URBE: Web service Retrieval based on Similarity Evaluation

Pierluigi Plebani and Barbara Pernici

Abstract—In this work we present URBE (Uddi Registry By Example), a novel approach for Web service retrieval based on the evaluation of similarity between Web service interfaces. Our approach assumes that the Web service interfaces are defined with WSDL (Web Service Description Language) and the algorithm combines the analysis of their structures and the analysis of the terms used inside them. The higher the similarity, the less the differences there are among their interfaces. As a consequence, URBE is useful when we need to find a Web service suitable to replace an existing one that fails. Especially in autonomic systems, this situation is very common since we need to ensure the self-management, the self-configuration, the self-optimization, the self-healing, and the self-protection of the application that is based on the failed Web service. A semantic-oriented variant of the approach is also proposed, where we take advantage of annotations semantically enriching WSDL specifications. SAWSDL (Semantic Annotation for WSDL) is adopted as a language to annotate a WSDL description. The URBE approach has been implemented in a prototype that extends a UDDI (Universal Description, Discovery and Integration) compliant Web service registry.

Index Terms—Web-based services, Information Search and Retrieval, WSDL/SAWSDL, similarity

1 INTRODUCTION

SERVICE Oriented Computing aims to provide a set of methods and tools to support the design and the execution of applications based on Web services. According to this paradigm, at design-time developers identify which activity is to be performed and then try to find and select the Web services closest to such requirements. In some cases such requirements can be figured out only at run-time, and specific modules are involved to perform the discovery and selection activities. This is the typical scenario in which automatic composition techniques and self-healing systems work.

Analyzing the literature, we can identify two main kinds of Web service retrieval approaches with respect to the way in which the Web services are described. On one hand, we have solutions, such as UDDI (Universal Description, Discovery and Integration) and ebXML (Electronic Business using XML) Registries, where Web service description documents are indexed according to keywords or pre-defined taxonomies. Regardless of the way in which the Web service is described, these registries do not exploit the content of the Web service description documents during retrieval. On the other hand, the Semantic Web community proposes solutions, such as OWL-S (Semantic Markup for Web Services) and WSMO (Web Service Modeling Ontology), in which Web services are organized in ontologies. In this case, exploiting the information included in the Web service descriptions, a reasoning process takes place during the Web service matching. Even if the semantic-oriented approaches to match Web services are more effective [1], they do not usually consider the structure of the Web service interfaces and the effort required to semantically describe the Web services is considerable. On the contrary, matching should consider that a Web service description based on WSDL (Web Services Description Language) [2], the most adopted Web service description model, can be easily and often automatically generated starting from the Web service code.

The goal of this work is to propose URBE (Uddi Registry By Example): a novel and effective Web service retrieval algorithm for substitution purpose, based on WSDL, as the model to define the Web service interfaces. Web service substitution can be performed both at design-time and at run-time and might occur, for instance, in case of Web service failures, or in case of Web service is unreachable. This scenario becomes very common in autonomic systems where we need to ensure self-management, self-configuration, self-optimization, self-healing, and self-protection of the applications [3]. To make substitution possible, the substitute Web service has to expose an interface which is equal or richer than the interface of the failed Web service.

Fig. 1 shows the interfaces of two similar Web services. In this case, even if they fulfill the same goal, i.e., currency exchange, the number of available operations, as well as the way in which the input and output parameters are named, is different. Since one of our goal is to evaluate the similarity for substitutability, we need to consider that substituting `CurrencyWS` with

- *P. Plebani is with the Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Via Ponzio 34/5, I-20133, Milan, Italy. E-mail: plebani@elet.polimi.it*
- *B. Pernici is with the Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Via Ponzio 34/5, I-20133, Milan, Italy. E-mail: pernici@elet.polimi.it.*

Manuscript received DD MMM. YYYY; revised DD MMM. YYYY; accepted DD MMM. YYYY; published online DD MMM YYYY.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-YYYY-XX-XXXX. Digital Object Identifier no. 10.1109/TKDE.XXXX.XXXX.

`CurrencyExchangeService` is different than the opposite. Indeed, if we usually invoke the `CurrencyWS`, in case of failure we can start invoking the corresponding operation of the `CurrencyExchangeService` after implementing a mediator able to transform the messages: skipping the value of the license number, i.e. `LicNumber`, obtaining the country name from the related currency symbol, and modifying the name of the invoked operation and the parameters. On the contrary, if we are using `CurrencyExchangeService` and we have to switch to `CurrencyWS`, the new Web service needs an additional parameter that could be obtained by payment of registration fees, i.e., `LicNumber`.

In this work we propose an algorithm able to evaluate the similarity degree between two Web services by comparing the related WSDL descriptions. Such an evaluation reflects how much will be the effort in case a Web service should be substitute with another one in terms of re-coding the client-side. The approach takes into account the relationships between the main elements composing a WSDL description (i.e., portType, operation, message, and part) and, if available, the annotations included in a SAWSDL (Semantic Annotated WSDL) file [4]. In this way, the semantic matching can be improved and, consequently, the performance of our approach improves as well. Mechanisms for actually building a mediation layer between two Web services having similar WSDL are out of the scope of this paper, as well as mechanisms for substituting failed Web services and for composing several Web services to obtain a required interface.

A prototype of URBE, as an enhanced UDDI Registry supporting content-based queries, has been developed. Besides the possibility of finding a substitute for a failed Web service, URBE can be also useful during the WS-BPEL (Web Services Business Process Execution Language) processes top-down design. In traditional design approaches the designer starts to identify the potential partners and then the WS-BPEL process is defined starting from the interfaces of the selected Web services (bottom-up approach). With URBE, the designer can initially focus on the process definition and then the Web services which are able to perform the required invocations can be discovered.

Our approach has been inspired by the literature in the software reusable components [5] and by work done in the Web service community [6] and the Information Retrieval community [7]. With respect to the existing approaches, our algorithm combines both semantic and syntactic aspects of the Web service that can be derived from a WSDL description. The semantic aspects are related to the goal of the Web service and correspond to the names used for the whole service, the operations and the parameters. Instead, the syntactic aspects can state the compliance between the input and output structures and the adopted data types. The algorithm assumes that, as usually occurs, the WSDL specification of the Web service interface is (semi-)automatically generated by a tool starting from a software module, such as a Java class. This implies that the resulting description will probably reflect the naming conventions usually adopted by developers [8].

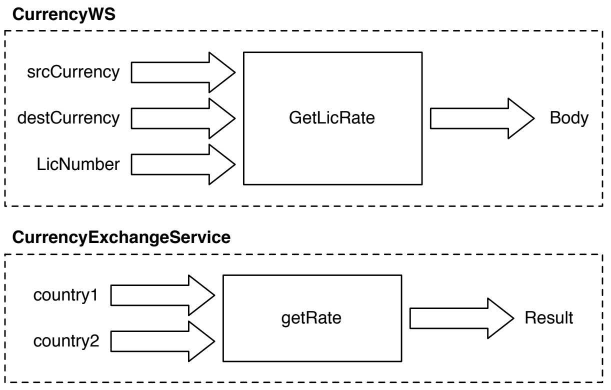


Fig. 1. Example of similar Web services.

The approach relies on a domain specific ontology where terms usually adopted in a given scenario are organized according to semantic relationships such as synonymy (two words have same meaning), antinomy (two word have opposite meanings), homonymy (a word with more meanings). Moreover, a general purpose ontology, e.g., WordNet, supports the matching between terms that are not included in the domain specific ontology.

The paper is organized as follows. Section 2 introduces the notation adopted throughout the paper and the overall view of the algorithm. Section 3 presents the similarity evaluation algorithm in detail. Next, Section 4 presents the similarity algorithm extension for semantically annotated Web services. A description of the prototype of URBE is introduced in Section 5. Section 6 describes how to tune the algorithm and introduces the experimental results, respectively. In Section 7, we compare our work with relevant approaches in the literature. Finally, some concluding remarks and a discussion on possible future extensions are presented.

2 NOTATION AND OVERALL ALGORITHM

We firstly introduce the σ_i *notation* which gives an abstract view of the most important elements constituting a Web service interface according to a WSDL description and that is considered during the similarity evaluation. More precisely:

- $\sigma_i = \langle name, \{op_k\}\rangle$ represents a Web service defined by: a unique name, and a set of K operations.
- $\sigma_i.op_k = \langle name, \{in_l, out_m\}\rangle$ defines an operation in terms of its name, and the L input and M output parameters.
- $\sigma_i.op_k.in_l = \langle name, type\rangle$ defines an input parameter as a pair: name of the parameter and data type. Data type can be either simple (e.g., string, double, int) or composite (e.g. address, date).
- $\sigma_i.op_k.out_m = \langle name, type\rangle$ defines an output parameter in the same way of the input one.

TABLE 1
WSDL entities and corresponding σ_i elements.

