A Comparison of Web Service Interface Similarity Measures

Natallia KOKASH

Department of Information and Communication Technology, University of Trento, Via Sommarive, 14, 38050 Trento, Italy, email: kokash@dit.unitn.it

Abstract. Web service technology allows access to advertised services despite of their location and implementation platform. However, considerable differences on structural, semantical and technical levels along with the growing number of available web services makes their discovery a significant challenge. Keyword-based matchmaking methods can help users to locate quickly the set of potentially useful services, but they are insufficient for automatic retrieval. On the other hand, the high cost of formal ontology-based methods alienates service designers from their use in practice. Several information retrieval approaches to assess the similarity of web services have been proposed. In this paper we proceed with such a study. In particular, we examine advantages of using Vector-Space Model, WordNet and semantic similarity metrics for this purpose. A matching algorithm relying on the above techniques is presented and an experimental study to choose the most effective approach is provided.

Keywords. web service, service discovery, service matchmaking

Introduction

During the last years the idea of software composition and refinement as opposed to software building from scratch was elaborated to the platform independent, distributed and based on open standards services paradigm. The state-of-the-art in system integration is defined by the implementation of service-oriented architectures using web service technology. Web services are loosely coupled, distributed and independent software entities, that can be described, published, discovered and invoked via the web infrastructure using a stack of standards such as SOAP, WSDL and UDDI [11]. Potentially, a large amount of compatible services simplifies building of new applications from existing components. However, the problem is very intricate due to the absence of service behavior specifications and control over the service lifecycle. Adaptivity is a highly desired property for service-based systems: troublesome components should be automatically changed to the analogues but troublefree ones. In this context the problem of service discovery acquires a significant importance. Garofalakis et al. [5] provide a survey of different perspectives in this area. Discovery can be carried out by developers at design-time or by self-assembling applications at either design or run time.

These processes are referred to as *manual* and *automated discovery*. Under *manual* discovery, a requester-human searches for a service description that meets the desired criteria. Under *automated* discovery, a requester-agent performs and evaluates this task.

State-of-the-art on automated and semi-automated web service discovery consists of many sound proposals. Simple keyword-based service search is traded against formal methods that require manual annotation of service specifications with semantic information. The latter ones do not fully bring the issue to a close and spawn additional problems such as multiple ontology mapping. As an effort to increase precision of web service discovery without involving any additional level of semantic markup, several approaches based on Information Retrieval (IR) techniques have been proposed [7] [16] [17] [18]. All of them report enhances in precision of automated service matchmaking. In this paper we provide a comparative analysis of the ideas underlying the nominated solutions to locate the most promising strategy. Further, we provide an implementation of a matching algorithm that combines similarity scores by searching the maximum-score assignment between different specification elements. WSDL specifications contain several elements, some of which can be very similar whereas another can be completely different. This presumes combination of lexical and structural information matchings.

The paper is organized as follows. In Section 1, we review the related work. In Section 2, web service description formats are discussed. Section 3 introduces similarity assessment techniques used in our approach. Section 4 describes the proposed web service matching algorithm. Experimental results are presented in Section 5. Finally, Section 6 concludes the paper and outlines future work.

1. Related Work

Currently UDDI registries[1] are the dominating technological basis for web service discovery. Alternatively, ebXML registries[2] can be used to advertise services available on the web. They allow storing actual WSDL specifications in a repository. As a consequence, such ability as retrieval of WSDL using custom ad hoc queries is enabled. The question about the use of registries seems to be irrelevant due to the advantages they bring to the technology. But existing registries are still small and mostly private. The discovery supported by registry APIs is inaccurate as retrieved services may be inadequate due to low precision and low recall. Users may need to examine different registries before they find an appropriate service. Approaches that reduce manual activities in service discovery and allow intelligent agents to identify useful services automatically are required. Below we analyze the existing methods targeted at improving automated service matchmaking.

