange of similar and composable services. Seekda! is part of the Service-Finder (Cefriel et al., 2009) framework, which is a

platform for service discovery where information about services is gathered from various sources like Web pages and blogs. The information is automatically added to a semantic model using automatic service annotation, realizing flexible discovery of services. Service-Finder uses service semantics for discovery and composition, but gathers this information dynamically and hence does not take into account predefined semantics.

2.3.2. Web service discovery using rich semantics

Another approach for semantic Web service discovery is the use of predefined ontologies. By identifying semantic similarities between ontologies, related semantic Web services can be discovered. To identify semantic similarities, Mediation Spaces can be used (Dietze et al., 2009). These Mediation Spaces mediate on data-level as well as semantic-level for discovery of related semantic Web services according to ontologies, other semantic Web services, or WSMO Goals.

GODO (Gomez et al., 2004) does not search for similar Web services, but instead employs a goal-driven approach. GODO has a repository with WSMO Goals and analyzes a user-described goal (in natural language). The WSMO Goal with the highest match will be sent to WSMX (DERI Galway, 2008), which is an execution environment for WSMO service discovery and composition. The WSMX environment subsequently searches for a WSMO Web service that is linked to the given WSMO Goal via WSMO Mediators and returns the WSMO Web service. Although this approach makes good use of the capabilities of the WSMO framework, it cannot be applied for other semantic languages like OWL-S and WSMO-Lite, as they do not have goal representation elements.

Bener et al. (2009) proposed an architecture that performs semantic matching of Web services based on both input and output descriptions, as well as preconditions and effects. In the semantic Web service matchmaking process, OWL-S descriptions are assumed (in contrast to the WSMO descriptions which are the focus of our research), and the matchmaking is done at the conceptual level using SWRL rules. Similar to the previous approach, Yang et al. (2008) make use of rules for matching semantic Web services. However, in contrast to the work of Bener et al. (2009), the authors take into account the service context, which is described by means of a context profile in OWL-S. Contexts are gathered through a Java Expert System Shell (JESS)-enabled context elicitation system featuring an ontology-based context model formally describing and acquiring contextual information. Service matching is done based in inputs and outputs and can result in an exact match, a plug-in match, and a subsumed match.

Luna et al. (2013) proposed BAX-SET PLUS, which is a multi-agent taxonomy-based method for categorization, search, and retrieval, of semantic Web services described in OWL-S. Here, user-selected concepts from a taxonomy are matched against concepts contained in OWL-S service descriptions. An important difference is that, in contrast to the work presented by the authors, we extract concepts from a natural language query instead of from an existing taxonomy that is browsed by the user. Moreover, our work focuses on WSMO service descriptions in WSML, instead of the OWL-S specifications focused on by Luna et al. (2013). Also, for all aforementioned approaches, no deep linguistic analysis, like proposed in our paper, is performed.

Recently, Paulraj and Swamynathan (2012) proposed a method for content-based semantic Web service discovery. In contrast to the general approach where user queries are matched against OWL-S inputs, outputs, preconditions, and effects (IOPE), the framework allows users to submit free text as input. This alleviates the restrictions put on user queries in that they must be of the same format as that of the IOPEs present in OWL-S. In their work, nouns are extracted from text that is initially unstructured. These nouns are subsequently used for service discovery, after a disambiguation process that makes use of the WordNet lexical database for determining the meaning of the nouns. Although there are similarities with the work presented in our paper, it should be noted that, similar to many semantics-based Web service discovery frameworks, Paulraj and Swamynathan (2012) focus specifically on OWL-S. In contrast, we focus on WSMO and aim for a more universal approach that can be utilized in various semantic Web service description languages (with a few adaptations). Although both in (Paulraj and Swamynathan, 2012) and our work, inputs are initially unstructured, in our current endeavours we provide a means for ranking the results, while Paulraj and Swamynathan (2012) do not implement any form of result ranking.

3. Semantic Web Service Discovery Framework

The SWSD framework comprises a keyword-based discovery process for searching Web services that are described using a semantic language. The search mechanism incorporates NLP techniques in order to establish a match between a user search query and a semantic Web service description. Logics-based specifications that are defined in the Web service descriptions are not taken into consideration, but the definitions of concepts stated in other imported ontologies are exploited. In this way, the framework is able to establish a broader search field by also using related concepts from the ontologies for identifying the context in which the Web service is operating.

Section 3.1 covers the global architecture of the SWSD framework and a short description of the 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. In Section 3.3 is presented how the context of a semantic Web service is established. Last, Section 3.4 describes two different algorithms to match a user query with a semantic Web service using the query and service contexts.

3.1. Framework Architecture

As an input, the SWSD framework requires a set of Web services that are described in semantic languages (e.g., WSMO (de Bruijn et al., 2005), WSMO-Lite (Vitvar et al., 2007), or OWL-S (Martin et al., 2004)). The descriptions are subsequently analyzed, resulting in the extraction of words that could represent the context of the Web services (i.e., the names of the operations, and nouns and verbs stated in non-functional descriptions of concepts or conditions). Next, the extracted words are disambiguated, as multiple senses can be assigned to the same words. Last, the disambiguated words are matched with the disambiguated words from the search query, resulting in a ranked list of Web services.