WSDL entity	σ_i element
PortType name	σ_i.name
Operation name	$\sigma_i.op_k$.name
Input message part name	$\sigma_i.op_k.in_l$.name
Input message part type	$\sigma_i.op_k.in_l$.type
Output message part name	$\sigma_i.op_k.out_m$.name
Output message part type	$\sigma_i.op_k.out_m$.type

```
...
    <message name="getRateRequest">
        <part name="country1" type="xsd:string"/>
        <part name="country2" type="xsd:string"/>
    </message>
    <message name="getRateResponse">
        <part name="Result" type="xsd:float"/>
    </message>
    <portType name="CurrencyExchangePortType">
        <operation name="getRate">
            <input message="tns:getRateRequest" />
            <output message="tns:getRateResponse" />
        </operation>
    </portType>
...
```

```
$\sigma_a$.name = CurrencyExchangePortType
$\sigma_a$.op$_1$   = { $\sigma_a$.op$_1$.name = getRate,
                  $\sigma_a$.op$_1$.in$_1$ = { $\sigma_a$.op$_1$.in$_1$.name = country1,
                                $\sigma_a$.op$_1$.in$_1$.type = xsd:string },
                  $\sigma_a$.op$_1$.in$_2$ = { $\sigma_a$.op$_1$.in$_2$.name = country2,
                                $\sigma_a$.op$_1$.in$_2$.type = xsd:string },
                  $\sigma_a$.op$_1$.out$_1$ = { $\sigma_a$.op$_1$.out$_1$.name = result,
                                 $\sigma_a$.op$_1$.out$_1$.type = xsd:float }
              }
```

Fig. 2. CurrencyExchangeService.wsdl interface and related σ_i representation.

The correspondences between the σ_i notation and the elements in a WSDL are shown in Table 1. It is worth noting that in our model, given a WSDL, we can have more than one σ_i, since σ_i corresponds to a `portType`. Therefore, in our approach we consider each `portType` as a separate Web service. Fig. 2 shows an example of a σ_i derived from a WSDL document. In this case, only one `portType` exists, so only one σ_i is produced.

With $\Sigma = \{\sigma_p\}$ we represent the Web service registry, in which the interfaces of all the available services are published. Thus, σ_p identifies a generic published Web service interface.

Finally, with σ_q we identify the query, i.e, the interface we are looking for in the registry. As a result, our approach returns Σ_{σ_q} as the set of Web services interfaces included in the registry which are similar to the query. Thus, according to a *query by example* approach, the user defines the characteristics of the desired Web service in the same way as the published ones. Since our approach only focuses on the Web service interface, for the sake of simplicity, hereafter, we use the terms *Web service* and *interface*, interchangeably.

```
input $\sigma_q$ ; //receive the query
input thSim; //set the threshold
let $\Sigma_{\sigma_q} = \emptyset$ ; //initialize the result

for (p=0; p<|$\Sigma$|;p++)
    sim=fSim($\sigma_q$, $\sigma_p$); //similarity evaluation
    if (sim >=thSim) then
       add($\Sigma_{\sigma_q}$,$\sigma_p$);
    end if
end for;
output $\Sigma_{\sigma_q}$
```

Fig. 3. Pseudo-code of the overall algorithm.

Assuming that the function $fSim$ returns the similarity degree between two Web services, as discussed in Section 3, the pseudo-code in Fig. 3 summarizes the main steps performed by URBE.

Generally speaking, once the query is defined, the system compares it with all the Web services published in the registry Σ. If the similarity value, obtained invoking the function $fSim$, is greater than a threshold (th_{fSim}), then the published Web service is added to the result set Σ_{σ_q}.

The function $fSim$ returns a value included in $[0..1]$. The higher the result of $fSim$, the higher the similarity between the two interfaces is. Since one of our main objectives is Web service substitution, a higher value of $fSim$ also means less burden with Web service substitution. In case of WS-BPEL design, if none of the published Web services exposes the same interface according to the partners definition, then the higher the value of $fSim$, the less the complexity in building a mediator.

In particolar, $fSim(\sigma_q, \sigma_p) = 1$ if the two Web services expose the same interface, whereas $fSim(\sigma_q, \sigma_p) = 0$ in case the interfaces are completely different. It is worth noting that a similarity value equals to 1 means that the interfaces of σ_q and σ_p are the same from a syntactical and structural standpoint. So, with our algorithm we are not able to state if given the same input, they return the same output only since they provide exactly the same kind of service from a semantical standpoint.

3 URBE SIMILARITY EVALUATION

The $fSim$ function relies on two main functions: a name similarity function $nameSim$, and a data type similarity function $dataTypeSim$. Generally speaking, the $nameSim$ function compares two names included in the Web service interfaces and evaluates how similar they are, whereas the $dataTypeSim$ function evaluates how similar two data types characterizing the parameters in the Web service interfaces are.

Both of these two functions, as well as the $fSim$ itself, rely on an additional maximization function called $maxSim$. This maximization function exploits the linear programming formulation of the *assignment in bipartite graphs* problem to realize which is the maximum similarity between the elements included in the two sets we are comparing.

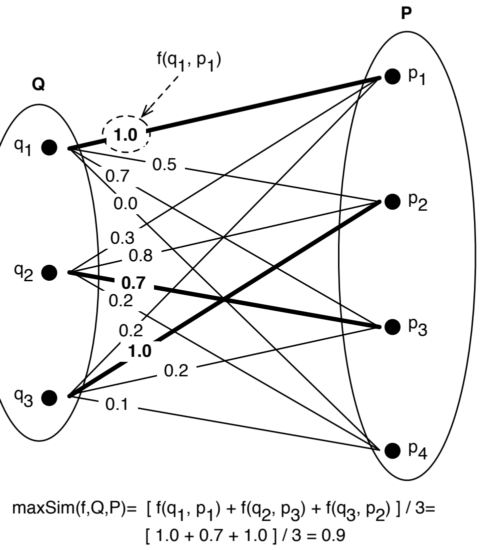


Fig. 4. Graphical representation of the assignment in bipartite graphs problem.

3.1 Maximization function $maxSim$

The maximization function relies on the *assignment in bipartite graphs*. Given a graph $G = (V, E)$, a matching is defined as $M \subseteq E$ so that no two edges in M share a common end vertex. An assignment in a bipartite graph is a matching M so that each node of the graph has an incident edge in M. Let us suppose that the set of vertices are partitioned in two sets Q and P, and that the edges of the graph have an associated weight given by a function $f : (Q, P) \rightarrow [0..1]$. The function $maxSim : (f, Q, P) \rightarrow [0..1]$ returns the maximum weighted assignment, i.e., an assignment so that the average of the weights of the edges is maximum. Fig. 4 shows a graphical representation of the problem, where the bold lines constitute the matching M.

Expressing the assignment in bipartite graphs according to a linear programming model, we have:

$$\begin{aligned} maxSim(f, Q, P) \quad = \quad & \frac{1}{|Q|} \cdot max \sum_{i \in I}^{j \in J} f(q_i, p_j) \cdot x_{i,j} \\ & \sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I \\ & \sum_{i \in I} x_{i,j} \leq 1 \quad \forall j \in J \\ & I = [1..|Q|], \;\; J = [1..|P|] \end{aligned} \tag{1}$$

Applying the assignment in bipartite graphs problem to our context, the set Q represents a query, whereas P is what we compare with the query to evaluate the similarity. Let us assume, for instance, that Q and P are composed of the operations in σ_q and σ_p. $|Q| < |P|$ means that the number of operations in Q is lower than the number of operations available in P; so, for each operation in Q we may find a corresponding operation in P. On the contrary, $|Q| > |P|$ means that we are asking for more operations than are actually available. Since our approach aims to state if σ_q can be replaced with σ_p, then the situation in which $|Q| < |P|$ is, in general, better than the case $|Q| > |P|$. For this reason, we divide the result of the maximization by the cardinality of $|Q|$. So, if $|Q| < |P|$ then $maxSim : (f, Q, P) \rightarrow [0..1]$, whereas if $|P| < |Q|$ then $maxSim : (f, Q, P) \rightarrow [0..\frac{|P|}{|Q|}]$. In this way, the function $maxSim$ is asymmetric, i.e., $maxSim(f, Q, P) \neq maxSim(f, P, Q)$.

Beyond the cardinality of the sets Q and P, the $maxSim$ function is affected by the weights of the edges that are computed according to the function f. In our context, such a function is a similarity function, which given two vertices states how similar they are. As we discuss in detail in the following, the goal of the $maxSim$ function varies with respect to the nature of the sets we are considering. For instance, if we are comparing two parameters, then $maxSim$ considers both the names of the parameters and the data types (exploiting the $nameSim$ and $dataTypeSim$ functions illustrated below); in case of we are comparing two operations, then $maxSim$ relies on the similarity function that takes into account both the name of the operations and all their parameters.

Adopting this approach, we are sure to find the global maximum similarity that can be obtained pairing the elements in the two sets. Alternative approaches are able to find only the local maximum since they scroll the elements in the first set and, after calculating the similarity with all the elements in the second set, they select the one with the maximum similarity. Since every element in one set must be connected, at most, at one element in the other set, such a procedure is able to find only the local maximum since it depends on the order in which the comparisons occur. For instance, considering the example in Fig. 4, q_1 will be paired to p_1 (weight=1.0) but, when analyzing q_2 the maximum weight is with p_2 (weight=0.8). This means that q_3 can no more be paired to p_2 even if the weight is maximum, since this is already matched to q_2. As a consequence, q_3 will be paired to p_3 and the average of the selected weights will be $(1.0 + 0.8 + 0.2)/3 = 0.6$ which is considerably lower than using $maxSim$ where the sum of the weights was $(1.0 + 0.7 + 1.0)/3 = 0.9$

3.2 Names similarity function $nameSim$

The goal of the $nameSim$ function is to compute the similarity between names with respect to the closeness of such names in a given ontology. In particular, assuming that the registry Σ includes Web services related to a given application domain, to compute the similarity among two names we rely on a *domain specific ontology* and a *general purpose ontology*.

