Look Ma, No Hands: Supporting the semantic discovery of services without ontologies

George A. Vouros, Fragkiskos Dimitrokallis, Konstantinos Kotis

AI-Lab, University of the Aegean, Karlovassi, Samos, Greece
{georgev, icsdm06002, kotis}@aegean.gr

Abstract. The work reported in this article aims to the discovery of WSDL specifications that are assessed to match to specific data requirements[1]: Going beyond the syntactic level, we aim at exploiting the human-intended semantics of WSDL specifications At the core of the proposed method lies the Latent Semantic Indexing (LSI) method, which automatically maps data requirements specified in a query to part elements of WSDL input and output messages. We study extensively the performance of the proposed method for different types of experiments' configurations. Experiments have been performed over an extended number of services for various domains, with very encouraging results.

Keywords: Latent Semantic Analysis, Semantic discovery of services

1 Introduction

The infrastructure for Web services is based mostly on standards such as UDDI [4, 16], SOAP [5] and WSDL [17]. WSDL (Web Service Description Language) is an XML-based language for the specification of web services. WSDL is mainly focusing on operational and syntactic details regarding the implementation and execution of Web services. The lack of explicit semantics in WSDL makes services' specifications insufficient to satisfy the requirements for effective Web service 'manipulation' (e.g. discovery, composition, invocation etc.) tasks, forcing the relevant mechanisms to be based on keyword matches. While we may relate different types of semantics to the web services (protocol semantics, execution semantics, non-functional semantics, e.g. security, QoS, and others), we can distinguish two widely recognized types: (a) Data semantics (introducing the semantic signature of services i.e. semantics of input/output messages of service operations), (b) Functional Semantics (function of operations and of the service itself). This paper focuses on data semantics, which is generally accepted to be one of the most critical aspects regarding services' semantic description.

The key technology to the "semantic discovery" (and thus to the "semantic matchmaking") of Web services is ontologies. The key idea here is that the use of

[1] This work is supported partially by the European Commission project Grid4All under grant number FP6, project IST-Grid4All-2006-034567

ontologies for the specification of web services shall support agents to exploit the semantics of services via logic-based inference mechanisms, gaining flexibility to the service manipulation tasks. Three main approaches have been proposed for bringing semantics to web services: OWL-S [1], WSDL-S [2] and WSMO [3]. However, given a) the use of WSDL for the specification of existent services, as well as the use of UDDI registries, b) the lack of commonly agreed domain ontologies, c) the lack of ontologies for specific domains, and d) the high cost of semantically annotating WSDL specifications by engineers, we have been motivated towards the approach proposed in this paper.

Focusing on data semantics, the aim of this paper is to support the semantic matchmaking of WSDL specifications, by exploiting the human-intended semantics of specifications, without the use of ontologies. Towards this target, our approach exploits the human-intended semantics of WSDL input/output messages and parts captured mainly via commentary and descriptive textual information. Such textual information regarding WSDL elements may be "fetched" from the service code, or they may be given by the service providers during service advertisement. This information is being exploited for the computation of the position that WSDL specifications can obtain in a semantic space, where the data semantics of WSDL specifications are expressed by means of a number of latent features. Given a query (this is considered to be a specification for the requested service's data semantics) the objective of the proposed method is to represent this query using the computed latent features and find the registered WSDL specifications that are close enough – and thus similar – to the query, in the semantic space. Doing so, one may consider that these latent features are the concepts of an ontology, which serve as intermediates for the matching of WSDL specifications.

At this point we need to emphasize that although latent features are being used for expressing the data semantics of requested and advertised services, the lack of ontologies prohibit logic-based inferences, leading to the inability of the method to explicitly identify inferred subsumption relations between services' specifications.

The rest of the paper is structured in the following way: Section 2 provides the related work, Section 3 provides a description of the problem and the background technology, and outlines the matchmaking method. Section 4 presents the experiments conducted towards evaluating the proposed method and finally, section 5 concludes the paper.

2 Related Work

Although several approaches introduce the semantic matchmaking of web services (e.g. [6, 7, 8, 9, 10 and 12]), most of them require the use of a *shared* and *well-agreed* ontology: Although this constraint may be relaxed by exploiting ontology mapping techniques, these techniques need to be further enhanced towards their generic deployment. Furthermore, the difficulties mentioned in Section 1 regarding the semantic description of web services need to be tackled.

In this section we emphasize on the approaches that are closely related to the proposed approach, i.e. to approaches that exploit the content of WSDL specifications, and/or exploit textual information related to these specifications.

The OWLS-MX matchmaker [7] performs hybrid semantic matching that complements logic based reasoning with syntactic IR based similarity metrics for services specified in OWL-S. OWLS-MX aims to exploit the implicit semantics of any part of OWL-S service description by representing it as a weighted category-index term vector. Index terms are stemmed lexical items from a shared minimal vocabulary. Concerning the similarity metrics, authors study four different token-based string metrics: the cosine, the loss of information, the extended Jacquard and the Jensen-Shannon information divergence. In contrast to OWLS-MX we go one step "back": we deal with WSDL specifications of services signatures, rather than with their semantic counterparts in OWL-S, aiming to support the content-based discovery of such specifications to a full extend, rather than in combination to logic-based approaches. This is a rather hard problem given the scarceness of the information, the dependency of specifications on the developers' whim, and the small size of the textual descriptions/comments. In addition, we aim at retrieving specifications that match exactly to a query (rather than retrieving specifications that are merely "neighbors" to the query).