Figure 2 describes the architecture of the SWSD framework. The process is subdivided into three major tasks, i.e., Service Reading, Word Sense Disambiguation (WSD), and Match Making. Service Reading comprises parsing a semantic Web service description, extracting names and non-functional descriptions of used concepts. The WSD task subsequently determines the senses of a set of words. Last, the Match Making step determines the similarity between the different sets of senses, which is ultimately used for ranking the analyzed 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 and their subelements, conditions and effects, help in understanding the context of the Web service. Thereby, by extracting concepts out of the `definedBy` statement of those

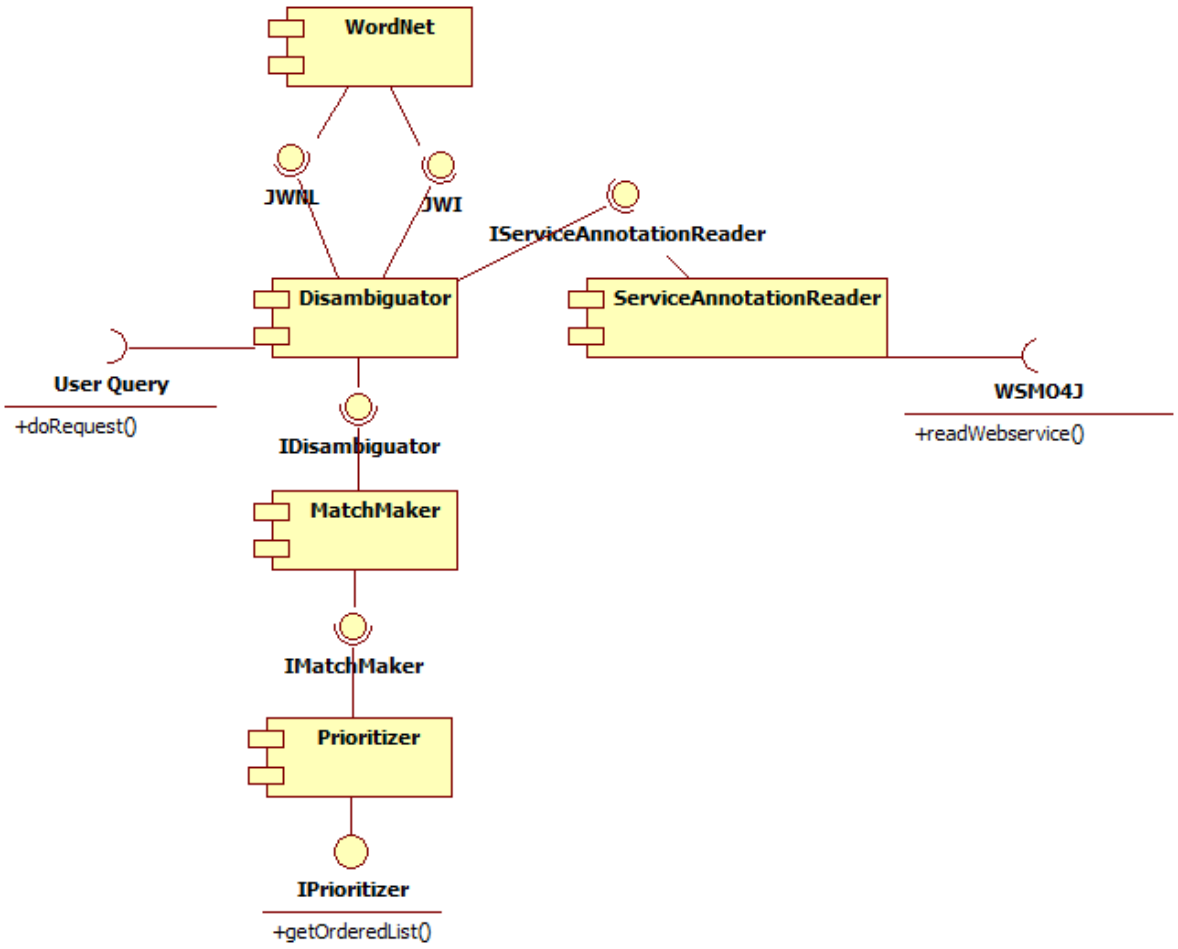

Figure 2: SWSD architecture.

elements and searching their non-functional descriptions in an ontology, the context of the Web service can be determined in a more accurate manner than just by using service names. 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 service 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 (EU IST and FIT-IT, 2008) can help in this process and for OWL-S and WSMO-Lite, which are written using an OWL (Bechhofer et al., 2004) or RDF (Klyne and Carroll, 2004) language, a parser like Sesame (openRDF.org, 2009) or Jena (HP Labs Semantic Web, 2009) 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 in "camel notation" (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 be useful. Using a POS-tagger, nouns and verbs from sentences are extracted and used for the WSD.

3.3. Word Sense Disambiguation

A user can represent its goal by defining a sets of words (nouns and verbs). Because many words have associated multiple meanings (e.g., *mouse* can be used to represent either an animal or a computer device), disambiguation of word senses helps in finding the correct context. The disambiguation will be applied to the set of words gathered from the user input and from a semantic Web service description, resulting in two sets of disambiguated senses that can be employed for matching.

As non-supervised WSD allows disambiguation of words without user interference, we use a variant of the SSI algorithm (Navigli and Velardi, 2005) for retrieving the senses out of a set of words, which is defined as follows:

$$selectedSense(word) = \arg \max_{s_j \in senses(word)} \sum_{sc_i \in I} sim(s_j, sc_i) \; . \tag{1}$$

The algorithm disambiguates a word (*word*) based on a previously disambiguated set of words and their related senses. Per word sense s_j, a similarity with the senses from the context (sc_i) is calculated, and the sense with the highest similarity is chosen. Subsequently, the word and its chosen sense are added to context I and the process is iterated until there are no ambiguous words left.

Naturally, at the start of the process, a context is not yet established, and hence $I = \emptyset$. Hence, we fill context I with all monosemous words, i.e., words that have only one sense, as these do not require any disambiguation. Subsequently, the iterative process of disambiguating polysemous words can be initiated, always targeting the least ambiguous words (i.e., with the least amount of senses) first. For each of the available 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 run is used for the context initialization.