The *domain specific ontology* includes terms related to a given application domain. We assume that this ontology can be built by a domain expert also analyzing the terms included in the Web services published in the registry. The *general purpose ontology* includes all the possible terms (at this stage we adopt Wordnet). We decided to rely on both ontologies since the domain specific ontology offers more accuracy in the relationships of the terms, whereas the general purpose one offers wider coverage. This happens because in a general purpose ontology a word may have more that one synonym set (a.k.a. *synset*): a set of one or more synonyms that are interchangeable in some context. On the contrary, we assume that in a domain specific ontology each word has a unique sense with respect to the domain itself. For instance, if we consider the noun *currency*, in WordNet it has two synsets. The first one is about the financial domain, i.e., the metal or paper medium of exchange currently being used; the second one is about a generic meaning, i.e., general acceptance or use. Comparing the term *currency* with the term *money* [1] we can realize that they are strictly related only if we consider the financial domain. On the other hand, if we consider the other synset the relationship is looser. Therefore, in case of general purpose ontologies, it is hard to figure out which is the correct domain to consider then, so we consider the average similarity for each synset.

Due to the nature of the names normally included in an automatically generated WSDL, name similarity can be applied only after a tokenization process which produces the set of terms to be actually compared. In fact, names included in a Web service interface could be like `currencyExchange`, or `userId` which are difficult to find in both our ontologies. On the contrary, in the ontologies we can find the terms composing these names: e.g., *currency*, *exchange*, *user*, *id*. For this reason, we perform the tokenization to decompose a given name in its terms. These terms will be the actual elements compared to obtain the similarity among names.

The tokenization process starts from a generic name n and then it builds the set of terms $\{t_i\}$ composing it, according to a set of rules inspired by common programmers naming conventions. At this stage, our tokenization takes care of case change, the use of underscore and hyphenation, and use of numbers. In Table 2 a list of examples is provided to clarify the rules which the tokenization relies on. The terms resulting from the tokenization are also stemmed. Thus, words such as *sending* or *exchanged* are transformed into their stemmed version: *send*, *exchange*. The stemming process is a well-known process and it is adopted by several Information Retrieval approaches [9].

The set of terms resulting from the tokenization and stemming process is the input of the name similarity function $nameSim$. If we consider two tokenizable names $n_q = \{t_{q,i}\}$ and $n_p = \{t_{p,j}\}$, their similarity evaluation relies on the $maxSim$ function introduced above:

TABLE 2
Tokenization rules examples.

Rule	Original term n	Tokenized version $\{t_i\}$
Case change	currencyExchange	currency, exchange
Case change	SendSMSTo	send, sms, to
Suffix numbers elimination	currency1	currency
Underscore separator	currency_exchange	currency, exchange

$$nameSim(n_q, n_p) = maxSim(termSim, \{t_{q,i}\}, \{t_{p,j}\}) \tag{2}$$

where $\{t_{q,i}\}$ and $\{t_{p,j}\}$ represent the set of terms obtained after the tokenization and stemming process.

Thus, we apply the maximization function when comparing two sets composed of the set of terms obtained after the tokenization process. In this case, the weights between the vertices is given by a function $termSim : (term, term) \rightarrow [0..1]$. In the literature, several approaches are available to state the similarity and the relatedness among terms [10]. These algorithms usually calculate such a similarity relying on the relationships among terms defined in a reference ontology (e.g., *is-a*, *part-of*, *attribute-of*). In our approach, to compute the similarity among terms we adopt the approach proposed by Seco et al. [11] where the authors adapt existing approaches relying on the assumption that concepts with many hyponyms [2] convey less information than concepts that have less hyponyms or any at all (i.e, they are leaves in the ontology).

3.3 Data types similarity function $dataTypeSim$

In a WSDL description, data types are expressed by XSD (XML Schema Definition) specifications. A data type can be *built-in* or *complex* [12]. In the former case, simple data types (e.g., *xsd:string*, *xsd:decimal*, *xsd:dateTime*) as well as derived data types (e.g., *xsd:integer*, *xsd:short*, *xsd:byte*) are included. In the case of complex data types, data type is expressed according to an XSD schema which is included, or imported, in the WSDL specification as a *complexType*: a data type which includes other data types (either built-in or complex).

The function $dataTypeSim : (dt_q, dt_p) \rightarrow [0..1]$ calculates the similarity between two data types defined in the following way:

- if both dt_q and dt_p are built-in XSD data types $dataTypeSim = simpleDTSim$.
- if both dt_q and dt_p are complex XSD data types $dataTypeSim = complexDTSim$.
- if dt_q is a built-in data type and dt_p is a complex data type (or vice-versa) then $dataTypeSim = 0$: we consider them totally different since they have an incomparable data structure.

1. see http://marimba.d.umn.edu/cgi-bin/similarity.cgi

2. A hyponym is a word of more specific meaning than a general term applicable to it, i.e., spoon is a hyponym of cutlery.

TABLE 3
Simple Data Type groups.

Group	Simple and derived XSD Data Types
Integer group	integer, byte, short, long
Real group	float, double, decimal
String group	string, normalizedString
Date group	date, dateTime, duration, gDay, gMonth, gMonthDay, gYear, gYearMonth, time
Boolean group	boolean

TABLE 4
$simpleDTSim$ function.

		dt_q				
	$dataTypeSim$	**Integer**	**Real**	**String**	**Date**	**Boolean**
dt_p	**Integer**	1.0	0.5	0.3	0.1	0.1
	Real	1.0	1.0	0.1	0.0	0.1
	String	0.7	0.7	1.0	0.8	0.3
	Date	0.1	0.0	0.1	1.0	0.0
	Boolean	0.1	0.0	0.1	0.0	1.0

TABLE 5
Referencing scale for evaluating the information loss (intermediate values are admissible as well).

Information loss	Value
data types are totally incompatible	1.0
in some rare case casting does not produce information loss	0.7
information loss happens by casting	0.5
often casting does not produce information loss	0.3
data types are the same	0.0

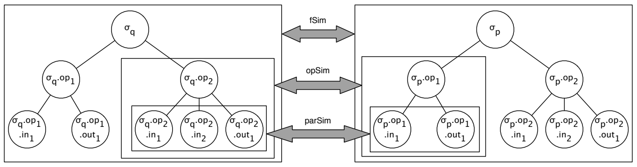


Fig. 5. Tree representation and nested comparison.

In case of built-in data types we take inspiration from [13]: we group the simple and derived XSD data types in five main classes as shown in Table 3. In this way, when the comparison between two built-in data types is needed, then we take into account the membership in these groups. In more detail, Table 4 defines the behavior of function $simpleDTSim$ where the values included in the table is inversely proportional to the information loss that will occur if we apply a casting from dt_q to dt_p. To quantify the information loss we propose the qualitative reference scale shown in Table 5. The similarity between two data types are calculated as the complement of the information loss. For instance, if dt_q belongs to the *integer group* and dt_p to the *real group* then we assume than $simpleDTSim = (1 - 0.0) = 1.0$ since we have no information loss. In the opposite situation instead, $simpleDTSim = (1 - 0.5) = 0.5$ since we can convert a real into an integer but we lose the decimals. Another significant situation is when dt_q is a boolean and dt_p is an integer. In this case, we assume that the information loss is 0.1 and not as 0.0, since it might happen that we can operate a transformation $true = 1$ and $false = 0$; so, $simpleDTSim = (1 - 0.9) = 0.1$. Generally speaking, $simpleDTSim$ is defined independently of the specific Web service and gives an idea of the information loss. The values included in the Table 5 are obtained conducting empirical evaluations.

In case of complex data types, the literature [14], [15] proposes several approaches to compare XSD schema documents. These approaches usually evaluate differently how much the trees representing the data types are different and provide distance metrics. In our case, in order to reduce the complexity of the overall algorithm, we focus our attention more on the semantics of data rather than on their structure. We assume that the goals of the parameters are well defined by the name of the data types. As a consequence, in case of comparison of complex data types, $complexDTSim = nameSim$. Obviously, considering also the data type structure would allow for a more precise evaluation but, as discussed in the following, the comparison among data types will occur several times and we need to mediate between execution time and the accuracy of the approach.

3.4 Web service similarity function $fSim$

The maximization function along with the name and data type similarity functions constitute the building blocks of the overall similarity function $fSim$. More specifically, the *similarity function* $fSim : (\sigma_q, \sigma_p) \rightarrow [0..1]$, given a couple of Web services interfaces σ_q and σ_p, returns a value stating how similar σ_p is to σ_q.

Fig. 5 gives a high level view of $fSim$. According to the σ_i notation, each interface can be represented as a three-level tree: first, we have σ_i representing a `portType`, then the set of operations $\sigma_i.op_k$, and finally, the set of parameters ($\sigma_i.op_k.in_l$, $\sigma_i.op_k.out_m$), representing the parameters of the supported operations. As a consequence, the functions which evaluate the similarity among the whole interfaces ($fSim$), operations ($opSim$), and parameters ($inParSim$ and $outParSim$) are nested in the same way. In detail, $fSim$ returns the similarity between the two Web services relying on the function $opSim$. For each requested operation, $opSim$ identifies the operation in the published Web service which is most similar to relying on $inParSim$ and $outParSim$. Finally, given two sets of requested parameters, $inParSim$ and $outParSim$ find the most similar parameters in the published ones. According to this:

$$\begin{aligned} fSim(\sigma_q, \sigma_p) = {} & wPTNameSim \cdot \\ & \cdot\ nameSim(\sigma_q.name, \sigma_p.name) + \\ & + (1 - wPTNameSim) \cdot \\ & \cdot\ maxSim(opSim, \sigma_q.\{op_{qk}\}, \sigma_p.\{op_{pk}\}), \end{aligned} \quad (3)$$

where:

$$\begin{aligned} & opSim(\sigma_q.op_{qk}, \sigma_p.op_{pk}) = \\ & \quad wOpNameSim \cdot nameSim(\sigma_q.op_{qk}.name, \\ & \qquad \sigma_p.op_{pk}.name) + \\ & \quad +(1 - wOpNameSim) \cdot \\ & \quad \cdot [0.5 \cdot maxSim(inParSim, \sigma_q.op_{qk}.\{in_{ql}\}, \\ & \qquad \sigma_p.op_{pk}.\{in_{pl}\}) + \\ & \quad +0.5 \cdot maxSim(outParSim, \sigma_p.op_{pk}.\{out_{pm}\}, \\ & \qquad \sigma_q.op_{qk}.\{in_{qm}\})] \end{aligned} \tag{4}$$

and

$$\begin{aligned} & inParSim(\sigma_q.op_{qk}.in_{ql}, \sigma_p.op_{pk}.in_{pl}) = \\ & \quad wParNameSim \cdot nameSim(\sigma_q.op_{qk}.in_{ql}.name, \\ & \qquad \sigma_p.op_{pk}.in_{pl}.name) + \\ & \quad +(1 - wParNameSim) \cdot \\ & \quad \cdot datatypeSim(\sigma_q.op_{qk}.in_{ql}.name, \sigma_p.op_{pk}.in_{pl}.name) \end{aligned} \tag{5}$$

$outParSim$ is similarly defined.

