A Linguistic Approach for Semantic Web Service Discovery

Jordy Sangers, Flavius Frasincar, Frederik Hogenboom, Alexander Hogenboom, and Vadim Chepegin

Abstract We propose a Semantic Web Service Discovery framework for finding semantically annotated Web services by using natural language processing techniques. The framework searches through a set of annotated Web services for matches with a user query, which consists of keywords, so that knowledge about semantic languages is not required. For matching keywords with Semantic Web service descriptions given in Web Service Modeling Ontology (WSMO), techniques like part-of-speech tagging, lemmatization, and word sense disambiguation are used. Three different matching algorithms are defined and evaluated for their ability to do exact matching and approximate matching between the user query and Web Service descriptions.

1 Introduction

With the emergence of Web services and the Service Oriented Architecture (SOA), business process components are increasingly decoupled, while systems and business processes converge, forcing companies to change their management strategies. The usage of Web services in SOA creates a wide network of services that collaborate in order to implement complex tasks. 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 hence cannot be used by machines to automatically process the descriptive information about a Web service.

Jordy Sangers (e-mail: jordysangers@hotmail.com) · Flavius Frasincar (e-mail: frasincar@ese.eur.nl) · Frederik Hogenboom (e-mail: fhogenboom@ese.eur.nl) · Alexander Hogenboom (e-mail: hogenboom@ese.eur.nl)
Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands

Vadim Chepegin (e-mail: vadim.chepegin@tieglobal.com)
Tie Kinetix, P.O. Box 3053, NL-2130 KB, Hoofddorp, The Netherlands

Several semantic languages [1, 13, 19] have been created to aid machines in automatically processing information on Web services. These languages 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 [5]. These Semantic Web service descriptions use ontologies to describe the behavior of a Web service by applying reasoning over their semantics. The semantics described in ontologies enable systems to interpret what a Web service is doing, stimulating service discovery [3] and composition [11]. The ontologies, however, are created by humans and therefore contain natural language as ontology meta-data. This lets humans understand the concepts defined, while a system can only understand concepts and their relationships to a limited extent as specified in the ontology axioms. Natural Language Processing (NLP) techniques can therefore help in better defining the context of a Web service.

When using a single 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 overcoming the ambiguity problems between multiple ontologies that are being used by Semantic Web service descriptions. Service composition is often driven by people with expertise in business processes and not by technicians. Thus, end users must be able to discover these Web services based on a search query written in an easy to understand language, i.e., human language. Therefore, a discovery mechanism must be developed in such a way that Semantic Web services can be found using natural language queries.

In this paper the Semantic Web Service Discovery (SWSD) framework is proposed, which enables users to search (using keywords) for existing Web services that are described by means of a Semantic Web language for service annotation. This process consists of several steps including information extraction, word sense disambiguation, and matching the user search context and Web service context by means of a similarity measure. The result of this process is a ranked list of Web services that match the users search criteria.

This paper is structured as follows. Section 2 discusses related technologies for describing and searching Web services. Section 3 proposes the SWSD framework for Web Services Discovery based on keywords. Our implementation of the framework is described in Sect. 4. Section 5 presents an evaluation of different matching algorithms between the user input and a Web service description. Last, Sect. 6 concludes the paper and discusses future work.

2 Related Work

Current approaches for Web Service Discovery (WSD) can be divided into two types. One approach for Web service discovery is based on clustering operation parameters, while the other approach uses rich semantics for Web service discov-

ery. These two different approaches are explained below by describing existing or proposed Web service discovery systems.

One approach for Web service discovery is by searching for similarities among different Web Service Definition Language (WSDL) service descriptions. In this way, similar operations and services can be discovered based on operation parameters, which enables searching for substitutable and composable Web services [8]. With this approach, the semantics of the Web service operations can be extracted and used for discovery purposes.

A Web service search engine that uses clustering of operation parameters is Woogle [7]. It is designed to search for similar Web service operations and composable Web service operations. Woogle computes automatically the underlying semantics of WSDL descriptions and uses these to match operations. The semantics are solely defined based on the operation parameters. 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.

Seekda! [16] also uses clustering of operation parameters for discovery of Web services. It also extracts semantics from the WSDL files, which enables runtime exchange of similar services and composition of services. Seekda! is part of a bigger system called Service-Finder [4]. Service-Finder is a platform for service discovery where information about services is gathered from different sources such as Web pages, blogs, and Web 2.0 services. The information is automatically added to a semantic model using automatic service annotation, realizing flexible discovery of services. Service-Finder uses its own semantics for discovery and composition, therefore does not taking into account predefined semantics.