Generally, information matching can be accomplished on two levels:

- In *syntactic matching* we look for the similarity of data using syntax driven techniques. Usually, the similarity of two concepts is a relation with values between 0 (completely dissimilar) and 1 (completely similar).

---

[1] http://www.uddi.org
[2] http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrim.pdf

- In *semantic matching* the key intuition is the mapping of meanings. There are several semantic relations of two concepts: *equivalence* ($\equiv$), *more general* ($\supseteq$), *less general* ($\subseteq$), *mismatch* ($\perp$) and *overlapping* ($\cap$). Nevertheless, they can be mapped into a relation with values between 0 and 1.

Among the areas closely related to service matching are:

Text document matching. These solutions rely on term frequency analysis and ignore document structure. Among the most popular methods are the Vector-Space Model (VSM), Laten Semantic Indexing (LSI) and Probabilistic Models [2]. However, solely they are insufficient in a web service context.

Semi-structured document matching. The major part of information on the web today is represented in HTML/XML formats. This fact spawned the research aiming to improve IR from semi-structured documents. Methods using plain text queries do not allow users to specify constrains on the document structure. On the other hand, recall of exact matching algorithms used XPath [3] or XQuery [4] is often too low. Kamps et al. [10] noted that structure in XML retrieval is used only as a search hint, but not as a strict requirement.

Software component matching. Software components can be compared with various degrees of accuracy. *Structural similarity* reflects the degree to which the software specifications look alike, i.e., have similar design structures. *Functional similarity* reflects the degree to which the components act alike, i.e., capture similar functional properties [9]. Functional similarity assessment methods rely on matching of pre/post-conditions, which normally are not available for web services. There is an ongoing research to support web service discovery by checking behavioral compatibility (e.g. [8]).

Schema matching. Schema matching methods [12] can be based on linguistic and structural analysis, domain knowledge and previous matching experience. However, the application of schema matching approaches is impeded by the fact that the existing works have mostly been done in the context of particular application domain. In addition, service specifications have much plainer structure than schemas.

In IR approaches to service discovery a query consists of keywords, which are matched against the stored descriptions in service catalogs. LSI, the prevailing method for small document collections, was applied to capture the semantic associations between service advertisements [13] in a UDDI registry. Bruno et al. [3] experimented with automated classification of WSDL descriptions using support vector machines. Stroulia at al. [16] developed a suite of algorithms for similarity assessment of service specifications. WSDL format does not provide any special semantic information but it contains the *documentation* tag with service documentation and elements with natural language descriptions of operations and data types. Identifiers of messages and operations are meaningful, XML syntax allows to capture the domain specific relations. The WordNet database was applied for semantic analysis. According to those experimental results, the methods are nei-

---

[3]http://www.w3.org/TR/xpath
[4]http://www.w3.org/TR/xquery

ther precise nor robust. The main drawback, in our opinion, is that poor heuristics in assigning weights for term similarity were used. Dong et al. [7] present a search engine Woogle focused on retrieval of WSDL operations. Their method is based on term associations analysis. The underlying idea can be expressed by the heuristic that parameters tend to reflect the same concept if they often occur together. The above approaches do not consider data types in a proper way. Carman and Serafini [15] designed an algorithm for semantic matching of complex types. Structure information is used to infer *equal*, *more general* or *less general* relations between type schemas. In [17] web service similarity is defined using a WordNet-based distance metric. Zhuang et al. [18] apply a similar approach. The future directions outlined in the papers include automated preprocessing of WSDL files with complex names handling and structural information analysis, provided in our approach. We also propose a new method to join structural, syntactic and semantic similarities of different elements in a single-number measure. Further, we compare matching algorithms with three different kernel functions.