It is worth noting that the following properties hold for the similarity function $fSim$:

- $fSim(\sigma_i, \sigma_i) = 1$: a Web service is totally similar to itself so it is fully replaceable;
- in general, $fSim(\sigma_i, \sigma_j) \neq fSim(\sigma_j, \sigma_i)$: the similarity depends on which Web service holds the role of query. This asymmetry derives from the asymmetry of both the maximization and the data type functions.

The parameter $wPTNameSim \in [0..1]$ defines how much the similarity of the name of the `portType`s has more importance than the similarity between the operations that these `portType`s contain in computing the overall similarity. In the same way, at operation level, the parameter $wOpNameSim$ weights the importance between the similarity of the operation names and the similarity of the related parameters in computing the operation similarity. Finally, $wParNameSim$ states how much the similarity of the parameter names has more weight than the analysis of the data types when computing the parameter similarity.

According to the $opSim$ formulation, we assume that the evaluation of similarity between the input and output parameters has the same importance. Especially in the case of Web service substitutability, it might also happen that the output parameters are more important than the input ones. In fact, when a Web service fails and needs to be replaced, the designer should look first of all to Web services which are able to produce the same outputs. This is because the Web service result will represent the input of other Web services. Thus, preserving the output structure the Web service substitution might have a lower impact on the whole system. To obtain this behavior, the weights for $inparSim$ and $outparSim$ should be properly set.

4 SEMANTIC EXTENSION

At this stage, WSDL represents the most widely used way to describe a Web service interface. Even if WSDL provides enough information to establish a connection with a Web service, this specification lacks details on the real goal of the whole Web service and the constituting operations as well. As a consequence, several efforts have been made by the Semantic Web community to improve the Web service description. One of these approaches is SAWSDL [4] which semantically enriches the Web service interface definition by annotations: elements of the WSDL are annotated with concepts organized in a reference ontology. Annotations provide new information which might be useful to state the Web service similarity. As a consequence, in this work, we also propose a semantic-aware variant of URBE. We decided to focus on SAWSDL since it is built on WSDL (the document structure remains the same) and simply adds some annotations which are able to better describe operations and parameters. This new approach affects the σ_i notation as introduced in Section 2. Now, we consider σ_i^s as the semantic version of σ_i which considers the annotations applied to Web service names, operations names, and parameters names:

- $\sigma_i^s = \langle name, ann, \{op_k^s\}\rangle$.
- $\sigma_i^s.op_k^s = \langle name, ann, \{in_l^s, out_m^s\}\rangle$.
- $\sigma_i^s.op_k^s.in_l^s = \langle name, ann, type\rangle$
- $\sigma_i^s.op_k^s.out_m^s = \langle name, ann, type\rangle$.

We assume that the Web service provider is in charge of annotating at design time the resulting description using the terms included in an ontology defined according to the OWL (Web Ontology Language) specification [16]. Annotations can be both classes (i.e., sets that contain individuals) or properties (i.e., binary relations on individuals) and in the ontology both classes and properties can be organized in a superclass-subclass hierarchy [17].

Introducing the annotation provides a more accurate description of the elements included in a WSDL description. Moreover, the most meaningful way in which the concepts are organized in the reference ontology allows for better accuracy during the Web service comparison. Thus, we can exploit these annotations when performing the similarity analysis. On this basis, the similarity function $fSim$ can be modified to take advantage of these new elements. Therefore, we propose $fSim^s$ as the semantic-aware variant of $fSim$:

$$\begin{aligned} fSim^s(\sigma_q^s, \sigma_p^s) = {} & wPTNameSim \cdot \\ & \cdot\ annSim(\sigma_q^s.ann, \sigma_p^s.ann) + \\ & + (1 - wPTNameSim) \cdot \\ & \cdot\ maxSim(opSim^s, \sigma_q^s.\{op_{qk}^s\}, \sigma_p^s.\{op_{pk}^s\}), \end{aligned} \tag{6}$$

```
function annSim(a_q, a_p)
  if (a_q is class) and (a_p is class)
      return pathSim(a_q, a_p)
  elseif (a_q is property) and (a_p is
property)
      return pathSim(a_q, a_p)
  elseif (a_q is class) and (a_p is property)
      return classpropSim(a_q, a_p)
  elseif (a_q is property) and (a_p is class)
      return propclassSim(a_q, a_p)
end function
```

Fig. 6. Pseudo-code of overall algorithm.

$$\begin{aligned} &opSim^s(\sigma_q^s.op_{qk}^s, \sigma_p^s.op_{pk}^s) = \\ &\quad wOpNameSim \cdot annSim(\sigma_q^s.op_{qk}^s.ann, \\ &\qquad\qquad \sigma_p^s.op_{pk}^s.ann) + \\ &\quad +(1 - wOpNameSim) \cdot \\ &\quad \cdot [0.5 \cdot maxSim(inParSim^s, \sigma_q^s.op_{qk}^s.\{in_{ql}^s\}, \\ &\qquad\qquad \sigma_p^s.op_{pk}^s.\{in_{pl}^s\}) + \\ &\quad +0.5 \cdot maxSim(outParSim^s, \sigma_p^s.op_{pk}^s.\{out_{pm}^s\}, \\ &\qquad\qquad \sigma_q^s.op_{qk}^s.\{in_{qm}^s\})] \end{aligned} \tag{7}$$

$$\begin{aligned} &inParSim^s(\sigma_q^s.op_{qk}^s.in_{ql}^s, \sigma_p^s.op_{pk}^s.in_{pl}^s) = \\ &\quad wParNameSim \cdot annSim(\sigma_q^s.op_{qk}^s.in_{ql}^s.ann, \\ &\qquad\qquad \sigma_p^s.op_{pk}^s.in_{pl}^s.ann) + \\ &\quad +(1 - wParNameSim) \cdot \\ &\quad \cdot datatypeSim(\sigma_q^s.op_{qk}^s.in_{ql}^s.ann, \sigma_p^s.op_{pk}^s.in_{pl}^s.ann) \end{aligned} \tag{8}$$

$outParSim^s$ is similarly defined.

With respect to the former $fSim$, instead of comparing the name adopted to define the service elements using the $nameSim$, we now compare the related annotations using the function $annSim$. Since it might happen that some of the operations and parameters of a Web service are not annotated, then we will use $nameSim$ whenever at least one of the compared elements is not annotated. In detail, $annSim : (a_q, a_p) \rightarrow [0..1]$ receives as input two annotations and returns their similarity according to the way in which they are related in the reference ontology. Hereafter, we assume that both a_q and a_p are included in the same ontology, otherwise $annSim$ returns 0. Since the annotations can be classes or properties, as shown in the Fig. 6, the $annSim$ relies on three functions to compute the similarity in all the possible situations: $pathSim$, $classpropSim$, and $propclassSim$.

In case both annotations are classes or both annotations are properties, to compute the similarity between the two annotations we take into account the subsumption path which connects them. Thus:

$$pathSim(a_q, a_p) = \begin{cases} 0, & \text{if no subsumption path exists} \\ \dfrac{1}{(pathlength(a_q, a_p) + 1)}, & \text{otherwise} \end{cases} \tag{9}$$

where $pathlength$ returns the number of hops constituting the longest path (i.e., the worst case) connecting the two classes or properties.

In case a_q is a class and a_p a property, the function $classpropSim$ firstly verifies that the domain of the property corresponds to the class. If so, it means that (i) the annotation in the query, i.e., a_q refers to a class with all its properties, and (ii) the annotation in the published service, a_p , refers only to one of those properties. Thus, we calculate the annotation similarity as follows:

$$annSim(a_q, a_p) = \begin{cases} \dfrac{1}{\#\text{properties of } a_q}, & \text{if } a_q \equiv \text{domain}(a_p) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

In the opposite case, i.e., a_q is a property and a_p is a class, the similarity is differently evaluated using $propClassSim$, since our final goal is to state the similarity for substitutability purposes. Thus, once we have verified that the a_p corresponds to the domain of the property a_q, then the similarity between annotations is 1. Indeed, now (i) the annotation in the query refers to a specific property, and (ii) the annotation in the published service certainly includes such a property since it refers to the whole set of properties for the defined class. More formally,

$$annSim(a_q, a_p) = \begin{cases} 1, & \text{if } a_q \equiv \text{domain}(a_p) \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

5 URBE IMPLEMENTATION

The approach presented in this work has been implemented on a prototype, based on a UDDI Registry, which supports both the Web service publication and the Web service retrieval activities.