Dealing also with WSDL specifications, aiming to address the challenges involved in searching for web services, authors in [11] present Woogle, a web-service search engine. In addition to simple keyword searches, Woogle supports similarity search for web services. The key ingredient of the approach is a refinement of the agglomerative clustering of parameters' terms in the collection of web services into semantically meaningful concepts. By comparing the resulting concepts, this work reports good similarity measures. In contrast to this approach, rather than exploiting parameters' terms co-occurrences, we aim at exploiting the human-intended meaning of WSDL input/output messages' part elements, "describing" them using a number of latent features, and positioning them in a latent space. This approach, in combination to the exploitation of services' textual comments/descriptions, as our experiments show, proves to be very precise.

Another web-service search engine (combining folksonomies and Semantic Web Services technologies) [20] presents a method for semantic indexing and approximate retrieval of Web services. It relies on graph-based indexing in which connected services can be approximately composed, while graph distance represents service relevance. A query interface translates a user's query into a virtual semantic Web service, which in turn is matched against indexed services. The approach is based on the association of classes (ontology conceptualizations or folksonomy tags) with WSDL service specifications: Thus it can be considered only as complimentary to our approach.

In [12], close enough to the approach proposed in this paper, authors discuss a set of WSDL similarity-assessment methods that can be used in conjunction with the current UDDI API to support the service-discovery process. This method utilizes the textual descriptions of the services, the identifiers of WSDL descriptions and the structures of their operations, messages and types to assess the similarity of two WSDL specifications. Given only a textual description of the desired service (as a whole), this approach uses an information-retrieval method to identify and order the most similar service description files. This step assesses the similarity of the query (requested-service description) - extended to include semantically similar words according to WordNet [14] - with the available services. A (potentially partial) specification of the desired service behavior may further refine discovery by a structure-matching computation step, exploiting mainly the lexical similarity of the identifiers. Rather than relying on the lexical appearance of the identifiers, we aim towards expressing their "meaning" by means of latent features, independently of any external resource: Then, matches between service signatures are determined based on semantic matches of WSDL input/output messages and parameters.

3 Service Discovery with Latent Semantics

3.1 Problem specification

This paper deals with the following problem:

"*Given, (a) a repository R of WSDL specifications and (b) a specification of a query Q that specifies the signature of the requested service, provide those services in R that match with Q".*

As already pointed, we deal only with services' signatures specified in WSDL (i.e. with data semantics). More specifically:

- The signature of a service *s* specifies a set of input/output messages. These are denoted <*s,X,i*>, where *s* is the service id, *X* is the type of the message, *input* or *output*, and *i* is the id of the specific message.
- Each message <*s,X,i*> is associated with textual annotations: <*s,i,Y,text*>, where *s* and *i* are as above, *Y* is the type of annotation provided, *description* or *comment*, and *text* is the actual annotation text (possibly *null*). Each message may be associated with more than one annotation.
- Each message <*s,X,i*> has one or more parameters (or parts): <*s,X,i,name,data_type*>, where *s,X,i* are as above, *name* is the name of the parameter, and *data_type* specifies a (atomic or complex) data type.
- Each parameter is associated with textual annotations: <*s,i,name,Y,text*>, where *s*, *i, name*, *Y* and *text* are as specified above. Each parameter may be associated with more than one annotation.

The way WSDL specifications are associated with annotations is thoroughly explained in the following paragraphs.

Each WSDL specification in the registry includes a specification of the corresponding service signature, as specified above. A query *Q* is also a specification of the requested service signature using the above constructs.

The aim of the discovery mechanism is to find the service *s* in the registry that matches the query *Q*. Formally, a service *s* matches a query *Q* , iff

$$\begin{gathered}\forall < s, input, i > \exists < Q, input, j >: match(< s, input, i >, < Q, input, j >) \\ \wedge \\ \forall < Q, output, i > \exists < s, output, j >: match(< Q, output, i >, < s, output, j >)\end{gathered} \quad \textbf{(1)}$$

Given the above formula, for every input parameter of the advertised service there must be a matching input parameter of the required service. Also, for each output parameter of the required service there must be a matching output parameter of the advertised service. We have to notice that alternative definitions, maybe allowing greater matching flexibility, may be provided.

The function *match* determines whether the corresponding parameters match, and provides their similarity degree. In the proposed approach the *match* function is being computed by means of the Latent Semantic Indexing method.

