Inf Syst Front (2018) 20:1319–1344
DOI 10.1007/s10796-016-9731-1

A structural-semantic web service selection approach to improve retrievability of web services

Martin Garriga[1,3] · Alan De Renzis[1,3] · Ignacio Lizarralde[2,3] · Andres Flores[1,3] · Cristian Mateos[2,3] · Alejandra Cechich[1] · Alejandro Zunino[2,3]

Published online: 27 December 2016
© Springer Science+Business Media New York 2016

Abstract Service-Oriented Computing promotes building applications by consuming and reusing Web Services. However, the selection of adequate Web Services given a client application is still a major challenge. The effort of assessing and adapting candidate services could be overwhelming due to the "impedance" of Web Service interfaces expected by clients versus the actual interfaces of retrieved Web Services. In this work, we present a novel structural-semantic approach to help developers in the retrieval and selection of services from a service registry. The approach is based on a comprehensive structural scheme for service Interface Compatibility analysis, and WordNet as the semantic support to assess identifiers of operations and parameters. We also empirically analyze, compare and contrast the performance of three service selection methods: a pure structural approach, a pure semantic approach, and the structural-semantic (hybrid) approach proposed in this work. The experimental analysis was performed with two data-sets of real-world Web Services and a service discovery support already published in the literature. Results show that our hybrid service selection approach improved effectiveness in terms of *retrievability* of Web Services compared to the other approaches.

Keywords Web services · Web service discovery · Service interface · Service selection · Structural service selection · Semantic service selection

✉ Martin Garriga
martin.garriga@fi.uncoma.edu.ar

1 GiiSCo Research Group, Faculty of Informatics, Universidad Nacional del Comahue (UNComa), Buenos Aires 1400, Neuquen, Argentina

2 ISISTAN Research Institute, Universidad Nacional del Centro de la Provincia de Buenos Aires (UNPCBA), Tandil, Buenos Aires, Argentina

3 Consejo Nacional de Investigaciones Científicas y Técnicas, (CONICET), Buenos Aires, Argentina

1 Introduction

Web Services is the common technological choice for materializing the Service-Oriented Computing (SOC) paradigm (Papazoglou et al. 2007; Erl et al. 2008). Basically, Web Services enable service providers to implement their services using well-known interoperable Web protocols, such as HTTP (HyperText Transfer Protocol) or SOAP (Simple Object Access Protocol). A Web Service *contract* (usually specified in WSDL[1] (Curbera et al. 2002)) exposes public capabilities to potential clients as operations without any ties to proprietary communication frameworks.

However, the broad use of the SOC paradigm requires efficient approaches to enable service discovery and consumption from within applications (McCool 2005). Discovery implies querying a registry to retrieve the most relevant candidate services, and then refine the set of candidates by picking up (selecting) one from them, while consumption means integrating the selected service into the client application so it can be actually called. In practice, developers manually search for candidate services mainly exploring registries or Web catalogs, and then assemble their client application components and selected services.

[1] Web Service Description Language - http://www.w3.org/TR/wsdl

As such, service discovery requires investing a large effort into service retrieval, i.e., obtaining candidate services from registries or catalogs, and service selection, i.e., assessing a set of retrieved services to find the most suitable one according to certain criteria. Even with a reduced set of services, such effort may overwhelm developers. Then, assessing services includes envisioning the required adaptations for their correct integration into the client application. Without properly analyzing meaningful information from interfaces, selection of the most suitable candidate service resembles to fortune-telling.

In this context, the aims of this paper are twofold. On the one side, we present a novel hybrid service selection approach, which improves service retrieval and selection through gathering and assessing structural and semantic information from services interfaces. On the other side, we compare the hybrid approach against other structural and semantic approaches for service selection.