Fig. 7 shows the main modules composing URBE. Core of the tool is the `Similarity Engine` which embeds the functions $fSim$ and $fSim^s$ previously presented. This module can be configured to dynamically bind to the specific ontology which is considered relevant with respect to the application domain requested by the user. In addition, Wordnet is available as a general purpose ontology and the Java Wordnet similarity library [3] developed by Seco et al. [11] is used to compute similarity between terms in Wordnet. In addition, an open-source implementation of UDDI v.2., i.e., jUDDI [4] is included to support the UDDI standard API described in [18] and an open-source implementation of a Mixed Integer Linear Programming solver, i.e., LpSolve [5], is used to solve the linear programming model on which $fSim$ relies. Finally, Jena library is used for accessing OWL-based domain-specific ontologies.

URBE can be downloaded at: http://black.elet.polimi.it/urbe and a test client is available at: http://black.elet.polimi.it/urbeClient.

All of these modules provide a set of functionalities grouped in three main APIs accessible via SOAP. First,

3. `http://eden.dei.uc.pt/ nseco/javasimlib.tar.gz`
4. `http://ws.apache.org/juddi/`
5. `http://http://sourceforge.net/projects/lpsolve`

the Domain Expert API allows the configuration of URBE to set all the parameters. Second, the UDDI API is available to publish and retrieve Web services. Finally, the URBE API allows retrieving the set of Web services which are similar to a given one.

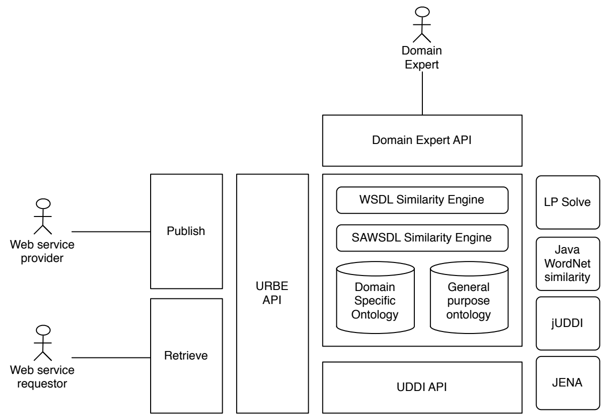


Fig. 7. URBE Architecture.

6 Experimental results

The evaluation of our work consists of two main steps. Firstly we consider a small set of Web services and we tune the algorithm, i.e., we identify the best values for the parameters which the algorithm depends on. Secondly over a larger set of Web services the performance of the algorithm and its semantic extension are evaluated and compared with related approaches.

In particular, *precision* (i.e., number of relevant returned Web services w.r.t. the number of returned Web services) and *recall* (i.e., number of relevant returned Web services w.r.t. total relevant Web services in the corpus) have been adopted as the parameters to evaluate the performance of our approach [7]. More precisely:

$$P(\sigma_q) = \frac{|\{\sigma_i \in \Sigma_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|}{|\Sigma_{\sigma_q}|} \quad (12)$$

$$R(\sigma_q) = \frac{|\{\sigma_i \in \Sigma_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|}{|\Re_{\sigma_q}|} \quad (13)$$

where σ_q is the query, Σ_{σ_q} the returned services after submitting the query, and $\Re_{\sigma_q}$ the relevant services for the given query.

Precision and recall are measures for the entire result set without considering the ranking order. Thus, *R-Precision* and *AP* (*Average Precision*) parameters are also considered. Both of them depends on the precision at a given cut-off point (P^n). Thus, assuming $\Sigma^n_{\sigma_q}$ as the set including the first n returned services:

$$P^n(\sigma_q) = \frac{|\{\sigma_i \in \Sigma^n_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|}{n} \quad (14)$$

As a consequence, the R-Precision returns the precision when the cut-off corresponds to the number of total relevant Web services in the corpus, i.e., $|\Re_{\sigma_q}|$:

$$\text{R-Precision}(\sigma_q) = P^{|\Re_{\sigma_q}|}(\sigma_q) = \frac{|\{\sigma_i \in \Sigma^{|\Re_{\sigma_q}|}_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|}{|\Re_{\sigma_q}|} \quad (15)$$

Finally, the AP is the average of precisions computed after truncating the list after each of the relevant documents, as long as all the relevant documents are retrieved:

$$\text{AP}(\sigma_q) = \frac{\Sigma_{r=1..N} P^r(\sigma_q)}{|\{\sigma_i \in \Sigma^N_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|} \quad (16)$$

where $N = |\{\sigma_i \in \Sigma_{\sigma_q} | \sigma_i \in \Re_{\sigma_q}\}|$ is the number of relevant documents.

The benchmark adopted for both tuning and evaluating the performance of the similarity algorithm has been obtained from the OWL-S service retrieval test collection (OWL-S TC) [6]. It consists of more than 570 Web services specified with OWL-S covering seven application domains, that are education, medical care, food, travel, communication, economy, and weaponry. The benchmark also includes 32 test queries, represented as OWL-S documents, each of which is associated with a set of services that the proponents of the benchmark have defined as relevant. Five of these 32 test queries are used to tune the algorithm, whereas the remaining 27 test queries are used to evaluate the approach.

Since our similarity algorithm works with WSDL, before using this benchmark, we derived a WSDL for each of the OWL-S documents in the test collection using the tool OWLS2WSDL [7]. As a consequence, we have a benchmark composed by more than 570 Web services σ_i and 32 test queries σ_q. For each of these queries we can calculate the *precision*, the *recall* and all the measures introduced above.

6.1 Tuning

The tuning phase aims at identifying for which values of $wPTNameSim$, $wOpNameSim$, $wParNameSim$ we obtain the best performances. For this goal, we randomly select 5 queries from the 32 test queries defined in the benchmark. More specifically:

- 1personbicyclecar_price_service.wsdl
- bookpersoncreditcardaccount__service.wsdl
- citycountry_hotel_service.wsdl
- shoppingmall_cameraprice_service.wsdl
- surfinghiking_destination_service.wsdl

For the sake of simplicity, in this presentation we mainly focus on the parameters $wPTNameSim$ and $wOpNameSim$ so, we set $wParNameSim = 0.7$. We decided to give more importance to the parameter name, since analyzing the corpus we realize that parameters are often strings. Therefore, if we gave the same importance to the data type as the name, we would introduce false

6. http://projects.semwebcentral.org/projects/owls-tc/
7. OWLS2WSDL tool is available at http://projects.semwebcentral.org/projects/owls2wsdl/

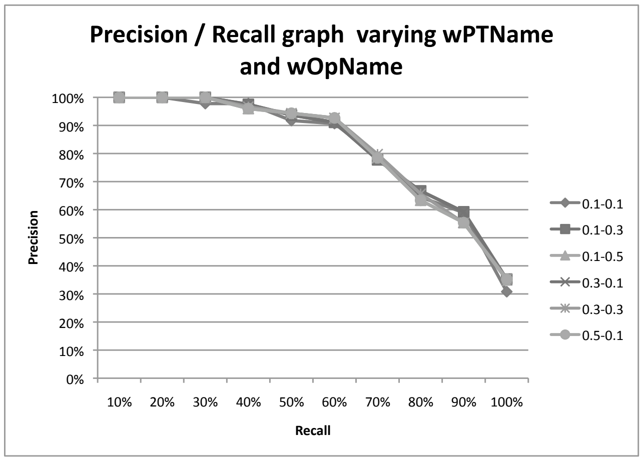


Fig. 8. Precision/recall graph varying $wPTNameSim$ and $wOpNameSim$.

positives and thus the overall accuracy of the result decreases. At the same time, we do not want to completely skip the data type analysis, since in some cases data types other than strings are present.

About $wPTNameSim$ and $wOpNameSim$, we identify their best values by submitting the five tuning queries several times varying these parameters. In particular, we run the algorithm 25 times, each of them with a different combination of $wPTNameSim$ and $wOpNameSim$ by using pair of values in the set [0.1; 0.3; 0.5; 0.7; 0.9].

Considering the resulting average for precision and recall, we notice very close performances for several combinations of $wPTNameSim$ and $wOpNameSim$. Fig. 8 shows the precision/recall curves for the six combinations, among the 25 ones, that provide the best trends. In particular, we obtain the best performances in case of $wPTName = 0.1$ and $wOpName = 0.3$. In order to plot Precision/Recall graphs where recall varies from 0% to 100%, we set $th_{fSim} = 0$. As a consequence, the result set will always include all the published services ranked with respect to the similarity value. According to this result, the global evaluation of URBE discussed in the next section will be performed using: $wPTNameSim = 0.1$, $wOpNameSim = 0.3$, $wParNameSim = 0.7$. It is worth noting that the tuning process should be executed by the registry manager if the Web service corpus considerably changes. Since new Web services can be added to the registry, as well as Web service might be deleted, the parameters might be affected by this modification since the algorithm depends on several aspects as the way in which the WSDL are built (e.g., automatically or manually) and which is the naming convention adopted by the programmers (e.g., Java, .NET).

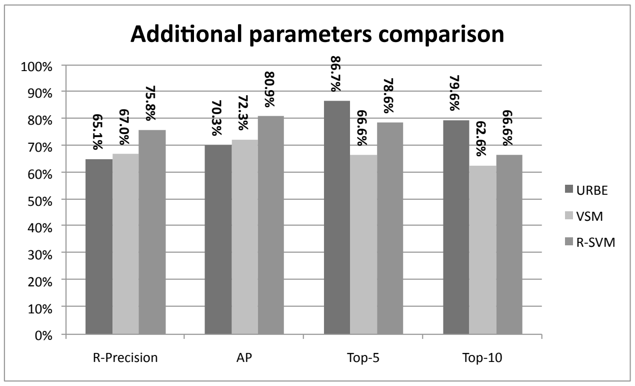
6.2 Algorithm evaluation

Our approach, once properly tuned, has been evaluated considering the whole benchmark. Excluding the five queries used during the tuning phase, we now submit the remaining 27 test queries of the total 32 queries defined in the benchmark and we compare the result with some relevant work. About the related approaches, their results are directly taken from the papers that introduce the work which usually relies on benchmarks built on purpose by the same authors that also authoritative define the relevant sets for each submitted query.