2. Web Service Specification Formats

Table 1 shows an example of two compatible WSDL definitions. Both specifications represent web search services: *GoogleSearch* and *WolframSearch*. If a client asks some service registry for service *GoogleSearch* which is not published there, *WolframSearch* can be returned instead. Then, if both services are advertised in the same registry they should be classified in the same group to simplify service location. If one of the services fails, another one can be invoked instead to satisfy the user request. These cases require establishment of exact correspondence between service operations, comparison of input/output parameters and checking of data type compatibility.

We examine five logical concepts of WSDL files that are supposed to contain meaningful information: *services*, *operations*, *messages*, *parts* and *data types*. Semantic information from WSDL file representation is shown in Figure 1. Each element has a *description*, i.e., a vector that contains semantic information about this element extracted from the specification. Data types can consist of several *subelements*. We do not consider explicitly their internal organization. However, names of the higher-level organizational tags (like data type category (*complexType*, *simpleType*, *group*, *element*) or composers (*all*, *sequence*, *choice*, *restriction*, *extension*)) are included into the element description. While matching data types, we do not take into account parameter order constraints since parser implementations often do not observe them. This does not harm well-behaved clients and offers some margin for errors. We rely on "relaxed" structural matching since too strict comparison can significantly reduce recall. For example, for the concepts of *GoogleSearch* web service in Table 1 the corresponding concepts of service *WolframSearch* can be found despite of their rather different organization.

Striving for the automated web service discovery and composition has lead to the idea of annotating services manually with semantic information. Recently appeared *WSDL-S* [1] proposal provides a way to associate semantics with web service specifications. It is assumed that there exist formal semantic models rele-

Table 1. WSDL specifications of two web services

GoogleSearch:
<message name="doGoogleSearchResponse">
  <part name="return" type="GoogleSearchResult"/ >
</message>
...
<complexType name="GoogleSearchResult">
  <all>
    <element name="searchComments" type="string"/ >
    <element name="estimatedTotalResultsCount" type="int"/ >
    <element name="resultElements" type="ResultElementArray"/ >
    ...
  < /all>
< /complexType>
WolframSearch:
<message name="WolframSearchResponse">
  <part element="WolframSearchReturn"/ >
</message>
...
<element name="WolframSearchReturn">
  <complexType>
    <sequence>
      <element name="Result" type="WolframSearchResult"/ >
    < /sequence>
  < /complexType>
< /element>
...
<complexType name="WolframSearchResult">
  <sequence>
    <element name="TotalMatches" type="int"/ >
    <element name="Comment" type="string"/ >
    <element name="Matches" type="WolframSearchMatchArray"/ >
    ...
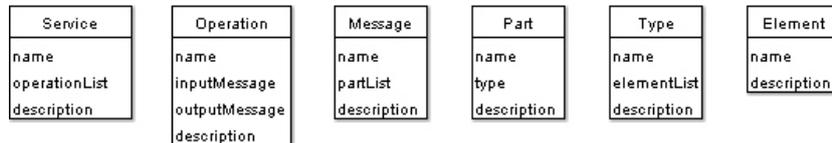 < /sequence>
< /complexType>

Figure 1. WSDL data representation

vant to web services. These models are maintained outside of WSDL documents and are referenced from it via extensibility elements. The specified semantic data include definitions of the *preconditions*, *inputs*, *outputs* and *effects* of service operations. The main advantage over the similar approaches is that developers can annotate web services with their own choice of ontology language. *Ontologies*, i.e.,

explicit and formal specifications of knowledge, are a key enabling technology for the semantic web. They interweave human understanding of symbols with their machine-processability. In respect to web service technology, ontologies can be used for describing service domain-specific capabilities, inputs/outputs, service resources, security parameters, platform characteristics, etc. The main difficulty in practice arises from the fact that the requester and the provider are unlikely to use the same ontology.

Inheritance is important in software development. A bridge between services with and without semantic annotations should be constructed. IR-based service matching algorithms can be extended to become uniform methods that allow matching of WSDL and WSDL-S specifications. In this case some element descriptions will contain references to the corresponding ontology concepts. The latter ones can be compared using specialized matchmaking algorithms.