Because previous studies have shown that the method of Jiang and Conrath (Jiang and Conrath, 1997) performs better than other Semantic distance measures (Budanitsky and Hirst, 2001), we employ this

method for computing the sense similarities. The measure makes use of the Information Content (IC), which depends on the probability of the occurrence of a particular sense in a large corpus:

$$IC(sense) = \frac{1}{\log P(sense)} \ . \tag{2}$$

Formula (3) gives the similarity between two senses. Besides the IC, also the Least Common Subsumer (LCS) between two concepts is used, representing the first ancestor concept that subsumes both senses when going in a bottom-up direction through a hypernym tree of senses. The IC of the LCS is computed and compared with the IC of the two senses:

$$sim(s_i, s_j) = \frac{1}{IC(s_i) + IC(s_j) - 2 \times IC(LCS(s_i, s_j))} \ . \tag{3}$$

3.4. Sense Matching

After the disambiguation of all words gathered from the user input and a semantic Web service description, one is left with several different sets of senses. For the matching process, each word in the user query is assumed to be equally important and hence the user input contains one set of senses. However, a Web service description can contain words that better represent the context of the Web service 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 starts with a high level view of the matching between the user input and the different levels of information extracted from a Web service description. Next, the Jaccard matcher for matching sets of words or senses is described. Last, a matching approach based on similarities between words or senses is explained.

3.4.1. Level Matching

Not only the disambiguated senses are matched, but also words that are not appearing in the used lexicon, as these words can represent important names or concepts for the discovery of Web services. Hence, for matching user input with a semantic Web service description, the user input contains a set of ambiguous words ws_u and a set of senses ss_u. The Web service description on the other hand contains multiple sets of words mws_w and multiple sets of senses mss_w. Because the Web service description, presented in the next section, provides n 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 is computed as 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:

$$finalSim(ss_u, mss_w, ws_u, mws_w) = \sum_{i=1}^{n} w_i \times levelSim(ss_u, mss_{wi}, ws_u, mws_{wi}) \ . \tag{4}$$

Here, the weights w_i are established through optimization techniques and 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.

For each set of words and senses from the Web service description, the system employs two measures, i.e., one for sense matching (defined in Formulas (6) and (8)), and one for the matching of non-disambiguated words (see Formulas (7) and (11)). These measures range between 0 (no match) and 1 (exact match) and are combined into a single measure using a weighted average:

$$levelSim(ss_u, ss_w, ws_u, ws_w) = w_{sense} \times senseSim(ss_u, ss_w) + w_{word} \times wordSim(ws_u, ws_w) \ , \tag{5}$$

which is used n times in Formula (4), where $mss_{wi} = ss_w$ and $mws_{wi} = ws_w$. The weights corresponding to the sense similarity and the word similarity are, as with the final similarity, established by means of optimization techniques and must sum up to 1.

3.4.2. Jaccard Matching

The Jaccard matcher, which is employed for matching sets of words or senses, makes use of the Jaccard Index (Jaccard, 1901). This method is often used for computing set similarity and thus compares different sense sets:

$$senseSim(ss_u, ss_w) = \frac{|ss_u \bigcap ss_w|}{|ss_u \bigcup ss_w|} \ . \tag{6}$$

Subsequently, we calculate a similarity coefficient by dividing the number of senses appearing in both sets by the total number of senses in both sets. This results in a percentage of exact matching items and can also be applied for matching the words that could not be disambiguated:

$$wordSim(ws_u, ws_w) = \frac{|ws_u \bigcap ws_w|}{|ws_u \bigcup ws_w|} \ . \tag{7}$$

3.4.3. Similarity Matching

Normally, for the calculation of similarity values, only perfect matching items are used. In order to additionally take into consideration partial matches, the similarity matcher makes use of a similarity-based approach for matching different sets of senses or non-disambiguated words, expressing close relatedness with values approaching 1, and non-relatedness with values close to 0.

Sense set similarities are calculated using the same similarity function as in WSD. The similarity between the user set of senses ss_u and a Web service set of senses ss_w is computed as follows:

$$senseSim(ss_u, ss_w) = \sum_{s_u \in ss_u} \frac{senseScore(s_u, ss_w)}{|ss_u| + |ss_w|} + \sum_{s_w \in ss_w} \frac{senseScore(s_w, ss_u)}{|ss_u| + |ss_w|} , \tag{8}$$

where 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.

The similarity between a sense s_a and a set of senses ss_b is calculated by taking the maximum similarity between the sense and one of the senses s_b from the other set, i.e.:

$$senseScore(s_a, ss_b) = \max_{s_b \in ss_b} senseNorm(s_a, s_b) . \tag{9}$$

The similarities calculated with Formula (3) range between 0 and infinity, yet we prefer a range between 0 and 1 for quantifying the matching degree. Hence, we employ a logarithmic function in order to transform the values of the similarity:

$$senseNorm(s_a, s_b) = 1 - e^{-sim(s_a, s_b)} . \tag{10}$$

Here, exact similar senses will have 1 as resulting similarity and a total mismatch between senses will result in 0.

As we are unable to determine the senses of some words, we fall back to non-semantic measurements of similarities for matching the sets of non-disambiguated words. For this, we employ the Levenshtein distance metric (Levenshtein, 1966), which calculates the total number of operations that need to be done in order to transform one word to another. The similarity between two sets of words is subsequently calculated similarly to the comparison of two sense sets, yet now the Levenshtein distance is applied instead of the similarity function from WSD.

We calculate the similarity between the user set of words ws_u and a Web service set of words ws_w as:

$$wordSim(ws_u, ws_w) = \sum_{w_u \in ws_u} \frac{wordScore(w_u, ws_w)}{|ws_u| + |ws_w|} + \sum_{w_w \in ws_w} \frac{wordScore(w_w, ws_u)}{|ws_u| + |ws_w|} , \tag{11}$$

where the similarity between a word and a set of words, $wordScore(w_a, ws_b)$ is computed as:

$$wordScore(w_a, ws_b) = \max_{w_b \in ws_b} wordNorm(w_a, w_b) , \tag{12}$$

and the Levenshtein distance is used in the following way for comparing two words w_a and w_b:

$$wordNorm(w_a, w_b) = \max(0, 1 - 2 \times \frac{levenshtein(w_a, w_b)}{maxLength(w_a, w_b)}) . \tag{13}$$

Here, $maxLength(w_a, w_b)$ denotes the number of tokens of the longest word that is being compared. If this formula returns a negative value, which means that the amount of changes that should be made exceed half of the length of the largest word, a value of 0 will be used to indicate a total mismatch.