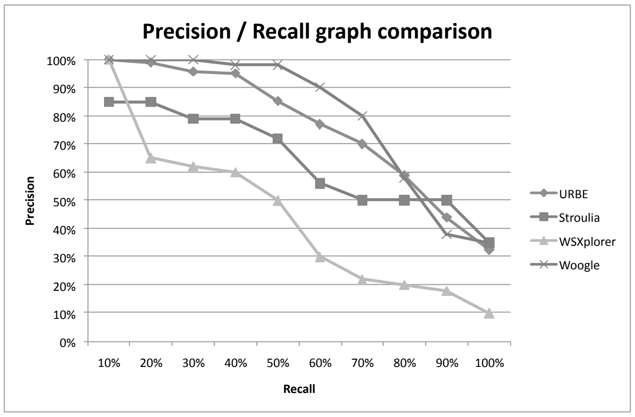
Fig. 9 shows the results of our experiments compared to the related work. Since the papers describing the related work discuss their results according to different models, we introduce the comparison in two main ways. In Fig. 9(a), the classical Vector Space Model (VSM) and the Ranking-SVM (R-SVM) described in Yu et al. [19] are compared with our algorithm by calculating R-Precision, $P^5(\sigma_q)$ (a.k.a *Top-5 precision*), $P^{10}(\sigma_q)$ (a.k.a. *Top-10 precision*), and Average Precision [7]. In Fig. 9(b), Precision/Recall graph along with the results of the approaches proposed by Hao et al. [20] (WSXplorer), Stroulia and Wang [13] (Stroulia), and Dong et al. [21] (Woogle).

Focusing on the first comparison, although the R-Precision and the AP results lower in URBE than VSM and R-SVM, the Top-5 as well as the Top-10 precisions are considerably higher. This means that, given a query, our algorithm almost always returns the relevant Web services in the first positions (i.e., 86.7% for Top-5 and 79.6% for Top-10). Considering our main scenario, where a developer queries the registry to obtain a Web service that can substitute an existing one, high values for Top-5 and Top-10 mean that the most similar ones are returned in the first positions.

About the second comparison, the Precision/Recall graph in Fig. 9(b) compares the performance of URBE against to WSXplorer, the approach proposed by Stroulia and Wang, and Woogle. Briefly, in WSXplorer compared Web services, accordingly to the structure of a WSDL, are represented as trees and the similarity is computed evaluating an edit distance between these trees. Stroulia and Wang approach is similar to URBE: a recursive pair-wise comparison among data types, messages, operations, and Web service is performed for stating the similarity. Even in this case, the name of operations, messages, and parameters are taken into account as well as the data type of the parameters, but Stroulia and Wang also consider, if available, text description fields. With respect to URBE, in this case, the structural similarity does not consider the number of elements compared, so it does not return a normalized score. In case of Woogle, the names adopted in the Web service descriptions are considered as bag of words so that the similarity is computed using the tf/idf algorithm. Nearby the term comparison, in Woogle the concept comparison is computed as well. Concepts are derived from the names used in the Web service description and the similarity between concepts occurs after a clustering process that identifies relationships among all the terms in the corpus of Web services. Thus, the similarity among concepts is computed with respect to such relationships.



(a)



(b)

Fig. 9. Comparison between URBE and some of the related work.

Considering WSXplorer and Stroulia and Wang approaches, URBE has the best performance with respect to both precision and recall. In details, in URBE the precision remains above the 80% as long as the recall is lower than 70%. Woogle, instead, has a better precision than URBE when the recall is lower but a worst precision with higher recall (greater than 80%). This difference may be motivated by the different way in which the semantic term similarity is evaluated and the query structure. Indeed, Woogle relies on a clustering algorithm that creates semantic associations between the terms included in all the published services regardless of the submitted query. About the query, Woogle allows to search for a single operation and a returned Web service is considered relevant if it contains at least one operation similar to the requested one. In case of URBE, the user can look for a Web service that includes more than one operations.

6.3 Semantic extension evaluation

To evaluate the semantic-aware variant of our retrieval algorithm, we start from the same benchmark introduced above. In this case, SAWSDL descriptions are obtained adding annotations to the WSDL previously derived from OWL-S files. In details, annotations are attached to the WSDL message parts of both input and output parameters using the `parameterType` fields in the OWL-S description. Relying on this relationship, we developed an application for automatically annotate a WSDL, previously obtained by using OWLS2WSDL, according to the SAWSDL specifications. Fig. 10 shows an excerpt of annotated message obtained by processing a OWL-S file. Fig. 11 shows the result of the evaluation comparing the precision/recall curve when the semantic annotations are considered (URBE-S) against the work of Syeda-Mahmood et al. [22] and the results introduced in the previous section when algorithm purely based on service interfaces is considered (URBE). The approach proposed by Syeda-Mahmood et al. (Syeda) does not consider the structure of the WSDL description but it only focuses on the terms or, if present, the annotations included in the Web service description.

As shown in the Fig. 11, we notice how the semantic-aware variant of our algorithm has a better precision than the other approaches. In addition, we also compare URBE-S with Woogle that, as discussed in the previous section, was better than our approach when considering low recall due to some use of semantics. Now, since we can exploit the semantic captured by the annotation, our results significantly improves even in case of lower recall and the gap with Woogle is reduced. In the comparison we also include the Precision/Recall graph of OWLS-MX [23] that is obtained with the same benchmark we adopted. Even if in this case the similarity is calculated compared two OWL-S descriptions, we can notice how URBE-S has a better trend even if it considers only the concepts related to the input and output parameters.

All the experiments discussed in this paper have been

```
...
<process:Input rdf:ID="_4WHEELEDCAR">
  <process:parameterType>http://127.0.0.1/ontology/
                         my_ontology.owl#4WheeledCar
  </process:parameterType>
  <rdfs:label></rdfs:label>
</process:Input>

<process:Output rdf:ID="_PRICE">
  <process:parameterType>http://127.0.0.1/ontology/
                         concept.owl#Price
  </process:parameterType>
  <rdfs:label></rdfs:label>
</process:Output>
...
```

```
...
<wsdl:message
  name="_4wheeledcar_Request">
  <wsdl:part name="_4wheeledcar"
             sawsdl:modelReference=
             "http://127.0.0.1/ontology/
              my_ontology.owl#4WheeledCar"/>
  <wsdl:part name="_Price"
             sawsdl:modelReference=
             "http://127.0.0.1/ontology/
              concept.owl#Price"/>
</wsdl:message>
...
```

Fig. 10. Example of SAWSDL derived from OWL-S description.

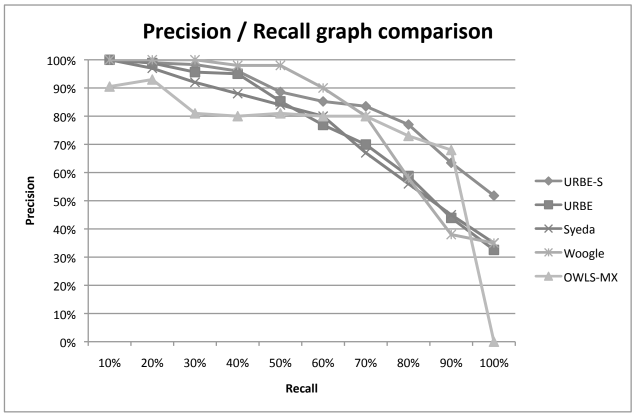


Fig. 11. Comparison between URBE, its semantic-variant URBE, another semantic-based approach, and Woogle.

done on an IBM eServer xSeries (with 2 CPU Intel Xeon 3GHz) and 4GByte of RAM. The part of the algorithm able to solve the linear programming problem exploits the open-source application LPSolve, and it has been formulated to always obtain the global optimum result. The execution-time of $fSim$ is directly affected by the exponential complexity to solve the assignment in bipartite graphs problems [24]. Some heuristics [25] reduce the complexity to $O(n^2)$, where n is the sum of the cardinalities of the two sets we consider. According to the analysis presented in [26], on average a Web service has 3 operations and each operation has 4 parameters (considering both inputs and outputs). Comparing two Web services with these characteristics, our approach requires about 0.3 sec.

7 Related work

Web service retrieval, also called Web service discovery, is one of the fundamental steps in Service Oriented Computing, and several papers can be found in the literature of several communities including: Information Systems, Software Engineering, Semantic Web, and Software Agents. Extending the definition provided in [27], Web service Discovery can be defined as the act of locating the machine-processable descriptions of the Web service that may have been previously unknown and which meet the criteria expressed by the service requestor. Garofalakis et al. in [6] introduce a general overview of current Web service discovery mechanisms and propose a categorization with respect to: architectures, standards, QoS-awareness, and data models. According to such a categorization our efforts aim to cover most of these aspects. At this stage, our similarity algorithm works with WSDL and SAWSDL standards, and takes into account both Information Retrieval and Semantic Web data models. The resulting tool, i.e., URBE, extends the UDDI Registry architecture.