3. Similarity Assessment

In this section, we cover the rationale of the proposed service matching method. The main idea of the algorithm is a combination of element level lexical similarity matching and structure matching:

1. The goal of *lexical matching* is to calculate the linguistic similarity between concept descriptions.
2. Under *structural matching* we understand the process of similarity assessment between composite concepts (services, operations, messages, parts, data types) that include several subelements.

3.1. Lexical Similarity

Three different linguistic similarity measures were used to compare textual concept descriptions.

Given a set of documents we can measure their similarity using *Term Frequency - Inverse Document Frequency (TF-IDF)* heuristic. Formally it is defined as follows: Let $D = \{d_1, ..., d_n\}$ be a document collection, and for each term w_j let n_{ij} denote the number of occurrences of w_j in d_i. Let also n_j be the number of documents that contain w_j at least once. The TF-IDF weight of w_j in d_i is computed as $x_{ij} = TF_{ij} \cdot IDF_j = \frac{n_{ij}}{|d_i|} \log(\frac{n}{n_j})$, where $|d_i|$ is the total number of words in document d_i. The similarity measure between two documents is defined by the *cosine coefficient*: $cos(x_i, x_k) = \frac{x_i^T x_k}{\sqrt{x_i^T x_i}\sqrt{x_k^T x_k}}$, where $x_i = (x_{i1}, ..., x_{im}), x_k = (x_{k1}, ..., x_{km})$ are vectors of TF-IDF weights corresponding to the documents d_i and d_k, m is the number of different words in the collection. A more detailed description can be found in [2].

WordNet is a lexical database with words organized into synonym sets representing an underlying lexical concept. To address the shortcoming of VSM considering words at the syntactic level only, we expanded the query and WSDL concept descriptions with synonyms from WordNet. After that we compared the obtained word tuples using TF-IDF measure.

Finally, element descriptions were compared using an approach more concerned with the meaning of words. *Semantic similarity* is a measure that reflects the semantic relation between two terms or word senses. Thus, after *tokenization* (splitting an input string into tokens, i.e. determining the word boundaries), *word stemming* (removing common morphological and inflexional endings from words) and *stopwords removing* (eliminating very frequently and very rarely used words), which are common for all three methods, the following steps can be performed to compute semantic similarity of two WSDL concept descriptions:

1. *Part of speech tagging.* Syntactic categories such as *noun, verb, pronoun, preposition, adverb, adjective* should be assigned to words.
2. *Word sense disambiguation.* Each word may have different lexical meanings that are fully understood only in a particular context. Disambiguation is a process of enumerating the likely senses of a word in a ranked order.
3. *Semantic matching of word pairs.* Given input strings X and Y a relative similarity matrix M can be constructed as follows: each element $M[i][j]$ denotes the semantic similarity between the word at position i of X and the word at position j of Y. If a word does not exist in the dictionary the edit-distance similarity and abbreviation dictionaries can be used.
4. *Semantic matching of word tuples.* The problem of capturing semantic similarity between word tuples (sentences) can be modelled as the problem of computing a total weight in a weighted bipartite graph described in the next section. Other metrics can be used as well [6].

3.2. Structural Similarity

Each concept of a query should be confronted with one of the concepts in the documents from the collection. This task can be formulated as *Maximum Weight Bipartite Matching* problem, where the input consists of an undirected graph $G = (V, E)$, where V denotes the set of vertices and E is the set of edges. A *matching M* is a subset of the edges such that no two edges in M share a vertex. The vertices are partitioned into two parts, X and Y. An edge can only join vertices from different parts. Each edge (i, j) has an associated weight w_{ij}. The goal is to find a matching with the maximum total weight. The problem can be solved in polynomial time, for example, using Kuhn's Hungarian method [4].