GODO [10] does not search for similar Web services, but uses a Goal-Driven approach. It consists of a repository with Web Service Modeling Ontology (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 an existing WSMO Goal will be searched based on those keywords. The WSMO Goal with the highest match will be sent to WSMX [6], 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.

The framework proposed in this paper aims to address a multitude of semantic description languages using a novel approach that combines the ontology structure with Web services similarities using natural language processing techniques for the automatic discovery of Web services. From this point of view it can be considered as a hybrid of the previously introduced approaches able to deal with a larger category of (semantic) Web services. In addition, our framework supports next to exact matching also approximate matching, improving the recall when perfect matching is not possible.

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 NLP 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 the logic-based semantics defined in the Web service descriptions, but uses the definitions of concepts stated in imported ontologies. By making use of these definitions, the framework can establish a broader search field by also employing related concepts from the ontologies to identify the context in which the Web service is operating.

The SWSD framework assumes that there is a set of Web services described using semantic languages such as WSMO [1], WSMO-Lite [19] or OWL-S [13]. 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 matching degree with the user search.

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 and extracting names and non-functional descriptions of used concepts. WSD determines the senses of a set of words present in the previously extracted information. During Match Making the similarity between the different sets of senses is determined, which is subsequently used for ranking the Web services.

3.1 Semantic Web Service Reader

In order to enable a search engine to look through Web service descriptions written in different languages, several different Web service description readers are required, i.e., one for each language. A Semantic Web service reader must be able to extract various elements out of a Web service description and its used ontologies. Names and non-functional descriptions of elements such as the capabilities, conditions, and effects of the Web service help in understanding the context of the Web service, i.e., they can foster establishing the right context. The non-functional descriptions are written in natural language and thus contain a human description of the specified element. Before extracting words from a Web service description, this description has to be parsed using language-specific parsers. Subsequently, word splitting needs to be performed based on case transition, after which each word is tagged with the right Part-of-Speech (POS).

3.2 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. Because words can have multiple meanings disambiguation is needed, resulting 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 [15] to get the senses out of a set of words using a semantic lexicon (e.g., WordNet [14]). The algorithm disambiguates a word based on a previously disambiguated set of words and their related senses. Per sense of the word, a similarity with the senses from the context 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 and another iteration is performed. This process continues until there are no ambiguous words left.

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 pair-wise context sense similarities (after disambiguating all words) is used for the context initialization.

Because studies have shown that the method of Jiang and Conrath [12] performs better than other semantic distance measures [2], this method is chosen to compute the semantic distance between two senses. Using this method, given two senses, a number between 0 and 1 is obtained, stating the similarity between the two senses. If this number is high, then the two senses given are close to each other and therefore are similar.

3.3 Sense Matching

After disambiguating each word gathered from the user input or a Semantic Web service description, we have obtained several different sets of senses. The framework assumes each word in the user query is 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 better 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.

3.3.1 Level Matching

Besides matching only disambiguated senses, words that do not appear in the used semantic lexicon should also be taken into account. These words can represent important names or concepts for the discovery of Web services and hence must also be used in the matching process. 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 senses (mss_w). Because the Web service description, presented in the next section, 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, as shown in (1). The weights (w_i) are established by means of experiments with different values and must sum up to 1 in order to make sure that the final similarity between the user query and a Web service description ranges between 0 and 1.

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

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 (shown in (3)) and one for the matching of words that could not be disambiguated (shown in (4)). 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 average (shown in (2)). These weights are, as with the final similarity, established by means of experiments with different rates and must sum up to 1.

$$levelSim(ss_u, ss_w, ws_u, ws_w) = w_{sense} \times senseSim(ss_u, ss_w) + w_{word} \times wordSim(ws_u, ws_w) \quad (2)$$

3.3.2 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:

$$senseSim(ss_u, ss_w) = \frac{|ss_u \bigcap ss_w|}{|ss_u \bigcup ss_w|} . \quad (3)$$

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 (not present in the semantic lexicon):

$$wordSim(ws_u, ws_w) = \frac{|ws_u \bigcap ws_w|}{|ws_u \bigcup ws_w|} \; . \tag{4}$$

3.3.3 Similarity Matching

To overcome the fact that for calculating similarity values only exact 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, their similarity value will approach 1. This approach allows more flexible matching between different items than previously considered.