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, this is one possible implementation of the framework, 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 service discovery. How the context of a semantic Web service is established is presented in Section 4.3. Last, Section 4.4 describes how matching algorithms are used for computing the similarity between the user input and 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 (de Bruijn et al., 2005) framework as WSMO is a powerful framework that can contain very wide information about Web services. So, 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 (EU IST and FIT-IT, 2008) 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 WSD, WordNet (Miller et al., 1990) is used through two WordNet API's, in order to find senses belonging to words. The Java WordNet Library (JWNL) (Walenz and Didion, 2011) is used for the morphological analysis of each word, and the MIT Java WordNet Interface (JWI) (Finlayson, 2012) is subsequently employed for retrieving WordNet synsets. Next, JWordNetSim (The University of Sheffield, 2009) is used to calculate the similarity between two WordNet senses because it contains an implementation of the Jiang and Conrath formula, which we proposed to use in the SWSD framework. For the part-of-speech tagging, the Stanford

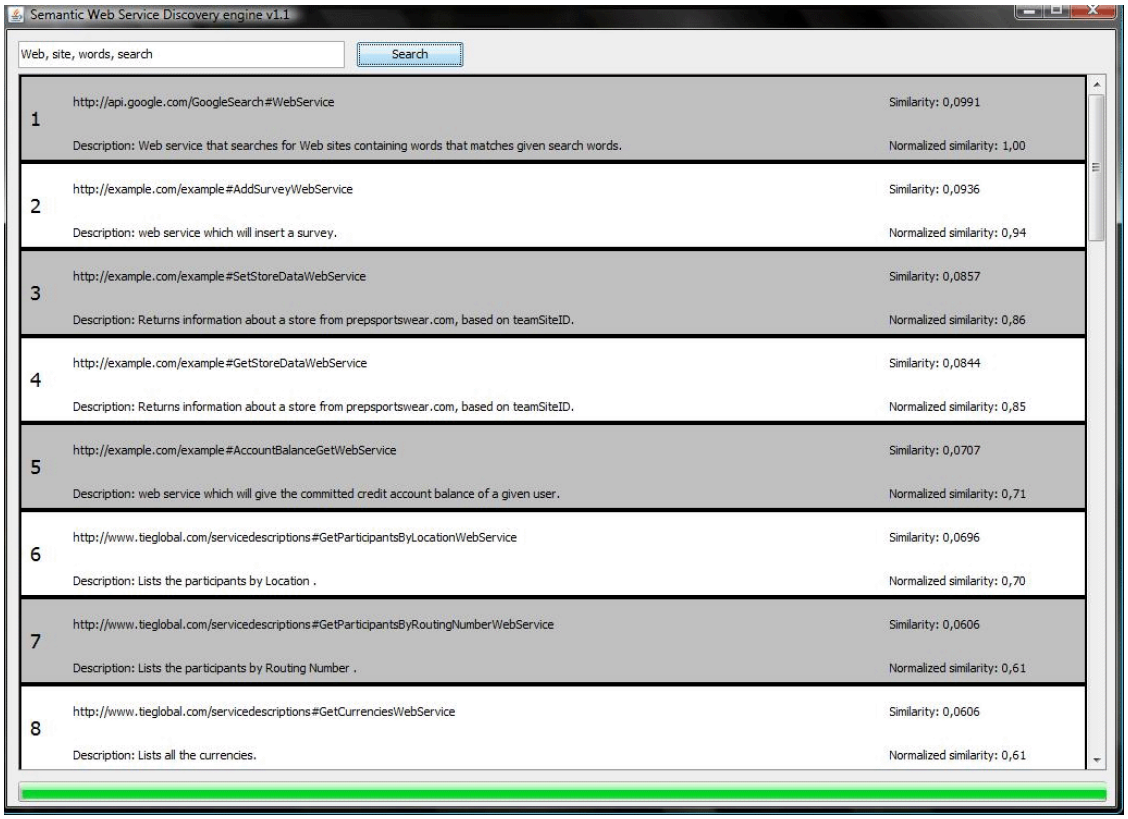

Figure 3: Result presentation example.

parser (The Stanford Natural Language Processing Group, 2009) is used. The overall implementation is made in Java, due to the availability of external packages written in this language for WSD and WSMO parsing and reading.

Figure 3 shows the user interface of the SWSD engine. The user can fill in a comma-separated list of words representing his goal and click the search button. The system will then check if each word given by the user can be a noun, verb, both noun or verb, or something else. This is needed since the WSD process can only be used for a set of words containing the same POS tag. For the words that do not have a clear POS, the user will be asked to select the appropriate POS to use for the search. After that, the system will do the search and 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 (computed by dividing the similarity score with the highest similarity score).