3.2 Background information: Latent Semantic Indexing

The Latent Semantic Indexing (LSI) is a vector space technique originally proposed for information retrieval and indexing [15]. It assumes that there is an underlying latent semantic space that it estimates by means of statistical techniques using an association $N \times M$ matrix of terms-documents. Latent Semantic Analysis (LSA) computes the arrangement of a k-dimensional semantic space to reflect the major associative patterns in the data. This is done by deriving a set of k uncorrelated indexing factors (latent features). As already pointed out in the introduction, these factors may be thought of as artificial concepts whose lexicalization is not important.

Given these factors, each term and document is represented by a vector of values, indicating its strength of association with each of these underlying concepts. In other words, the meaning of each term and document is expressed by k factor values, or equivalently, by the location of a vector in the k-space defined by the factors. Then, a document is the (weighted) sum of its component term vectors. The similarity between two documents is computed by means of the dot product between the corresponding representation vectors.

Concerning our problem, each document corresponds to each input/output message or a message part, and each term is a distinct word in any of these “documents” (as it will be explained, these *pseudo*-documents are being constructed by means of annotations and message/part names).

For the computation of the k factors LSI employs a two-mode factor analysis by decomposing the original association matrix into three other matrices of a very similar form. This is done by a process called “Singular Value Decomposition (SVD)”. This results in a breakdown of the original term-document relationships into linearly independent factors. Some of these factors are not significant and are ignored. The resulting k factors specify the dimensionality of the semantic space. By virtue of dimension reduction from the N terms space to the k factors space, where $k < N$, terms that did not actually appear in a document may still end up close to the document, if this is consistent with the major patterns of association in the data.

When one searches an LSI-indexed database of documents, it provides a query (i.e. a pseudo-document), which is a list of terms. The similarity between the query and any document is computed by means of the dot product between the corresponding representation vectors. Doing so, LSI returns a ranked list of documents, according to their similarity to the query.

3.3 The Matchmaking Method

The proposed matchmaking method assumes as input a specification of the desired web service (i.e. the query *Q*) and a repository *R* of registered services' WSDL specifications. All these services are "accompanied" by annotation files that associate descriptive and commentary information to services' specifications. After applying the matchmaking method to the advertised web services, the output is a ranked list of services according to their semantic similarity to the query. Although in our experiments (for implementation convenience, only) queries are being specified in WSDL, the proposed approach does not necessarily require the use of WSDL for the syntax of the query: Given that the queries are transformed in "bags of words" (as required by LSI), the proposed approach can also be used with keyword-based or template-driven querying forms as in Web-services search engines (e.g. Opossum in http://dori.technion.ac.il/, SeekDa in http://seekda.com/).

The proposed matchmaking method combines multiple sources of evidence to determine similarity between the signatures of two web services. In particular it considers the similarity between input and output messages, and between their parameters (i.e. input parts and output parts, respectively).

More precisely our approach is divided into the following stages: a) the matching of input messages, b) the matching of output messages, c) the matching of input parts and d) the matching of output parts. The algorithm determines the matching of each of these elements individually. The results are linearly combined to a single similarity measure between WSDL specifications (Figure 2).

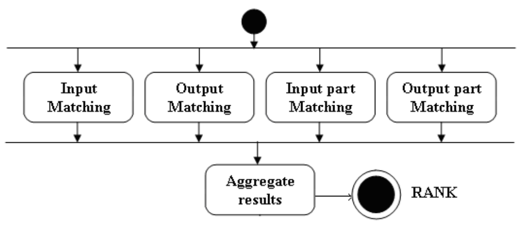


Fig. 2. The combination of stages for the computation of the overall similarity measure.

3.3.1 Annotation Files

The external annotation files (EAF) provide "slots" for the textual annotation of WSDL elements: "Comments" and "description" slots for the service itself, for each service's interface, operation and input/output messages, and for each of the corresponding parameters. It also provides support for specifying mappings between WSDL elements and ontologies. The EAF is an XML file aligned with the WSDL specification via XPATH expressions. Currently, comments and descriptions are considered to aggregate any type of textual information.

Although we plan to incorporate SAWSDL [18] into our framework, we do not commit to the use of SAWSDL at this stage, emphasizing mostly on the use of textual descriptions/comments for WSDL elements.

3.3.2 Input/Output Messages' similarity

Let us consider a message <*s,X,i*>, where *s* is the service *id*, *X* is the type of the message (input or output) and *i* is the id of the specific message; as well as its associated textual annotations <*s,i,Y,text*>, where *Y* is the type of annotation provided (description or comment). We construct a bag of words including all words comprising the message name and all words in the texts that annotate this specific message: This constitutes the pseudo-document corresponding to the message. Each word may be considered as a single term, or as a set of words (in case it is being constructed by concatenating a number of simple terms, e.g. getinputdate_of_arrival): We deal with both cases in our experiments. To improve the precision of our method we eliminate words with little substantive meaning, i.e. stop-words, and we consider only words that appear more than a certain number of times in the specific bag.