We applied the above method on different levels of our matching algorithm: (1) to get semantic similarity of two descriptions, (2) to calculate similarity of complex WSDL concepts given similarity scores for their subelements. Weight w_{ij} of each edge is defined as a lexical similarity between elements i and j.

The total weight of the maximum weight assignment depends on the set sizes. There are many strategies to acquire a single-number dimension-independent measure in order to compare sets of matching pairs, the simplest of which is the matching average (see Table 2). Here $|X|$ is the number of entries in the first part, $|Y|$ is the number of entries in the second part, and $|X \cap Y|$ denotes the number of entries that are common to both sets. Finally, $|X \setminus Y|$ defines the number of entries in the first set that are not in the second, and $|Y \setminus X|$ is the number of entries in the second set that are not in the first. Two elements $i \in X$, $j \in Y$ are considered to be similar if $w_{ij} > \gamma$ for some parameter $\gamma \in [0, 1]$.

Table 2. Similarity coefficients

Coefficient name	Formula
Matching average	$2 * Match(X, Y)/(|X| + |Y|)$
Dice coefficient	$2 * |X \cap Y|/(|X| + |Y|)$
Simpson coefficient	$|X \cap Y|/(min(|X|, |Y|))$
Jaccard coefficient	$|X \cap Y|/(|X \cap Y| + |X \setminus Y| + |Y \setminus X|)$
First Kulczynski coefficient	$|X \cap Y|/(|X \setminus Y| + |Y \setminus X|)$
Second Kulczynski coefficient	$(|X \cap Y|/|X| + |X \cap Y|/|Y|)/2$

4. Web Service Matching Algorithm

To obtain WSDL concept descriptions we extracted: (1) sequences of more than one uppercase letters in a row, (2) sequences of an uppercase letter and following lowercase letters, (3) sequences between two non-word symbols from element names, namespaces, data types, documentation and organizational tags. Our experiments show that these simple heuristics work fairly well. For example, from *"tns:GetDNSInfoByWebAddressResponse"* we get the following word tuple: {*tns, get, dns, info, by, web, address, response*}.

After extracting meaningful words from all WSDL specifications we built word indices, where a relative TF-IDF coefficient is assigned to each word. We must note that word stemming (accomplished by the classical Porter stemming algorithm) neither reduced the index dimension nor improved performance on our test bed (described in Section 6). Stopwords removing also brought no effect. Frequently used words in WSDL specifications like *get/set*, *in/out*, *request/response* may distinguish conceptually different elements (e.g., *GetDNSInfoByWebAddressSoapIn* and *GetDNSInfoByWebAddressSoapOut*). To reduce dimension of word vectors to be compared we used three separate word indices: the first index for data types, the second one for operations, messages and parts and the third one for service descriptions.

The information extracted from WSDL specifications is short and rather different from natural language sentences. A clear semantic context is missing in the concept descriptions collected from several technical XML tags. Due to this reason, word sense disambiguation seems to be infeasible. To define a lexical similarity of all possible senses of two terms we used a WordNet-based metric designed by Seco et al. [14]. Its Java implementation is available on `http://wordnet.princeton.edu/links.shtml`.

Our matching algorithm is presented in Table 3. The overall process starts by comparing service descriptions and the operations provided by the services, combined after in a single-number measure. Operation similarity, in their turn, is assessed based on operation descriptions and their input/output messages. To compare message pairs we again evaluate similarity of message descriptions and compare their parts. Since one part with a complex data type or several parts with primitive data types can describe the same concept, we must compare message parts with subelements of complex data types as well.