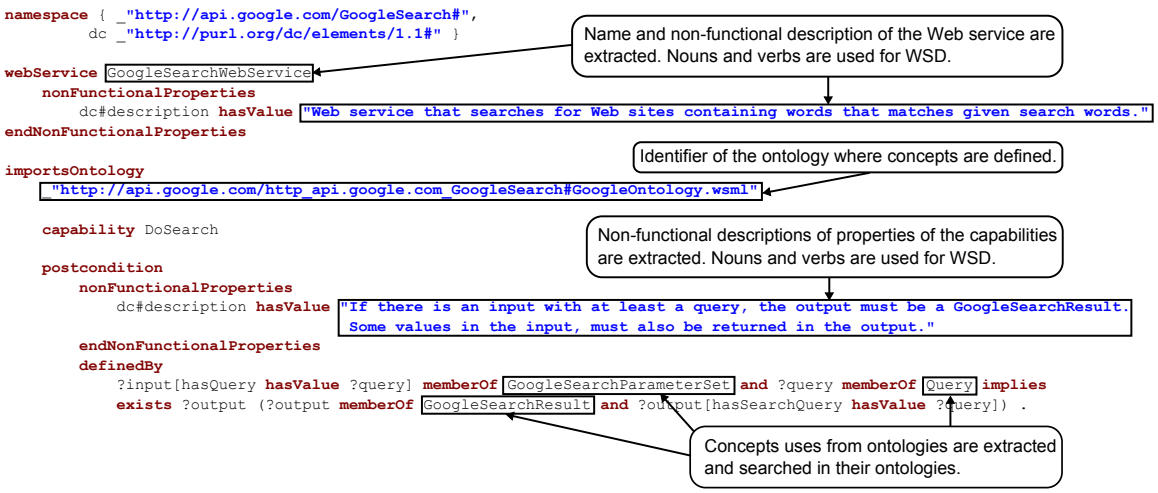

Figure 4: A WSMO Web service information extraction example.

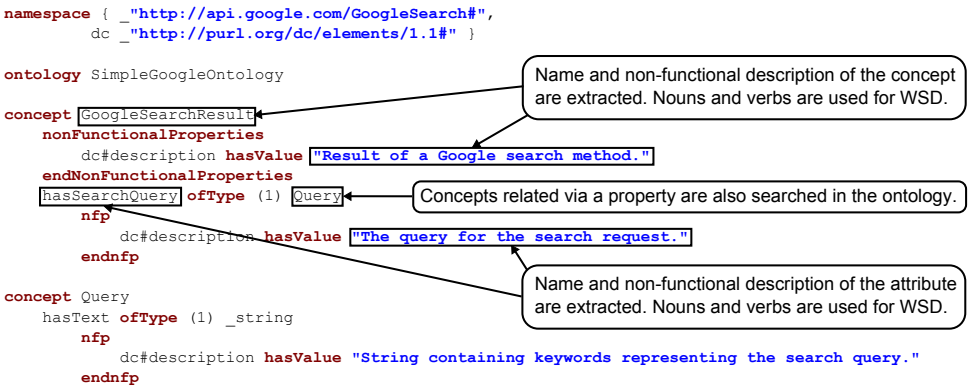

Figure 5: A WSMO ontology information extraction example.

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 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 are used to search in ontologies to find their description. Figure 5 shows an example WSMO Ontology containing some concepts used in the Web service description from Figure 4. 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 Web service. Each of these levels has a different amount of importance to 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, 0.210, (direct relation, from Web service);
- Non-functional descriptions of properties of capabilities of the Web service, 0.167, (direct relation, from Web service);
- Non-functional descriptions and names of concepts used by Web service, 0.191, (direct relation, from ontology);
- Non-functional descriptions and names of attributes of concepts used by the Web service, 0.008, (indirect relation, from ontology);
- Non-functional descriptions and names of concepts related via attributes with concepts used by the Web service, 0.153, (indirect relation, from ontology);
- Non-functional descriptions and names of superconcepts of the concepts used by the Web service, 0.158, (semi-direct relation, from ontology);
- Non-functional descriptions and names of subconcepts of the concepts used by the Web service, 0.113, (semi-direct relation, from ontology).

The weights are established by making use of a Genetic Algorithm, which is a specific technique to find approximate solutions for optimization problems. In our case, we want to find a set of weights which optimize the search results. Genetic Algorithms mimic an evolution of individuals in a certain population. Each individual is represented with a chromosome consisting of data that can be recombined and mutated during next generations. We choose to represent the individuals using a chromosome consisting of nine variables, the seven weights used for stating the importance of the information gathered from a Web service, and two weights for setting the trade-off between the sense similarity and the lexical representation similarity.

From each generation, the best individuals will survive and create an offspring. To define which individuals are good enough to survive, a fitness function is used. In our case, this is a function that takes the average precision values for each query used in our two data sets described in Section 5. Last, the genetic algorithm has been executed with a population size of 100 and the number of generations set to 50.

As can be seen from the weights, the information directly related to the Web service is the most important information for the matching process. This information consists of the name and non-functional descriptions of the Web service, its capabilities, and concepts used. Less important for the matching are the non-functional descriptions and names of the super- and subconcepts of the used concepts by the Web service. Least important and thus given a low weight is the information that is indirectly related to the Web service description. This information is represented by attributes and concepts related to the concepts used by the Web service. Using these weights, the information that is semantically closest to the core Web service description will have the most impact for the matching process.

The names and non-functional descriptions of the entities returned from these two readers will then go through a NLP 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 Figure 4 (*"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 (as only these can be found in WordNet).

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 in WordNet by giving 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.

Once having a context, the SSI algorithm will be used to find the synsets of the other words. The similarity is measured using an implementation of the Jiang and Conrath method (Greenwood, 2007). 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 importance 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, than the names of related concepts. These weights are determined using a Genetic Algorithm 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 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 that are not in WordNet. 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, using a weighted average, we determine a final similarity value. The weights have been optimized by the genetic algorithm described in Section 4.2 and set to 44/100 for the synset similarity and 56/100 for the lexical representation similarity. The value of the synset similarity is higher than the lexical representation similarity as they provide for a richer information comparison.

5. Evaluation

This section covers the evaluation of different matching algorithms that can be used for semantic Web service discovery. The algorithms described in Section 3.4 are implemented in the SWSD engine and are

evaluated by 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 Section 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. Experimental Setup

For testing the algorithms provided in Section 3.4, we make use of a repository of 35 WSMO Web service descriptions stored in WSML format. As for WSMO Web service descriptions there are no existing data sets readily available such as for instance the OWLS-TC 4.0 OWL-S service retrieval test collection (Klusch et al., 2010), we have manually built 35 WSMO service descriptions.