Although the classification proposed by Garofalakis et al. provides a good overview, the paper does not enter into the details of the retrieval algorithms on which the listed mechanisms are based on. Focusing on this aspect, Klein and Bernstein [1] identify four main retrieval approaches: keyword-based, concept-based, table-based, and deductive. Keyword-based approaches require that all the documents are associated with keywords which the retrieval algorithms are based on. Concept-based approaches rely on defining an ontology of concepts for classifying documents. Table-based approaches consist of attribute-value pairs that capture service properties. Finally, deductive-based approaches require the description of Web services with formal logics. Retrieval then consists in inferring which services achieve the functionality described in the query. As stated by the authors, the keyword-based approaches are the easiest to be implemented, but they provide lower precision and recall. On the contrary, the deductive-based ones appear to be the most precise, but they involve the use of formal logic: Web service capabilities and user queries are more difficult to define and the computational complexity grows. According to this classification, our approach mixes the simplicity of the table-based approach and the precision of concept-based retrieval techniques. In fact, we mainly use WSDL for expressing the Web service capabilities, as usually done by programmers. In addition, the queries follow the same grammar and this way it is more usable for the requestor.

As in our work, other work in the literature relies on the syntax of the Web service description and compares the signature of the requested service with respect to the signatures of the existing Web services. This approach is closely related to the approaches studied in the reusable components retrieval literature [28]. In this field, as stated by Zaremski and Wing, there are two types of methods to address this problem: signature matching [5] and specification matching [29]. In particular, signature matching considers two levels of similarity introducing the *exact* and *relaxed* signature matching. In our work, signature matching represents the core of the approach. In addition, our similarity algorithm also quantifies how similar a Web service is to another one, instead of simply dividing the retrieved Web services in exact matching and relaxed matching ones. In addition to the signature, Stroulia and Wang, in [13], exploit the description fields as well. Adopting classical Information Retrieval techniques (e.g., tf-idf [7]) combined with Wordnet, they are able to increase the precision of the retrieval mechanisms. This approach can be considered complementary to ours since we do not consider the text description. On the contrary, the Stroulia and Wang approach does not take into account the number of operations and parameters. To improve the effectiveness of the similarity evaluation, Dong et al. [21] with Woogle propose to run a clustering algorithm for identifying the relationships among the terms adopted in all the published Web services. Then, the operation similarity is based on these pre-built relationships. With respect to our approach, Woogle allows to submit a single operation. In our case, we can search for a Web service that includes more than one operation.

Similarity analysis can also consider the component behavior. Mecella et al. [30] compare the external behavior of Web services, i.e., their conversations. Considering the conversation as a state machine, their approach compares these state machines in order to evaluate their compatibility. About the similarity of the Web service signatures, Mecella et al. do not deal with names and types compatibility assuming the presence of a domain expert able to manually discover possible similarities. In some other work, [31]–[34] describe the Web service behavior according to other models, such as Petri Nets, or considering the execution paths. In this case, the similarity evaluation compares the Web service conversations, i.e., the order in which the provided methods have to be invoked. In our approach we consider all the involved Web services as stateless ones. Other approaches use pre- and post-conditions to model the behavior. For instance, Budak Arpinar [35] study the relationships between pre- and post-conditions to figure out which is the goal of the Web service and to provide a tool for automatic composition. Spanoudakis et al. [36] introduce an integrated approach for monitoring Web service interactions to discover and to repair possible anomalies. The framework identifies the source of the anomalies and automatically defines the query as the Web service that it is required to replace the failed one. The discovery algorithm is based on the evaluation of the similarity between the execution path of the query and the candidate Web services. At this stage, our algorithm does not consider pre- and post-conditions, since WSDL does not include this information. These conditions are out-of-scope of our purposes, since we need to find similarities among Web services regardless of their invocation sequence. In future work we can consider these conditions to improve the effectiveness of our algorithm to support dynamic Web service composition.

With respect to our previous work [37], the entire approach relies on a linear programming model able to find a global optimum in the comparison of terms. In addition, we also consider tokenization and stemming, and the retrieval algorithm can rely on existing application domain ontologies. Finally, similarity for semantically annotated Web service has been introduced.

Syeda-Mahmood et al. [22] have an approach very close to ours, since the authors consider both the WSDL description and an annotated version of it. The work is also interesting since it demonstrates the effectiveness of using a domain-specific ontology: the precision of their retrieval mechanism increases when a domain-specific algorithm is involved. Moreover, as done in our approach to increase the precision, the authors also consider the term tokenization according to a code convention. Besides these elements, our work differs since it also considers the structure of the WSDL description, whereas Syeda-Mahmood et al. only focus on the terms.

It is worth noting that all the service discovery algorithms listed above rely on a set of techniques to assess the similarity among Web services. Actually these techniques are used also before the Web service discovery phase, in order to classify the Web service published in the registry to facilitate the subsequent discovery. Usually such a classification is made according to the functionalities, so that the aim is to cluster similar Web services with respect to their capabilities. In this way, Web service discovery is performed in two steps. First, the cluster containing the most relevant Web services is identified. Then, a finer-grained analysis is performed for the Web services belonging to such a cluster. Perryea and Chung [38] introduce the concept of Web service community as a way of clustering similar Web services.

A further class of similarity algorithms [23], [39]–[43] retrieves the Web services with a reasoning process on a semantic specification. Description Logic is the usual formalization adopted and results in languages such as OWL-S [44] and WSMO [45]. Even if these approaches are more effective than the ones based on WSDL, building a logic-based Web service description requires more effort for developers. Moreover, our work focuses also on the structure of the Web service, for substitution purposes. In the above mentioned algorithms the result of the retrieval activity is a set of Web services that achieve the same goal. Nothing can be said about how the goal is achieved. In addition, these approaches are usually able to group Web service in similarity classes, i.e., exact match, partial match, and relax match. On the contrary, in our approach, we offer a finer grained Web service ranking based on a similarity value. The Semantic Web community also adopts SPARQL [46] (Simple Protocol and RDF Query Language), a query language for RDF (Resource Description Framework) documents [47], as a way to express the characteristics of the required Web service [48]. According to a query-by-example approach, in our work the requested Web service is defined using the same language adopted to describe the published Web services, i.e., WSDL or SAWSDL.

Table 6 summarizes the comparison among the most significant related work with respect to the nature of the approach. The classification takes into account which Web service elements are considered (interface, pre- and

TABLE 6
Related work on Web service retrieval comparison.

Proposal	**Interface analysis**	**Pre/post conditions analysis**	**Behavior analysis**	**Semantic**
Zaremski and Wing [29]	X	X	-	-
Stroulia and Wang [13]	X	-	-	X
Mecella et al. [30]	X	-	X	-
Dong et al. [21]	X	-	-	X
Budak Arpinar [35]	X	X	-	-
Syeda Mahmood [22]	X	-	-	X
URBE	X	-	-	X

post-conditions, or behavior) and if a semantic Web technique is adopted to compare the names included in the description.

8 Concluding remarks and future work

In this paper we have presented URBE, an approach for evaluating the similarity between Web service interfaces for substitutability purposes. The Web service requestor, after submitting the interface of the desired Web service, can obtain a list of similar Web services. The evaluation of the similarity between Web services considers both the semantic and the structure of a WSDL description. The semantic analysis takes into account the names adopted to describe the elements composing a Web service (operations and parameters), whereas the structure analysis takes into account the number of operations as well as the number and data types of the parameters. In addition, our approach also supports SAWSDL as a description model. In this case, the semantic analysis takes advantage of the semantic relationships between annotations in the SAWSDL as also demonstrated in the Semantic Service Selection (S3) Contest [49].

A prototype of URBE has been developed as a UDDI-compliant registry that supports our retrieval model and it has been used to validate of our approach.

Further work will focus on improving performance in terms of execution time. First of all, a clustering of the Web services published in the registry can be periodically done in order to automatically create the application domain-based classification. In case the Web services are also described with OWL-S, we plan to exploit also this description to create these clusters. Secondly, we will refer to [50], where the authors propose a set of basic principles towards efficient semantic Web service discovery. In particular, these principles focus on: semantic level (reducing ontology management) and matching level (reducing the number of comparisons). Additional improvements can be also done for the tuning phase. About this, we need to study some approaches for automatically tuning the parameters with respect to the nature of the published Web services.

Moreover, the semantic analysis of a WSDL can consider differently the comparison between method names and parameter names. About the former, the verb is more important since a method name should define an action. About the latter, the parameter names similarity should mainly consider the noun, i.e., the meaning of data on which the action is performed or its output. Considering the SAWSDL analysis, the next steps aim to consider in a single step the annotations at different levels in the structure. For instance, if the required operation is called as *formatDocument*, whereas the offered operation has the operation and input parameter annotated as with *format* and *document*, then we should realize that they are strictly related.

Finally, quality of service will be addressed in future work for extending the registry. At this stage we have only some preliminary work [51] where a quality model and a quality selection approach have been proposed. An additional extension might allow URBE, given a request of Web service, to return a set of Web services that, once composed, can satisfy the initial request.

Acknowledgments

The authors would like to thank Danilo Ardagna for his help and all the students who helped us in developing URBE. A special thanks to Fabrizio De Santis for his valuable effort in creating a re-engineered and usable version of the tool. This work has been partially funded by Italian MIUR-FIRB Tekne Project.

References

[1] M. Klein and A. Bernstein, "Toward high-precision service retrieval," *Internet Computing, IEEE*, vol. 8, no. 1, pp. 30–36, Jan-Feb 2004.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1," http://www.w3.org/TR/wsdl, World Wide Web Consortium, W3C Note, March 2001.

[3] J. O. Kephart and D. M. Chess, "The vision of autonomic computing." *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[4] J. Farrel and H. Lausen, "Semantic annotations for WSDL and XML schema," http://www.w3.org/TR/sawsdl/, April 2007.