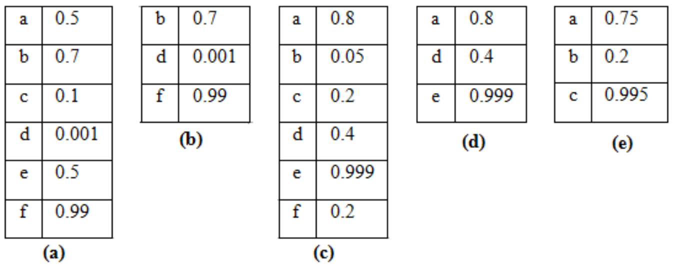
We compute the semantic similarity between each input (output) message of the query *Q* and the input (respectively, output) messages of all the web services in the registry by means of the LSI method: The method computes the semantic space by building the association matrix, associating words occurring in the web services' inputs (or outputs) and in their annotations, with the input (respectively, output) messages' pseudo-documents.

Similarly, each query to LSI is being constructed by means of the words in the corresponding messages of the query specification *Q*: In other words, for each message in *Q,* the proposed method constructs a separate LSI query. For each input/output message of the query *Q*, LSI returns a matrix of registered services' messages, together with a matching degree: Since each input/output message of *Q* can match only with an input/output message of each *s* in the registry, the proposed method keeps the highest ranked message per service *s* (i.e. the most similar one). For each registered service *s* whose input/output messages match with the messages of *Q*, a ranking (matching) degree d_s is computed by averaging the similarities of all input and output messages.

3.3.3 Input/Output part semantic similarity

Let us consider a parameter (message part) <*s,X,i,name,data_type*>, of a message *i* of a service *s*, where *name* is the name of the parameter, and *data_type* specifies its data type. This parameter is associated with textual annotations of the form <*s,i,name,Y,text*>, where *Y* denotes the type of annotation and *text* the annotation text. In this case, we construct a bag of words including all words in name and all words in each text annotating this specific parameter: This constitutes the pseudo-document corresponding to this parameter. As in messages, each word may be considered as a single term, or as a set of words (in case it is being constructed by concatenating a number of simple terms): We deal with both cases in our experiments. The pre-processing stage of eliminating stop-words, and words that do not occur many times in the specific bag, applies here as well.

We compute the semantic similarity between each input (output) parameter of the query *Q* and the input (respectively, output) parameters of all the web services in the registry by means of the LSI method: The method computes the semantic space by building the association matrix, associating words that occur in the web services' input (output) parameters and in their annotations, with the pseudo-documents corresponding to the parameters. Similarly, each query to LSI is being constructed by means of the words corresponding to an input/output parameter of the query *Q*.



a	0.5
b	0.7
c	0.1
d	0.001
e	0.5
f	0.99

(a)

b	0.7
d	0.001
f	0.99

(b)

a	0.8
b	0.05
c	0.2
d	0.4
e	0.999
f	0.2

(c)

a	0.8
d	0.4
e	0.999

(d)

a	0.75
b	0.2
c	0.995

(e)

Fig. 3. Ranking services based on input parts' similarities

For each input/output parameter of the query *Q*, LSI returns a matrix of registered services' parameters, together with a matching degree: Since each input/output parameter of *Q* can match only with an input/output parameter of each *s* in the registry, the proposed method keeps the highest ranked parameter per service *s*. For each registered service *s* whose input/output parameters match with the messages of *Q*, a ranking degree d_s is computed by averaging the matching degree of all input and output parameters.

Since the method of ranking registered services using input/output messages, as well as the method of using input/output parameters, is the same, let us exemplify the computations with a simple example concerning input parameters. Suppose we have the following advertised services: i) service s_1 with three input parts: a, b, c, ii) service s_2 with one input part: d, and iii) service s_3 with two input parts: e, f. The service request *Q* has two input parts: x, y. The method uses LSI to compute the similarity between the input part x of *Q* and all input parts of the advertised services, i.e. a, b, c, d, e and f. The results are shown in table (a) of Figure 3. The method keeps for each service only the input part that has the maximum degree of match. The result after this step is shown in table (b) of Figure 3. Then, LSI computes the similarity between the other input part *y* and all the input parts of the advertised services. The results are shown in table (c) of Figure 3. Again, the method keeps for each service only the input part that has the maximum degree of match. The result after this step is shown in table (d) of Figure 3. Finally, these two tables (b) and (d) are summed and the result is divided by the number of input parts of the request, as shown in table (e). This final matrix associates each advertised service with a similarity to the query *Q*, based on the computation of their input parts' similarities.

3.3.4 Input/Output messages' similarity by means of input/output parts similarity

In addition to the above described approach, we have been experimenting with an alternative method for the computation of input/output messages' similarities, so as to enrich the information consulted for the matching of messages: We construct the bag of words for each input/output message not only using the message names with their textual annotations, but adding also the associated input/output parts with their annotations. In particular, we identify the input/output messages of a web service operation together with the corresponding input/output parts of it. Then, the method computes the matching similarity of the advertised services with the query in the same way as we have described in the previous paragraphs.