For evaluation purposes, we have defined 61 test queries in order to analyze the outcomes of each of the algorithms. These queries represent possible sets of keywords a user could use when searching for a Web service. For each query, we have defined a list of preferred Web services (present in the repository of the SWSD engine) that should be returned with a high ranking, as they all are significantly related to the queries.

Besides the algorithms described in Section 3.4, a simple matching algorithm that does not make use of natural language processing is added for the evaluation. The algorithm employs the Jaccard matching algorithm only for lexical representations of the query keywords and the extracted words. The algorithm makes use of the different information levels of a Web service and can therefore be used for testing whether there is an added value in using NLP in the process of semantic Web service discovery.

The testing is done with usage of precision and recall metrics. 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 y_j is stated as preferred, then the Web service is classified 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:

$$p_i = \frac{\sum_{j=1}^{i} y_j}{i} , \tag{14}$$

$$r_i = \frac{\sum_{j=1}^{i} y_j}{k} . \tag{15}$$

Figure 6 shows the calculation of precision and recall for a given list of classified Web services. For each preferred Web service – 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. The PR-graph can show 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.

5.2. Experimental Results

To test the performances of the three matching algorithms, we divide our 61 predefined queries into two types. In total, 33 queries are used for measuring the matching performance of the algorithms for searches for Web services that are present in the repository. The remaining 28 queries are used for measuring the performance of queries for Web services that are not present in the repository. In the latter set of queries, a number of similar Web services from the repository are defined in order to test performance of similar Web service discovery when a specified service does not exist in the repository. Both sets of queries are divided into two parts, a training set and a test set. The training set, consisting of 40 queries, is used for training the weights that are used by the matching algorithms, while the test set, having 21 queries, is used for testing the performance of the matching algorithms.

Testing with 21 queries and three matching algorithms generates 63 PR-graphs. Visualizing this amount of graphs is not very insightful and hence we create PR-graphs consisting of average precision values for the recall points. This enables us to relate all the different algorithms at once. However, the testing is done with

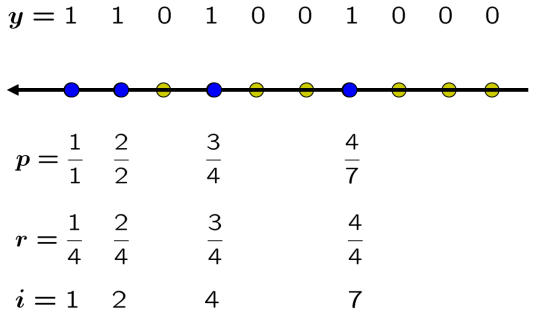

Figure 6: Precision and recall calculation example.

lists of preferred Web services that can vary in the number of Web services they consists of. For testing, lists that contain two to five preferred Web services have been used. Because these variations in number of Web services cause different recall values, average precision values was only be calculated for queries that have the same amount of preferred Web services.

Hence, we analyze eight different PR-graphs that visualize the performances of the different matching algorithms. The four PR-graphs that are depicted in Figure 7, show the average results for the exact matching tests. For each of the four different numbers of preferred returned Web services (n) a PR-graph is created. The four PR-graphs that are shown in Figure 8, show the average results for the approximate matching tests.

From the different PR-graphs that are shown in Figure 7, we can make two observations. First, in most cases, the Jaccard algorithm shows a higher precision for most recall values than the simple and the similarity algorithm. Second, all algorithms require about the same precision for providing a full recall. This means that in order to provide all the preferred Web services to the user, the algorithms are required to display about the same amount of Web services. 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 than 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 Figure 8, we can make the observation that the similarity algorithm performs overall better for discovery of similar Web services than the Jaccard and the simple matching algorithm, as in most of the cases the PR-graph lines of the similarity algorithm are above the lines of the Jaccard algorithm and the simple matching algorithm.

5.3. Summary

In order to test the performance of the three matching algorithms explained in Section 3.4, we have performed 61 tests, of which 31 tests were done to measure the performance of the algorithms according to discovery of exact matching Web services, and 28 tests were done to measure the performance of the algorithms according to discovery of similar Web services. Each test consisted of a set of keywords used as a query and a set of Web services that were preferred to be provided to the user as early as possible. Based on our evaluation, we conclude that the Jaccard matching algorithm is the best method for discovering exact matching Web services, whereas the similarity matching algorithm is the best method for discovering similar Web services.