[5] A. Zaremski and J. Wing, "Signature matching: a tool for using software libraries," *ACM Trans. Softw. Eng. Methodol.*, vol. 4, no. 2, pp. 146–170, 1995.

[6] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, "Contemporary Web service discovery mechanisms," *Journal of Web Engineering*, vol. 5, no. 3, pp. 265–290, 2006.

[7] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.

[8] S. Microsystems, "Code conventions for the Java programming language," http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html, April 1999.

[9] M. Lennon, D. Pierce, B. Tarry, and P. Willett, "An evaluation of some conflation algorithms for information retrieval," *Journal of Information Science*, vol. 8, no. 3, pp. 99–105, 1988.

[10] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet::Similarity - measuring the relatedness of concepts," in *Proc. National Conf. on Artificial Intelligence, July 25-29, San Jose, California, USA*, 2004, pp. 1024–1025.

[11] N. Seco, T. Veale, and J. Hayes, "An intrinsic information content metric for semantic similarity in Wordnet," in *Proc. Eureopean Conf. on Artificial Intelligence (ECAI'04), Valencia, Spain, August 22-27*. IOS Press, 2004, pp. 1089–1090.

[12] P. Biron and A. Malhotra, "XML schema part 2: Datatypes second edition (W3C Recommendation)," http://www.w3.org/TR/xmlschema-2/, October 2004.

[13] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing Web-service similarity," *Int'l J. Cooperative Inf. Syst.*, vol. 14, no. 4, pp. 407–438, 2005.

[14] E. Rahm and P. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[15] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese, "Fast detection of XML structural similarity," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 2, pp. 160–175, 2005.

[16] M. Dean and G. Schreiber (eds.), "OWL: Web ontology language," http://www.w3.org/TR/owl-ref/, February 2004.

[17] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[18] T. Bellwood, "UDDI version 2.04 API specification," http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, July 2002.

[19] Y. Jianjun, G. Shengmin, S. Hao, Z. Hui, and X. Ke, "A kernel based structure matching for Web services search," in *Proc. of the Int'l Conf. on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, 2007, pp. 1249–1250.

[20] Y. Hao, Y. Zhang, and J. Cao, "WSXplorer: Searching for desired web services," in *Proc. Int'l Conf. Advanced Information Systems Engineering (CAiSE'07), Trondheim, Norway, June 11-15*, 2007, pp. 173–187.

[21] X. Dong, J. Madhavan, and A. Halevy, "Mining structures for semantics," *SIGKDD Explor. Newsl.*, vol. 6, no. 2, pp. 53–60, 2004.

[22] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A. Ivan, and R. Goodwin, "Searching service repositories by combining semantic and ontological matching," in *Int'l Conf. on Web Services (ICWS'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 13–20.

[23] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," in *Proc. Int'l Conf. on Autonomous agents and multiagent systems (AAMAS'06)*. New York, NY, USA: ACM Press, 2006, pp. 915–922.

[24] L. Wolsey, *Integer Programming*. John Wiley and Sons, 1998.

[25] J. Wang, J. Xiao, C. Lam, and H. Li, "A bipartite graph approach to generate optimal test sequences for protocol conformance testing using the Wp-method," in *Proc. Asia-Pacific Software Engineering Conf. (APSEC'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 307–316.

[26] H. Kil, S.-C. Oh, and D. Lee, "On the topological landscape of Web services matchmaking," in *Proc. Int'l Workshop on Semantic Matchmaking and Resource Retrieval (VLDB-SMR'06)*, CEUR, Ed., vol. 178, 2006, published on line.

[27] D. Booth, F. M. H. Haas, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web service architecture," W3C Working Group, February 2004.

[28] E. Damiani, M. G. Fugini, and C. Bellettini, "A hierarchy-aware approach to faceted classification of objected-oriented components," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, no. 3, pp. 215–262, 1999.

[29] A. Zaremski and J. Wing, "Specification matching of software components," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 4, pp. 333–369, 1997.

[30] M. Mecella, B. Pernici, and P. Craca, "Compatibility of e-services in a cooperative multi-platform environment," in *Proc. Int'l Workshop on Technologies for E-Services (TES'01), Rome, Italy, September 14-15*, 2001, pp. 44–57.

[31] J. Bae, L. Liu, J. Caverlee, and W. Rouse, "Process mining, discovery, and integration using distance measures," in *Int'l Conf. on Web Services (ICWS'06)*, September 2006, pp. 479–488.

[32] M. Mecella, F. Parisi-Presicce, and B. Pernici, "Modeling e-service orchestration through Petri nets," in *Proc. Int'l Workshop on Technologies for E-Services (TES'02), Hong Kong, China, August 23-24*. London, UK: Springer-Verlag, 2002, pp. 38–47.

[33] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "A foundational vision of e-services." in *Web Services, E-Business, and the Semantic Web, Second Int'l Workshop (WES'03), Klagenfurt, Austria, June 16-17*, 2003, pp. 28–40.

[34] A. Martens, "Process oriented discovery of business partners," in *Proc. Int'l Conf. on Enterprise Information Systems (ICEIS'05), Miami, USA, May 25-28*, 2005, pp. 57–64.

[35] I. Budak Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko, "Ontology-driven Web services composition platform," in *Proc. IEEE Int'l Conference on e-Commerce Technology (CEC'04)*, 6-9 July 2004, pp. 146–152.

[36] G. Spanoudakis, A. Zisman, and A. Kozlenkov, "A service discovery framework for service centric systems," in *Int'l Conference on Services Computing (SCC'05)*. IEEE Computer Society, 2005, pp. 251–259.

[37] D. Bianchini, V. D. Antonellis, B. Pernici, and P. Plebani, "Ontology-based methodology for e-service discovery," *Journal on Information Systems*, vol. 31, no. 4-5, pp. 361–380, 2006.

[38] C. Perryea and S. Chung, "Community-based service discovery," in *Int'l Conf. on Web Services (ICWS'06)*, Sept. 2006, pp. 903–906.

[39] S. Agarwal and R. Studer, "Automatic matchmaking of Web services," in *Int'l Conf. on Web Services (ICWS'06)*, Sept. 2006, pp. 45–54.

[40] D. Bianchini, V. D. Antonellis, and M. Melchiori, "Hybrid ontology-based matchmaking for service discovery," in *Proceedings of the ACM symposium on Applied computing (SAC'06)*. Dijon, France: ACM Press, 2006, pp. 1707–1708.

[41] B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Toumani, "On automating Web services discovery," *The VLDB Journal*, vol. 14, no. 1, pp. 84–96, 2005.

[42] K. Sycara, S. Widoff, M. Klusch, and J. Lu, "Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace," in *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 2. Hingham, MA, USA: Kluwer Academic Publishers, 2002, pp. 173–203.

[43] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of Web services capabilities," in *Proc. Int'l Semantic Web Conference on The Semantic Web (ISWC'02)*. London, UK: Springer-Verlag, 2002, pp. 333–347.

[44] D. Martin (ed.), "OWL-S: Semantic Markup for Web Services. W3C Submission," http://www.w3.org/Submission/OWL-S/, November 2004.

[45] H. Lausen, A. Polleres, and D. Roman (eds.), "Web Services Modeling Ontology (WSMO). W3C Submission," http://www.w3.org/Submission/WSMO/, June 2005.

[46] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," http://www.w3.org/TR/rdf-sparql-query/ (W3C Candidate Recommendation), June 2007.

[47] D. Beckett (ed.), "RDF/XML Syntax Specification (Revised). W3C Recommendation," http://www.w3.org/TR/rdf-syntax-grammar/, February 2004.

[48] S. Lamparter and A. Ankolekar, "Automated selection of configurable Web services," in *8. Int. Tagung Wirtschaftsinformatik*. Universittsverlag Karlsruhe, Germany, March 2007.

[49] M. Klusch, "OWL-S and SAWSDL Service Matchmakers. S3 Contest 2008 Summary Report," http://www-ags.dfki.uni-sb.de/~klusch/s3/s3c-2008.pdf, October 2008.

[50] S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny, "Towards efficient matching of semantic Web service capabilities," in *Int'l Workshop on Web Services Modeling and Testing (WS-MATE'06). Online proceedings*, 2006.

[51] M. Fugini, P. Plebani, and F. Ramoni, "A user driven policy selection model," in *Proc. Int'l Conf. on Service-Oriented Computing (ICSOC'06), Chicago, IL, USA, December 4-7*, ser. LNCS 4294, A. Dan and W. Lamersdorf, Eds. Springer, 2006, pp. 427–433.

Pierluigi Plebani received the master degree in computer science engineering and the PhD degree in information engineering from Politecnico di Milano, Italy, in 2000 and 2005, respectively. He currently belongs to the Information Systems group as assistant professor in the Dipartimento di Elettronica ed Informazione of Politecnico di Milano. His research interests concern Web service retrieval methods driven by both functional and quality aspects, and the design of adaptive information systems.

Barbara Pernici received the Italian laurea in electronic engineering from Politecnico di Milano, Italy in 1981, and the Master of Science in Computer Science from Stanford University in 1984. She currently belongs to the Information Systems group as full professor in the Dipartimento di Elettronica ed Informazione of Politecnico di Milano. Her research interests include workflow information systems design, cooperative information systems, adaptive information systems, service engineering and web services, data quality, and computer based design support tools. She has published more than 40 papers in international journals, co-edited 12 books, and published about 140 papers at international level. She serves as elected chair of TC8 Information Systems of the International Federation for Information Processing (IFIP) and of IFIP WG 8.1 on Information Systems Design.