3.3.5 Combining individual similarities

The result of the matching algorithm is a list of registered services matching to the query *Q*, with their rankings. These ranking degrees result from averaging the degrees computed by each of the individual stages described: a) the matching of input messages, b) the matching of output messages, c) the matching of input parts and d) the matching of output parts.

4 Experiments

To evaluate our approach, we have used service specifications from the OWL-S Service Retrieval Test Collection (OWLS-TC) version 2 [19] as well as an additional set of nine (9) Web services for Network Simulation (NS). From the OWLS-TC collection, we have translated a set of 70 services to WSDL, due to problems faced with the OWL-S-to-WSDL translation tool and due to services' description duplicates concerning the WSDL part elements. We have been experimented with 7 OWLS-TC domains and the additional NS domain. Table 1 summarizes information concerning the characteristics of the services in our repository *R,* i.e. the different domains of the services (column 1), the number of services for each domain (column 2), and the total number of WSDL input/output messages (column 3) and WSDL input/output distinct parts (column 4) for each domain.

Table 1. Information about the experimental domains and services

	Domain	**# Services**	**# WSDL messages**	**# WSDL Parts**
1	Weapon	3	6	7
2	Education	9	9	82
3	Economy	14	14	31
4	Travel	9	9	27
5	Portal	22	8	63
6	Books	8	22	19
7	Medical	5	5	39
8	Network Simulation	9	21	41
	Total	**79**	**94**	**309**

Concerning the WSDL specifications for domains 2 to 6, message part names are mainly composed by a single-word term capitalized and an underscore character as a prefix (e.g. _COUNTRY). For the domain 7, the messages part names are composed by multi-word terms, either separated with an underscore or with no separator, or using a combination of these (e.g. GetPatientMedicalRecords_AuthorizedMedicalRecords). We handle individual, distinct terms of multi-word terms separately, only in cases these are separated by an underscore separator, which is one of the most generic case considered.

WSDL specifications have been manually annotated by human annotators that have adequate knowledge of the related domains and services. They have been advised to carefully choose the annotations in order to indicate as close as possible the intended meaning of the input/output messages and of their parameters. Where possible, annotators have been advised to get feedback from xsd-schema complex types included in the WSDL specifications. Services in the "Network Simulation" domain have been annotated by their developers. More specifically, we can identify 3 different annotation cases that have been applied in the corpus:

- Annotations that are formed by "free text including terms from xsd-schema types and from the WSDL message part name".
- Annotations that are formed by "free text including terms only from the WSDL message part name".
- Annotations that are formed by "a single term". In this case, annotators choose a single term without considering xsd-schema types or message's part names. For instance, for the "wsdl:part name="Capital_City"", the annotator has chosen the single-term-description "*<description> Capital </description>*", which "captures" the intended meaning of the entity "Capital City".

The result of this process is a set of annotations with terms belonging in one of the following three categories: a) terms from xsd-schema types, b) terms from WSDL message part names, or c) terms chosen by human annotators.

Table 2. Information concerning annotations per domain

Domain	**Annotation type**
Travel, Portal	Descriptions: category (b), Comments: category (a)
Books, Education	Descriptions: category (b), Comments: category (a)
Economy	Descriptions: category (b), Comments: category (a)
Weapon	Descriptions: category (b), Comments: category (a)
Medical	Descriptions: category(c), Comments: category (a)
Network Simulation	Descriptions: category (c), Comments: category (c)

The use of free text with terms from the xsd-schema types and/or the WSDL message part names, means that the annotator is allowed to form a natural language sentence using these terms: E.g. "*<description> The service requests accommodation using country information </description>*". Although the use of free text may distract the matching method (due to the incorporation of noisy terms), the freedom that the approach gives to the annotator is important and realistic. In this example of annotation, the annotator has combined terms (e.g. "country") from messages' part names (category b). As another example, in the comment "*<comment> The country name is a string. A country is described with its capital, its currency, and its government </comment>*" the annotator has used terms (terms "capital", "currency", "government") from the xsd-schema type that corresponds to the specific message part name (category a). Table 2 summarizes information concerning annotations per domain. It must be noticed that all annotations contain 5 to 7 "significant" terms; i.e. non stop-words that may drive the computation of elements' intended mappings.

Given a query *Q*, we consider that the repository *R* includes one matching service. For evaluation purposes we measured the *precision* (*p*) of the method (i.e. the percentage of times that the method returns the correct service at the top of the ranked list), as well as the *Top-k precision* (p_k) of the method (i.e. the percentage of times that the correct service is among the top *k* in the ranked list). Specifically, we have measured p_5 and p_{10}.

The evaluation of the approach has been extensively conducted with a large number of variations of the queries, so as to test the robustness of the proposed approach, even in cases where annotations are misleading, or missing. According to these variations, the words that are being used for the construction of the queries for the input/output messages and parts vary significantly.