Function *compareDescriptions*(d_1, d_2) compares two concept descriptions d_1 and d_2 either using TF-IDF heuristic with or without WordNet synonyms or by applying lexical semantic similarity measure. Function *getAssignment*(M) finds the maximum weight assignment considering matrix M as a bipartite graph where

Table 3. WSDL matching algorithm

---

double compareTypes($type_1$, $type_2$)

1    $scoreList \leftarrow$ compareDescriptions($type_1.description$, $type_2.description$)
2   **for** (int $i = 0$; $i < type_1.elementList.length$; i++)
3   **for** (int $j = 0$; $j < type_2.elementList.length$; j++)
4        $d_1 = type_1.elementList[i].description$
5        $d_2 = type_1.elementList[j].description$
6        $M[i][j] =$ compareDescriptions(d_1, d_2)
7    $scoreList \leftarrow$ getAssignment($M[i][j]$)
8   **return** getScore($scoreList$)

---

double compareElementLists($partList$, $elementList$)

1   **for** (int $i = 0$; $i < partList.length$; i++)
2   **for** (int $j = 0$; $j < elementList.length$; j++)
3        $M[i][j] =$ compareDescriptions($partList[i].description$, $elementList[j].description$)
4    $scoreList \leftarrow$ getAssignment($M[i][j]$)
5   **return** getScore($scoreList$)

---

double compareParts($part_1$, $part_2$)

1    $scoreList \leftarrow$ compareDescriptions($part_1.description$, $part_2.description$)
2    $scoreList \leftarrow$ compareTypes($part_1.type$, $part_2.type$)
3   **return** getScore($scoreList$)

---

double compareMessages(msg_1, msg_2)

1    $scoreList \leftarrow$ compareDescriptions($msg_1.description$, $msg_2.description$)
2   **for** (int $i = 0$; $i < msg_1.partList.length$; i++)
3      $elementList_1 \leftarrow msg_1.partList[i].type.elementList$
4   **for** (int $j = 0$; $j < msg_2.partList.length$; j++)
5      $elementList_2 \leftarrow msg_2.partList[j].type.elementList$
6   **for** (int $i = 0$; $i < msg_1.partList.length$; i++)
7   **for** (int $j = 0$; $j < msg_2.partList.length$; j++)
8        $M[i][j] =$ compareParts($msg_1.partList[i]$, $msg_1.partList[j]$)
9    $score_1 =$ getScore(getAssignment($M[i][j]$))
10   $score_2 =$ compareElementLists($msg_1.partList, elementList_2$)
11   $score_3 =$ compareElementLists($msg_2.partList, elementList_1$)
12   **return** max($score_1, score_2, score_3$)

---

double compareOperations(op_1, op_2)

1    $scoreList \leftarrow$ compareDescriptions($op_1.description$, $op_2.description$)
2    $scoreList \leftarrow$ compareMessages($op1.inputMessage$, $op2.inputMessage$)
3    $scoreList \leftarrow$ compareMessages($op1.outputMessage$, $op2.outputMessage$)
4   **return** getScore($scoreList$)

---

double compareServices($service_1$, $service_2$)

1    $scoreList \leftarrow$ compareDescriptions($service_1.description$, $service_2.description$)
2   **for** (int $i = 0$; $i < service_1.operationList.length$; i++)
3   **for** (int $j = 0$; $j < service_2.operationList.length$; j++)
4        $M[i][j] =$ compareOperations($service_1.operationList[i]$, $service_1.operationList[j]$)
5    $scoreList \leftarrow$ getAssignment($M[i][j]$)
6   **return** getScore($scoreList$)

rows represent set X, columns represent set Y and edge weight w_{ij} is equal to $M[i][j]$. The total similarity score can be measured by any of the coefficients in Table 2. We considered the impact of each element within a complex concept to be proportional to its length, i.e., given list $scoreList$ of matching scores of different elements, $getScore(scoreList) = \sum_{i=1}^{n} scoreList[i]/n$, where n is the list length.

5. Experimental Results

To evaluate the effectiveness of different approaches we run the experiments using a collection of web services described in [17]. It consists of 40 XMethods service descriptions from five categories: *ZIP* code finder, *Weather* information finder, *DNA* information searcher, *Currency* rate converter and *SMS* sender. In Table 4, the collection characteristics and preprocessing time performance are shown.