For calculating a similarity between two sets of senses, the same similarity function as in WSD is applied. Equation (5) 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 (s_w) 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 (s_u) is added to that to provide a symmetric match.

$$\begin{aligned} senseSim(ss_u, ss_w) = & \sum_{s_u \in ss_u} \frac{senseScore(s_u, ss_w)}{|ss_u| + |ss_w|} + \\ & \sum_{s_w \in ss_w} \frac{senseScore(s_w, ss_u)}{|ss_u| + |ss_w|} \end{aligned} \tag{5}$$

The similarity between a sense (s_a) and a set of senses (ss_b) is determined by the maximum similarity between the sense and one of the senses (s_b) from the other set. Equation (6) shows this computation.

$$senseScore(s_a, ss_b) = \text{argmax}_{s_b \in ss_b} senseNorm(s_a, s_b) \tag{6}$$

Because the similarity distance method from Jiang and Conrath 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 Equation (7), exact similar senses will result in 1 as resulting similarity and a total mismatch between senses will result in 0:

$$senseNorm(s_a, s_b) = 1 - e^{-sim(s_a, s_b)} \; . \tag{7}$$

For matching the sets of non-disambiguated words, the Levenshtein Distance metric is used. This metric calculates the total number of edit 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 (8). Equation (9) shows how the similarity between a word and a set of words is computed. Finally, (10) 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 updates had to be done to change one word into another word, a value of 0 will be used to indicate a total mismatch.

$$wordSim(ws_u, ws_w) = \sum_{w_u \in ws_u} \frac{wordScore(w_u, ws_w)}{|ws_u| + |ws_w|} + \sum_{w_w \in ws_w} \frac{wordScore(w_w, ws_u)}{|ws_u| + |ws_w|} \tag{8}$$

$$wordScore(w_a, ws_b) = \text{argmax}_{w_b \in ws_b} wordNorm(w_a, w_b) \tag{9}$$

$$wordNorm(w_a, w_b) = 1 - 2 \times \frac{levenshtein(w_a, w_b)}{maxLength(w_a, w_b)} \tag{10}$$

4 Semantic Web Service Discovery Engine

Our implementation of the SWSD approach, the Semantic Web Service Discovery Engine, allows users to search for semantically annotated Web services on an existing repository by defining a set of keywords. It is able to handle Web services that are annotated using the WSMO [1] framework. Based on the modularity of the implementation, the engine can be extended with readers that can parse other Semantic Web service languages.

A WSMO Web service reader and a WSMO Ontology reader have been implemented in Java using the WSMO4J [9] API. After reading the Web service files, the found concepts are used for scanning the ontologies for their descriptions. 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. The engine uses seven different levels of information about the Web service, each of which has a different associated importance for the matching process. Importance is expressed using weights (established after examining multiple Semantic Web service descriptions and experimenting manually with non-normalized weights ranging from 1 to 10), which, after normalization, sum up to 1. The different levels and their associated weights are:

- Non-functional description and name of the Web service, 7/27, (direct relation);
- Non-functional descriptions and names of concepts used by Web service, 5/27, (direct relation);
- Non-functional descriptions of properties of capabilities of the Web service, 4/27, (direct relation);
- 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;
- Non-functional descriptions and names of concepts related via attributes with concepts used by the Web service, 3/27 (indirect relation);
- Non-functional descriptions and names of attributes of concepts used by the Web service, 1/27 (indirect relation).

The names and non-functional descriptions of the entities returned from the readers undergo an additional NLP step: nouns and verbs are extracted from the non-functional descriptions using the Stanford POS tagger [17] and words are split if they consist of case-transitions. Subsequently, our WSD implementation makes use of the WordNet [14] API and also the JWordNetSim [18] API for similarity calculation between two WordNet senses.

5 Evaluation

This section covers the evaluation of different matching algorithms that can be used for Semantic Web service discovery. The algorithms described in Sect. 3 are implemented in the SWSD engine and are evaluated using a set of predefined queries and sets of preferred Web services related to one of the queries. The evaluated matching algorithms are: simple, Jaccard-based, and similarity matching. The simple algorithm uses the Jaccard similarity only for lexical representations of words. It does not make use of NLP and thus it is used as a baseline to check whether the NLP steps add value to the discovery of Semantic Web services. The Jaccard-based and similarity matching algorithms work as described in Sect. 3.