Specifically, words concerning the query *Q* may be fetched from:

- WSDL input/output message and part names without splitting them into single words (this is considered to be the default case and it is not denoted in the experiments).
- Splitting all words in the bag (this case is denoted by "split")
- WSDL input/output messages' annotations (this case is denoted by "in/out")
- WSDL input/output parts' annotations (this case is denoted by "inP/outP")
- WSDL input/output messages' annotations in conjunction with information from the corresponding input/output parts (this case is denoted by "inWP/outWP")

Combinations of the above cases give the different configurations of the method. For instance, the cases where the method considers only the matching of input/output parts, constructing queries using WSDL input and output parts' annotations together with part names are denoted as LSA-inP-outP. As another example, the cases where the method considers (a) the matching of input/output messages, constructing queries using words from the WSDL input and output messages' annotations in combination with words from input and output parts' annotations (i.e. the last case above), in combination with (b) the matching of messages parameters, constructing queries using WSDL input and output parts' annotations, together with (c) splitting all words, is denoted as LSA-inWP-outWP-inP-outP-split. Conclusively, as shown in the Figures 4, 5, 6, we have run experiments with eight (8) configurations of the proposed method.

For each case, we have considered alternatives for the filtering of words, based on the times (*t*) of words' appearance in the bag. Due to the very low numbers of words' appearances we distinguished two cases: (a) Filtering out words that appear exactly one time ($t>1$) and (b) including all words ($t>=1$).

First, we evaluated the proposed approach by placing each of the 79 WSDL specifications included in the repository *R* as a query. This may be considered to be the best case for our method since each query matches exactly with an advertised service (Figure 4). In addition, we evaluated the method using each of the 79 services in *R* as a query, but without considering their textual annotations. The results of these cases are also presented in Figure 4. As shown, annotations play a significant role towards increasing the precision of our method: This is particularly true for cases with $t>1$. However, when all words are included in the bag of words ($t>=1$), even if generally the results are better when considering annotations (even better from the cases where $t>1$), some configurations fail to achieve better results from the corresponding cases with no annotations: This is due to the incorporation of “noisy” words.

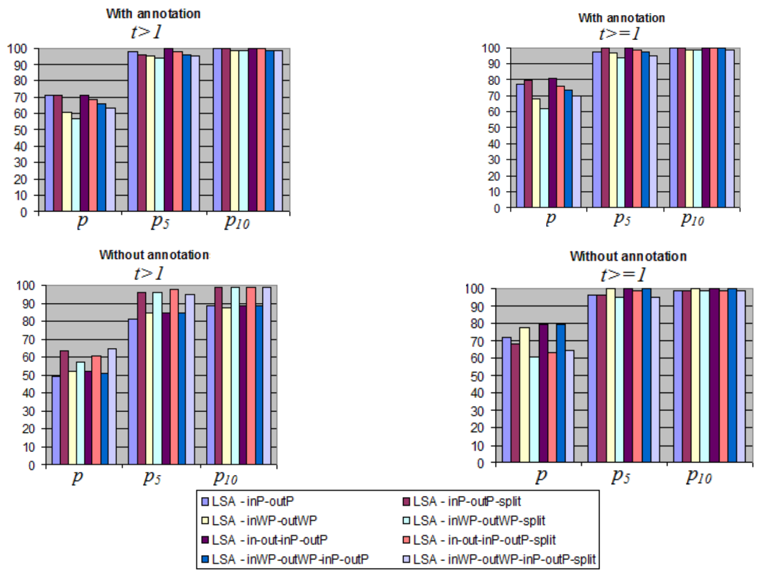


Fig. 4. Cases where Q is in R.

To test the robustness of the proposed method in the presence of noisy words and in cases where queries do not contain many “useful” terms, we have been experimenting with a variety of cases for the queries. Thus, we have conducted experiments by adding extra *new* words (let us call them noise-words) in the bag of words of Q. Noise-words are randomly chosen from the textual annotations of services included in the experimental corpus, given that these services belong to the same domain with the domain of the query service. We have been experimenting with one, three, and five noise-words per query. Figure 5 shows the results of these experiments, together with two more graphs presenting results of the proposed method when WSDL messages and part names are not included in the queries (and without adding noise-words). In addition to the above, we have conducted further experiments with noise-words: This time noise-words replace (one, three, or five) words included in the query. Noise-words are new words and being chosen as in the previous cases. Figure 6 presents the results of these additional experiments, in two different sets of cases: a) Replacement of words with noise-words in queries that include WSDL messages and part names, and b) replacement of words with noise-words in queries that do not include WSDL messages and part names.

In Figures 5 and 6, it can be shown that the precision of the method scales proportionally to the number of noise-words. The best results are achieved in cases where all words are taken into account (t>1). In these cases, when the method considers the input/output messages and their input/output parts separately, together with the part name (split or not) (LSA-in-out-inP-outP-{split}) it achieves the best results, even in cases with increased noise: It manages to present the correct service among the top 5 (respectively, top 10) ranked services with precision greater than

85% (respectively, 93%). It must be pointed that the addition of 5 noise-words with/without replacement, incorporates a major (if not radical) change in the specification of the queries.