Table 4. Preprocessing performance

Services	Operations	Messages	Parts	Types
40	628	837	1071	410
Parsing time (sec): 37		**Indexing time (sec): 2**		

Since we did not use any additional information apart from those indicated in WSDL specification (i.e., service documentation and quality parameters), our method can be compared with the interface similarity defined in [17]. Their precision varied from 42 to 62%. The effectiveness of our method was evaluated by calculation of *average precision*, that combines precision, relevance ranking, and overall recall. Formally, it is defined as a sum of the precision at each relevant document in the extracted list divided by the total number of relevant documents in the collection: $avgPrec = \sum_{j=1}^{n} (relevant[j] \sum_{k=1}^{j} relevant[k]/j)/r$, where n is the number of documents, r is the total number of relevant documents for the query, and $relevant[i]$ is 1 if i-th document in the extracted list is relevant and 0 otherwise. Results are shown in Figure 2.
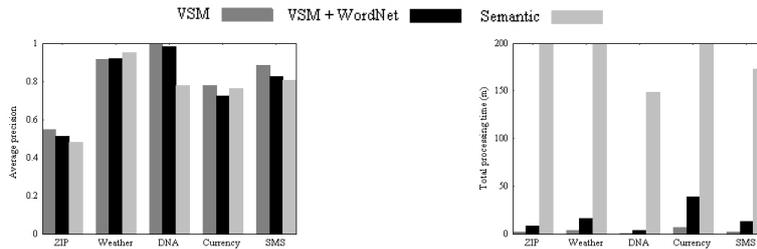

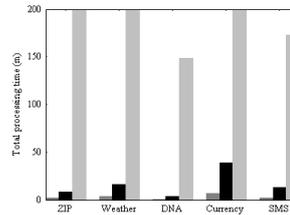**Figure 2.** Average precision **Figure 3.** Processing time

Methods using TF-IDF heuristic and synonyms from WordNet were quite fast, while the usage of lexical semantic similarity required a significant time even for such a small collection. In addition, semantic similarity does not bring any gain in matching precision. This conclusion is confirmed by experiments on the collection described in [16]. We compared 447 services divided into 68 groups.

Because of space restriction the results are not published here and can be found along with algorithm sources on `http://dit.unitn.it/∼kokash/sources`.

An interesting observation is that the groups with better precision in [17] correspond to the groups with worse average precision in our experiments. This may have happened due to different proportions of structure vs. semantic similarity impact on the final similarity score. Enriching element descriptions by synonymous from the WordNet ontology leads to significant increase in index size (see Figure 5). As we can conclude from this statistics, data types are the most informative part of WSDL files. Exhaustive WordNet context essentially differs the text corpus used in concise service descriptions. Yet, WordNet does not provide multitude associations that are required for service matching. Thus, words "currency" and "country" are not recognized as related concepts. Nevertheless, it is clear that given a country name we can get its currency and use a web service accepting currency codes as input to exchange money. Consequently, operations *getRateRequest(country1, country2)* and *conversionRate(fromCurrency, toCurrency)* had significantly lower similarity score than they are expected to have. A repository of verified transformations should be created by clustering of lexically similar terms, terms in complex data types and explicit user experiences.

Table 5. Index size

	Type	Operation	Description	Total
Terms	1634	1336	177	3147
Synonyms	3227	1460	703	5390
Total	4861	2796	880	8537

6. Conclusions and Future Work

We proposed a consistent technique for lexical and structural similarity assessment of web service descriptions, that can be useful in discovery, service version comparison, estimation of efforts to adapt a new service, automated service categorization and blocking in service registries. Our approach can significantly reduce manual operations in these areas provided that the advertised specifications contain feasible information. What we frequently observed in our test collections was an absence of any documentation and/or meaningful identifier names.