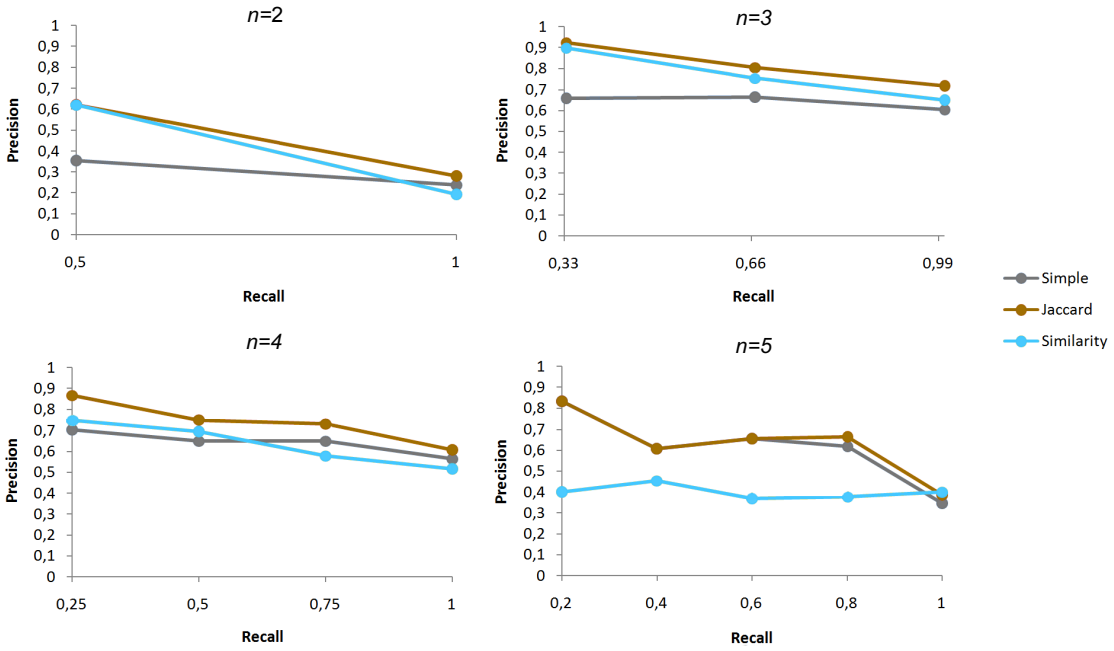

Figure 7: PR-Graphs for discovery of exact matching services.

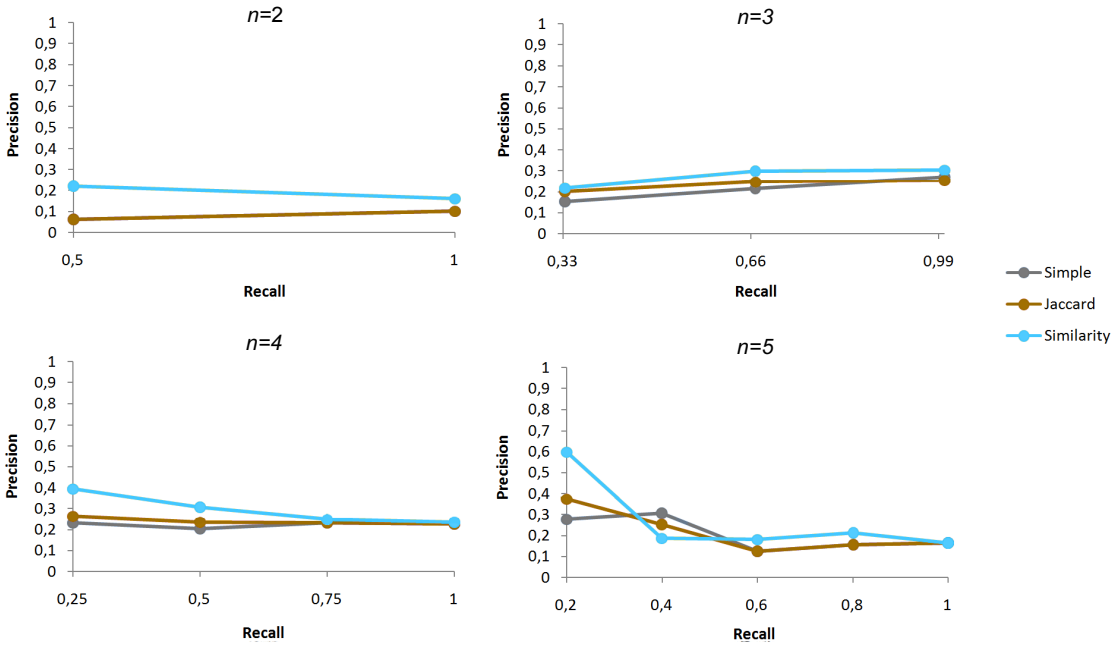

Figure 8: PR-Graphs for discovery of similar services.

6. Conclusions and Future Work

The SWSD framework proposes a keyword-based discovery process for searching Web services that are described using semantically enriched annotations done by means of semantic languages for service description. It makes an intensive use of natural language processing techniques and a WordNet-based similarity measure for matching keywords. By using an approach with two different similarity functions, one for lexical similarities and one for semantic similarities, standard lexical matching algorithms as well as semantic matching algorithms, using ontologies, can be applied for discovery of semantic Web services.

The SWSD engine can search for WSMO Web services based on user search keywords. A matching score is computed based on the similarity between the words in the user query and a Web service description. Experiments have been done to test the performance of three different matching algorithms. The Jaccard matching algorithm performs best for discovering exact matching Web services, while matching using a similarity approach gives the best results for finding similar Web services.

As a future work, 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. Also, additional information about a Web service (e.g., signature of its operations, messages being exchanged, etc.) could be retrieved from its WSML specification. With this information, the context of the Web service can be described in more detail.

References

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., 2004. OWL Web Ontology Language Reference – W3C Recommendation 10 February 2004. From: `http://www.w3.org/TR/owl-ref/`.

Bener, A. B., Ozadali, V., Ilhan, E. S., 2009. Semantic Matchmaker with Precondition and Effect Matching Using SWRL. Expert Systems with Applications 36 (5), 9371–9377.

Brickley, D., Guha, R., 2004. RDF Vocabulary Description Language 1.0: RDF Schema – W3C Recommendation 10 February 2004. From: `http://www.w3.org/TR/rdf-schema/`.

Budanitsky, A., Hirst, G., 2001. Semantic Distance in WordNet: An Experimental, Application-Oriented Evaluation of Five Measures. In: Workshop on WordNet and Other Lexical Resources at 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001). Association for Computational Linguistics, pp. 29–34.

Cefriel, Seekda!, Ontoprise, University of Sheffield, 2009. Service-Finder. From: `http://www.service-finder.eu/`.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2001. Web Services Description Language (WSDL) 1.1 – W3C Note 15 March 2001. From: `http://www.w3.org/TR/wsdl`.

de Bruijn, J., 2008. D16 The WSML Specification – WSML Working Draft 2008-08-08. From: http://www.wsmo.org/TR/d16/.

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., 2005. Web Service Modeling Ontology (WSMO) – W3C Member Submission 3 June 2005. From: http://www.w3.org/Submission/WSMO/.