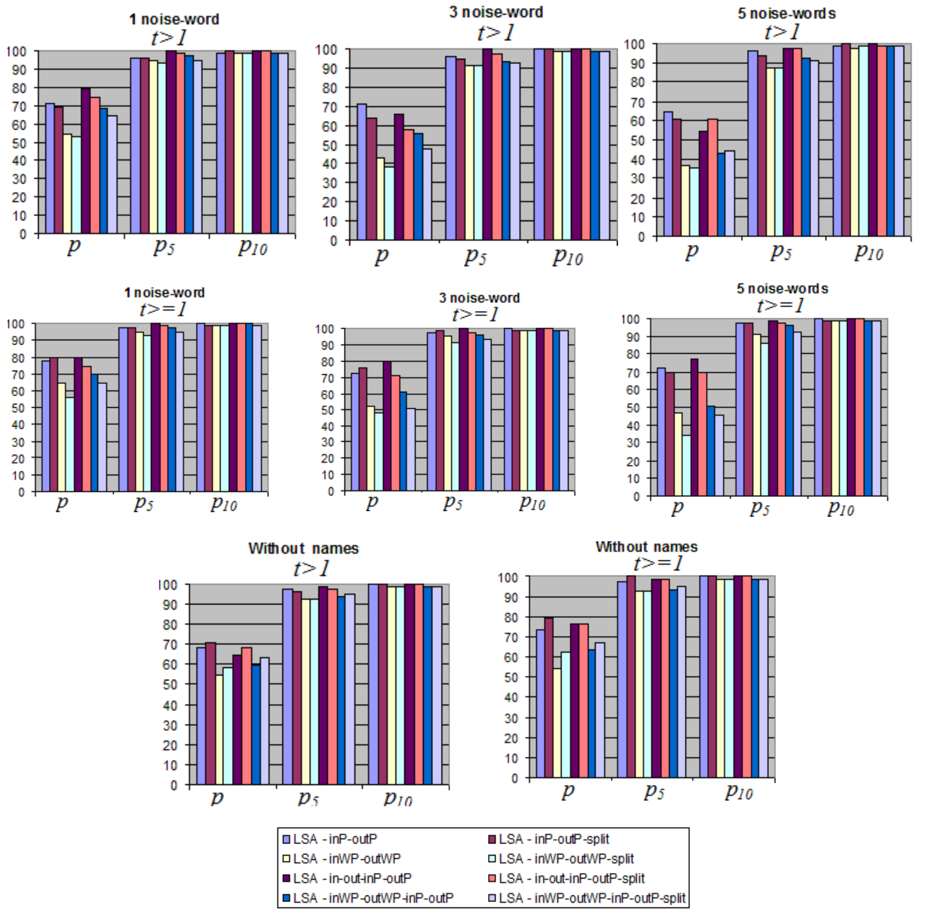


Fig. 5. Results of the proposed method a) adding noise-words in the queries, b) omitting WSDL messages and part names without adding noise-words

5. Concluding Remarks

The work reported in this article aims to the discovery of WSDL specifications that match specific data requirements by computing the intended semantics of part elements of input/output messages.