Three different functions to measure specification lexical similarity were applied. The classical vector-space model has shown the best performance. Surprisingly, application of semantic similarity metric did not help to improve precision/recall of service interface matching. The reason for this can be in ambiguity of the terms used in service specifications. For some service classes, comparison of WordNet-empowered descriptions brought a slight improvement. However, classical TF-IDF heuristic over-performed the other approaches in most cases. Due to excessive generality of WordNet ontology many false correlations were found.

Particularly lacking from the literature was a comparative analysis of the existing IR techniques applied for web service matchmaking. Our experiments enlighten this situation and pose some relevant issues for future research. The

matching algorithms based on semantic similarity metric should be optimized. More careful study of different approaches is also desirable. We suppose that this work can be improved by using state-of-the-art IR approaches like classification learning or supervised service matching. Also, we are planning to investigate service behavioral compatibility in combination with matching of their structural, syntactic and semantic descriptions.

References

[1] Akkiraju, R., et al.: "Web Service Semantics - WSDL-S", April 2005, http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf.

[2] Baeza–Yates, R., Ribiero–Neto, B.: Modern Information Retrieval. Addison Wesley, 1999.

[3] Bruno, M., Canfora, G. et al.: "An Approach to support Web Service Classification and Annotation", *IEEE International Conference on e-Technology, e-Commerce and e-Service*, 2005.

[4] Galil, Z: "Efficient Algorithms for Finding Maximum Matching in Graphs", ACM Computing Surveys, Vol. 18, No. 1, 1986, pp. 23-38.

[5] Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: "Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?", *International Workshop on Web Engineering*, 2004.

[6] Corley, C., Mihalcea, R., "Measuring the Semantic Similarity of Texts", *Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, pp. 13-18, 2005.

[7] Dong, X.L. et al.: "Similarity Search for Web Services", *Proceedings of VLDB*, 2004.

[8] Hausmann, J.H., Heckel, R., Lohmann, M.: "Model-based Discovery of Web Services", *Proceedings of the IEEE International Conference on Web Services*, 2004.

[9] Jilani., L.L., Desharnais, J.: "Defining and Applying Measures of Distance Between Specifications", IEEE Transactions on Software Engineering, Vol. 27, No. 8, 2001, pp. 673–703.

[10] Kamps, J., Marx, M., Rijke, M., Sigurbjornsson, B.: "Structured Queries in XML Retrieval", *Conference on Information and Knowledge Management*, 2005.

[11] Papazoglou, M. P., Georgakopoulos, D.: "Service-oriented computing", *Communications of the ACM*, Vol. 46, No. 10, 2003, pp. 25–28.

[12] Rahm, E., Bernstein, P.: "A Survey of Approaches to Automatic Schema Matching", *VLDB Journal*, Vol. 10, No. 4, 2001, pp. 334–350.

[13] Sajjanhar, A., Hou, J., Zhang, Y.: "Algorithm for Web Services Matching", *Proceedings of APWeb*, 2004, pp. 665–670.

[14] Seco, N., Veale, T., Hayes, J.: "An Intrinsic Information Content Metric for Semantic Similarity in WordNet", *European Conference on Artificial Intelligence*, 2004.

[15] Carman, M., Serafini, L., Traverso, P.: "Web Service Composition as Planning", *Workshop on Planning for Web Services*, 2003.

[16] Stroulia, E., Wang, Y.: "Structural and Semantic Matching for Accessing Web Service Similarity", *International Journal of Cooperative Information Systems*, Vol. 14, No. 4, 2005, pp. 407-437.

[17] Wu, J., Wu, Z.: "Similarity-based Web Service Matchmaking", *IEEE International Conference on Services Computing*, 2005, pp. 287-294.

[18] Zhuang, Z., Mitra, Pr., Jaiswal, A.: "Corpus-based Web Services Matchmaking", *AAAI Conference*, 2005.