DERI Galway, 2008. Web Service Execution Environment. From: http://www.wsmx.org/.

Dietze, S., Gugliotta, A., Domingue, J., 2009. Exploiting Metrics for Similarity-based Semantic Web Service Discovery. In: IEEE 7th International Conference on Web Services (ICWS 2009). IEEE Computer Society, pp. 327–334.

Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J., 2004. Similarity Search for Web Services. In: 30th International Conference on Very Large Data Bases (VLDB 2004). Vol. 30. pp. 372–383.

EU IST, FIT-IT, 2008. WSMO4J API. From: http://wsmo4j.sourceforge.net/.

Farrag, T. A., Saleh, A. I., Ali, H. A., 2012. Towards SWSs Discovery: Mapping from WSDL to OWL-S Based on Ontology Search and Standardization Engine. IEEE Transactions on Knowledge and Data Engineering. To appear (DOI: 10.1109/TKDE.2012.25).

Farrell, J., Lausen, H., 2007. Semantic Annotations for WSDL and XML Schema – W3C Recommendation 28 August 2007. From: http://www.w3.org/TR/sawsdl.

Finlayson, M., 2012. JWI: The MIT Java WordNet Interface. From: http://projects.csail.mit.edu/jwi/.

Gomez, J. M., Rico, M., Garcia-Sanchez, F., Bejar, R. M., Bussler, C., 2004. GODO: Goal Driven Orchestration for Semantic Web Services. In: 1st Workshop on Web Services Modeling Ontology Implementations (WIW 2004). Vol. 113. CEUR Workshop Proceedings.

Greenwood, M., 2007. JWordNetSim. From: http://nlp.shef.ac.uk/result/software.html.

Hadley, M. J., 2009. Web Application Description Language – W3C Member Submission 31 August 2009. From: http://www.w3.org/Submission/wadl/.

Hikimpour, F., Sell, D., Cabral, L., Domingue, J., Motta, E., 2005. Semantic Web Service Composition in IRS-III: The Structured Approach. In: 7th IEEE International Conference on E-Commerce Technology (CEC 2005). IEEE Computer Society, pp. 484–487.

HP Labs Semantic Web, 2009. Jena. From: http://jena.sourceforge.net/.

Jaccard, P., 1901. Étude Comparative de la Distribution Florale dans une Portion des Alpes et des Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 37, 547–579.

Jiang, J., Conrath, D., 1997. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In: International Conference Research on Computational Linguistics (ROCLING X). pp. 19–33.

Keller, U., Lara, R., Lause, H., Polleres, A., Predoiu, L., Toma, I., 2005. Semantic Web Service Discovery – WSMX Working Draft - October 3, 2005. From: http://www.wsmo.org/TR/d10/v0.2/d10.pdf.

Klusch, M., Kapahnke, P., Fries, B., Khalid, M. A., Vasileski, M., 2010. OWLS-TC 4.0 – Release 21 September 2010. From: http://semwebcentral.org/projects/owls-tc/.

Klyne, G., Carroll, J. J., 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax – W3C Recommendation 10 February 2004. From: http://www.w3.org/TR/rdf-concepts/.

Kopecký, J., Vitvar, T., Fensel, D., Gomadam, K., 2009. D12v0.1 hRESTS & MicroWSMO – CMS WG Working Draft 10 March

2009. From: http://cms-wg.sti2.org/TR/d12/v0.1/.

Lathem, J., Gomadam, K., Sheth, A. P., 2007. SA-REST and (S)mashups: Adding Semantics to RESTful Services. In: 1st International Conference on Semantic Computing (ICSC 2007). IEEE Computer Society, pp. 469–476.

Levenshtein, V. I., 1966. Binary Codes Capable of Correction Deletions, Insertions, and Reversals. Soviet Physics Doklady 10 (8), 707–710.

Luna, J. A. G., Pardo, I. D. T., Builes, J. A. J., 2013. Recent Progress in Data Engineering and Internet Technology. Vol. 156 of Lecture Notes in Electrical Engineering. Springer, Ch. BAX-SET PLUS: A Taxonomic Navigation Model to Categorize, Search and Retrieve Semantic Web Services, pp. 359–364.

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., 2004. OWL-S: Semantic Markup for Web Services – W3C Member Submission 22 November 2004. From: http://www.w3.org/Submission/OWL-S/.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K., 1990. Introduction to WordNet: An On-Line Lexical Database. International Journal of Lexicography 3 (4), 235–244.

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

openRDF.org, 2009. Sesame. From: http://www.openrdf.org/.

Paulraj, D., Swamynathan, S., 2012. Advanced Computing, Networking and Security. Vol. 7135 of Lecture Notes in Computer Science. Springer, Ch. Content Based Service Discovery in Semantic Web Services Using WordNet, pp. 48–56.

Sangers, J., Frasincar, F., Hogenboom, F., Hogenboom, A., Chepegin, V., 2012. A Linguistic Approach for Semantic Web Service Discovery. In: Casillas, J., Martínez-López, F. J., Corchado, J. M. (Eds.), First International Symposium on Management Intelligent Systems (IS-MiS 2012). Vol. 171 of Advances in Intelligent Systems and Computing. Springer, pp. 131–142.

Semantic Technology Institute, 2009. Seekda! From: http://seekda.com/.

The Stanford Natural Language Processing Group, 2009. Stanford Log-linear Part-Of-Speech Tagger. From: http://nlp.stanford.edu/software/tagger.shtml.

The University of Sheffield, 2009. Pure Java WordNet Similarity Library. From: http://nlp.shef.ac.uk/result/software.html.

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

Walenz, B., Didion, J., 2011. JWNL: Java WordNet Library. From: http://sourceforge.net/projects/jwordnet/.

Yang, S. J. H., Zhang, J., Chen, I. Y. L., 2008. A JESS-Enabled Context Elicitation System for Providing Context-Aware Web Services. Expert Systems with Applications 34 (4), 2254–2266.