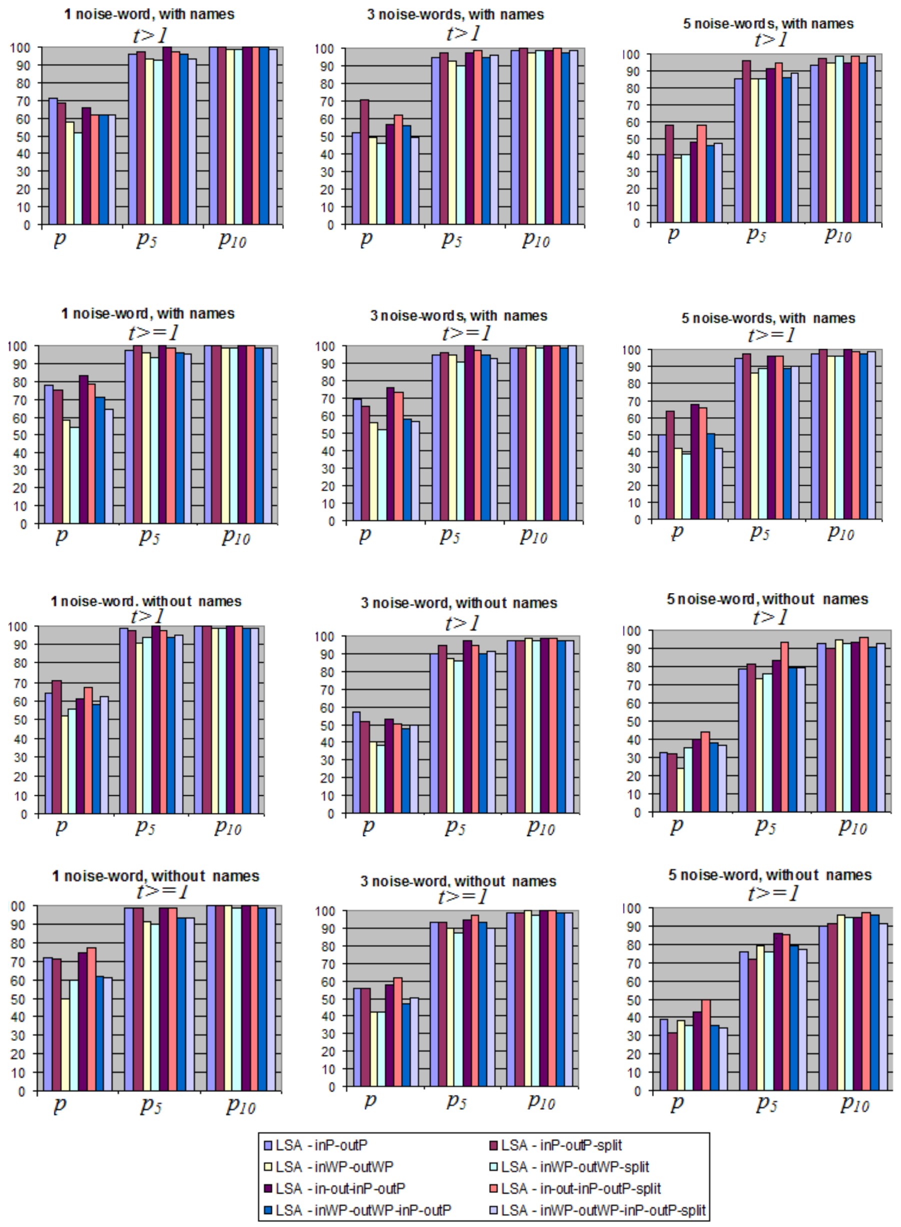


Fig. 6. Results of experiments with replacement of query words with noise-words

Going beyond the syntactic level, we aim at exploiting the human-intended semantics of WSDL specifications captured by means of comments and descriptions been associated with these elements. The basic constituent of the proposed method is the Latent Semantic Indexing (LSI) method, which maps data requirements specified in a WSDL query to part elements of WSDL input and output messages. Preliminary results for different types of experiments' configurations are very encouraging: This "basic" method manages to achieve quite high precision. Further work includes testing the method in a large repository of WSDL specifications, and studying extensively its dependency on the quality of annotations, also in combination with other methods.

References

1. M. Burstein, et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission, Nov. 2004.
2. R. Akkiraju, et al. Web Service Semantics - WSDL-S. W3C Member Submission, Nov. 2005.
3. D. Roman, et al. WSMO - Web Service Modeling Ontology. In DERI Working Draft 14, vol. 1, pp. 77–106. DERI, IOS Press, 2005.
4. OASIS http://www.oasis-open.org/home/index.php
5. Simple Object Access Protocol (SOAP) http://www.w3.org/TR/2000/NOTE-SOAP-20000508
6. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In Proc. of ISWC 2002, pp. 333–347. Springer Verlag, 2002.
7. M. Klusch, B. Fries, M. Khalid, and K. Sycara. "OWL-S: Hybrid OWL-S Service Matchmaking". In Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, 2005.
8. Fernandez, A. Polleres, and S. Ossowski. "Towards Fine-grained Service Matchmaking by Using Concept Similarity". In Proc. of SMR2 2007, vol. 243 of CEUR-WS, pp. 31–45, November 2007.
9. d' Amato, S. Staab, N. Fanizzi, and F. Esposito, "Efficient discovery of services specified in description logics languages", In Proc. of SMR2 2007, vol. 243 of CEUR-WS, .pp. 15-29, Busan, Korea, November 2007.
10. M. C. Jaeger, G. Rojec-Goldmann, G. Muhl, C. Liebetruth, and K. Geihs,"Ranked Matching for Service Descriptions using OWL-S", In Proc. of KiVS, p. 91-102 2005.
11. X. Dong, A. Halevy, J. Madhavan, E. Nemes, J. Zhang, "Similarity Search for Web Services", In Proc. of the 30th VLDB Conf., pp. 372-383 , Toronto, Canada, 2004,
12. D. Skoutas, A. Simitsis, T. Sellis, "A Ranking Mechanism for Semantic Web Service Discovery", in IEEE Congress on Services, pp. 41-48, Salt Lake City, UT, USA, 2007
13. Y. Wang and E. Stroulia, "Semantic Structure Matching for Assessing Web-Service Similarity", In Proceeding of ICSOC 2003, pp. 194-207, Trento, Italy, 2003.
14. Miller, G. (1995). WordNet: A lexical database for English. Communications of the ACM, 38(11) pp. 39-41
15. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science (1990), 41(6), 391-407
16. UDDI technical paper, http://www. uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
17. Web Services Description Language (WSDL) http://www.w3.org/TR/wsdl
18. Semantic Annotations for WSDL and XML Schema (SAWSDL) http://www.w3.org/TR/2007/ REC-sawsdl- 20070828/
19. OWLS-TC, http://projects.semwebcentral.org/projects/ owls-tc/
20. E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A Semantic Approach to Approximate Service Retrieval, to appear at ACM Transactions on Internet Technology, February 2008