In order to evaluate the performance of the three matching algorithms, we have conducted 61 separate experiments, in which we used 14 Web services semantically annotated using WSMO. 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. The other 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 well the algorithms can discover similar Web services.

Testing with 61 queries and three matching algorithms results in 183 PR graphs. As a thorough analysis of all graphs is cumbersome, PR graphs consisting of average precision values for the recall points are created. This enables the comparison of all

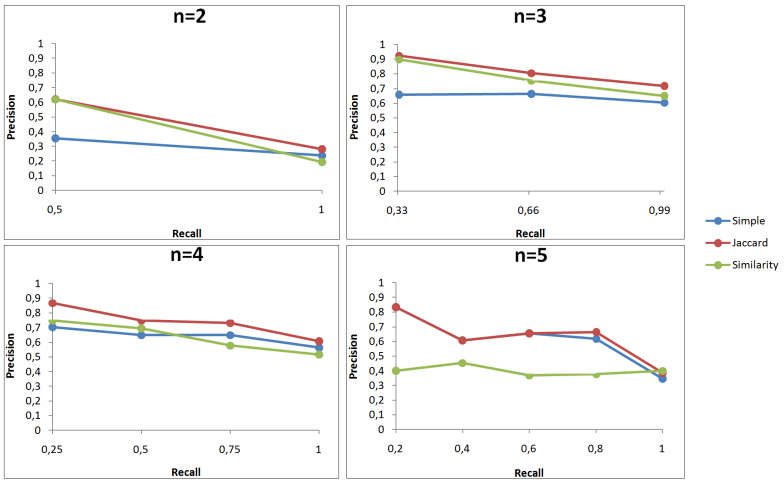

Fig. 1 PR graphs for discovery of exact matching services

the different algorithms at once. However, the testing is done with lists of preferred Web services that can vary in the number of listed services. For testing, lists with two to five 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.

The performances of the different matching algorithms are visualized by eight different PR graphs. The four PR graphs that are displayed in Fig. 1 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 Fig. 2 show the average results for the approximate matching tests.

From the different PR graphs that are shown in Fig. 1, we can make two observations. First, the Jaccard-based algorithm has in most cases a higher precision for the first half of the graph than the simple and the similarity algorithm. 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. Hence, the user has to browse through a number of services – the same for each algorithm – to find the last preferred service. However, as the Jaccard-based algorithm displays a higher precision for a low recall, the Jaccard-based algorithm provides at least some of the preferred Web services in an earlier stage to the user then the other algorithms. 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. 2, we can make the observation that in case of non-exact matching, the similarity algorithm performs overall better for discovery of similar Web services than the Jaccard-based and simple matching algorithms. In most of the cases the precision lines of the similarity algorithm are above the line of the Jaccard-based algorithm.

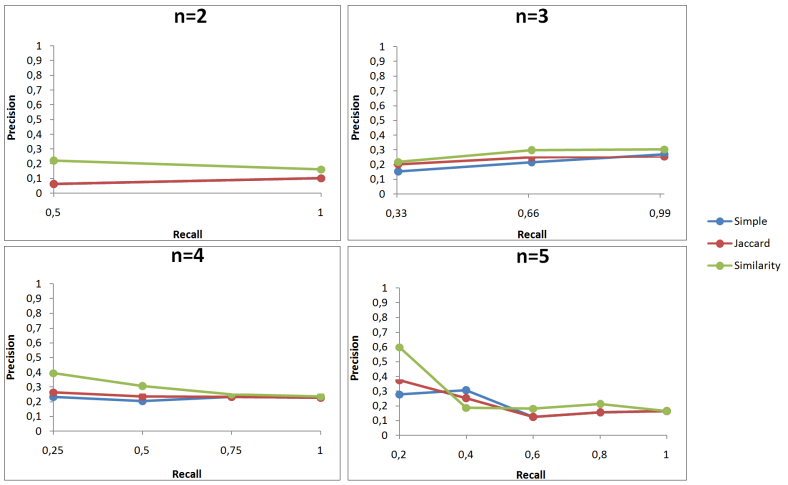

Fig. 2 PR graphs for discovery of similar services

6 Conclusions and Future Work

In order to facilitate managers to compose Web services for business processes, we proposed the SWSD framework, which is a keyword-based discovery process for searching Web services that are described using semantically enriched annotations. It makes an intensive use of natural language processing techniques and a WordNet-based similarity measure for matching keywords written in natural language with semantic Web services described using semantic languages, hereby supporting managers to discover relevant Web services using natural language queries.

Our implementation of the SWSD framework can search for WSMO Web services based on user search words for similar matches. A matching score is computed based on the similarity between the words in the user query and a Web service description. After experimenting with different matching functions, we found that Jaccard matching is performing best for discovering exact matching Web services, while matching using a similarity-based approach gives the best results for finding similar Web services.

As the SWSD engine is currently limited to WSMO descriptions, as 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. Another limitation of the proposed framework is the lack of detailing in Web service contexts, which can be tackled in future work by retrieving additional information from WSDL files. Finally, the weights that are used by the matching algorithms, which are currently established by means of experiments, could be determined by making use of artificial intelligence techniques such as neural networks or Bayesian networks, to optimize the discovery process.

References

1. 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). http://www.w3.org/Submission/WSMO/
2. Budanitsky, A., Hirst, G.: Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In: WordNet and Other Lexical Resources: Applications, Extensions and Customizations. NAACL 2001 Workshop (2001)
3. Bussler, C., Cimpian, E., Fensel, D., Gomez, J.M., Haller, A., Haselwanter, T., Kerrigan, M., Mocan, A., Moran, M., Oren, E., Sapkota, B., Toma, I., Viskova, J., Vitvar, T., Zaremba, M., Zaremba, M.: Web Service Execution Environment (WSMX). W3C Member Submission 3 June 2005 (2005). http://www.w3.org/Submission/WSMX/
4. Cefriel, Seekda!, Ontoprise, University of Sheffield: Service-Finder (2010). http://www.service-finder.eu/
5. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). W3C Note 15 March 2001 (2001). http://www.w3.org/TR/wsdl
6. DERI Galway: Web Service Execution Environment (2010). http://www.wsmx.org/
7. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity Search for Web Services. In: 30th International Conference on Very Large Data Bases (VLDB 2004), vol. 30, pp. 372–383 (2004)
8. Ernst, M.D., Lencevicius, R., Perkins, J.H.: Detection of Web Service Substitutability and Composability. In: International Workshop on Web Services — Modeling and Testing (WS-MaTe 2006), pp. 123–135 (2006)
9. EU IST, FIT-IT: WSMO4J API (2010). http://wsmo4j.sourceforge.net/
10. Gomez, J.M., Rico, M., Garcia-Sanchez, F., Bejar, R.M., Bussler, C.: GODO: Goal driven orchestration for Semantic Web Services. In: 1st Workshop on Web Services Modeling Ontology Implementations (WIW 1004), vol. 113. CEUR Workshop Proceedings (2004)
11. Hikimpour, F., Sell, D., Cabral, L., Domingue, J., Motta, E.: Semantic Web Service Composition in IRS-III: The Structured Approach. In: 7th IEEE International Conference on E-Commerce Technology (CEC 2005), pp. 484–487. IEEE Computer Society (2005)
12. Jiang, J., Conrath, D.: Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In: International Conference Research on Computational Linguistics (ROCLING X), pp. 19–33 (1997)
13. 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). http://www.w3.org/Submission/OWL-S/
14. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: Wordnet: An on-line lexical database. International Journal of Lexicography **3**(4), 235–244 (1990)
15. Navigli, R., Velardi, P.: Structural Semantic Interconnections: a Knowledge-Based Approach to Word Sense Disambiguation. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, pp. 1075–1086. IEEE Computer Society (2005)
16. Semantic Technology Institute: Seekda! (2009). http://seekda.com/
17. The Stanford Natural Language Processing Group: Stanford Log-linear Part-Of-Speech Tagger (2009). http://nlp.stanford.edu/software/tagger.shtml
18. The University of Sheffield: Pure Java WordNet Similarity Library (2010). http://nlp.shef.ac.uk/result/software.html
19. Vitvar, T., Kopecky, J., Fensel, D.: WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. In: 5th IEEE European Conference on Web Services (ECOWS 2007), pp. 77–86. IEEE Computer Society (2007)