In a previous work, we have presented an approach for structural service selection (Garriga et al. 2013). In this work we empower the service selection approach by also considering semantic information present in service interfaces, through the use of the WordNet lexical database (Miller et al. 1990) – thus resulting in a *hybrid* approach. Based on the structural and semantic information, the service selection approach measures the adaptability degree of candidate services upon a (potentially partial) required functionality at the client side. The required functionality is typically provided by the developers as an example of their needs in their preferred programming language (Crasso et al. 2011a) to guide the search engine to select the most adequate candidate service. In this paper, we focus on Java, a language very popular in SOC development. Alternatively, we also study a purely semantic service selection approach based on the notion of information content (Pirró 2009), which is also compared with the aforementioned approaches.

Additionally, we performed a set of experiments to compare the performance of the hybrid approach against other service selection approaches upon the original results of a service registry called EasySOC (Crasso et al. 2014). For comparison purposes, we also consider the Stroulia algorithm (Stroulia and Wang 2005), a foundational approach in the field of Web Service retrieval and selection. Results were measured in terms of retrievability, by means of well-known metrics from the Information Retrieval (IR) field (Recall and Discounted Cumulative Gain). Thus, we assess to what extent these service selection approaches also improve retrievability of suitable services according to the aforementioned metrics.

This paper is organized as follows. Section 2 explain two baseline service selection approaches: purely structural and purely semantic. Then, Section 3 presents the novel structural and semantic (a.k.a. *hybrid*) service selection approach. Section 4 presents the experiments comparing these Web Service selection approaches using two data-sets of real Web Services and the EasySOC service registry, and the Stroulia algorithm as the baseline. Section 5 presents related work and explains how our approach improves over them. Finally, Section 6 outlines conclusions and future work.

2 Baseline approaches for service selection

During the development of a SOC application, specific parts of a system may be implemented in the form of in-house components, whereas some (non-implemented) software pieces could be fulfilled by associating them to external Web Services. In this case, a list of candidate Web Services could be obtained by making use of a service registry. Nevertheless, even with a wieldy candidate list, a developer must be skillful enough to select the most appropriate service for the client application. Therefore, a reliable and practical support is required to help them making these decisions. In this section we describe two baseline service selection approaches. Section 2.1 presents the structural-based service selection method introduced in our previous work (Garriga et al. 2013). Section 2.2 presents a semantic-based service selection method, developed in the context of this paper, based on a semantic similarity metric developed in (Pirró 2009), which combines features and the intrinsic information expressed by identifiers.

2.1 Structural service selection

Let I_R be the interface of certain required functionality, and I_S the interface of a candidate Web Service S. For each pair of operations (op_R, op_S), a likely equivalence is foremost based on structural conditions for some signature elements – namely, return type, parameter types and exceptions. Notice that elements are named according to Java terminology, rather than using WSDL conventions for Web Service interfaces. The reason is that structural assessment is performed upon Java interfaces previously derived from WSDL specifications.

Constraints are based on individual conditions for each element comprising operations signature: the Return type (R), the operation Name (N), the Parameters list (P), and the Exceptions list (E). Table 1 summarizes the set of operation matching conditions.

Particularly, in a purely structural service selection approach, conditions for operation matching depend upon type equivalence and subtyping rules, summarized in Table 2 for the case of built-in types in the Java language. The counterpart of the subtype equivalence is *precision loss*,

Table 1 Structural-based selection: operation matching conditions for Interface Compatibility

Return Type	R0: Not Compatible	R1: Equal return type
	R2: Equivalent return type (subtyping, Strings or Complex types)	R3: Non equivalent complex types or precision loss
Operation Name	N0: Not compatible	N1: Equal operation name
	N2: Equivalent operation name (substring)	N3: Distinct operation names
Parameters List	P0: Not compatible	P1: Equal number, type and order for parameters
	P2: Equal number and type for parameters, but different order	P3: Equal number of parameters, with compatible types
	P4: Non equivalent complex types or precision loss	
Exceptions List	E0: Not compatible	E1: Equal number, type and order for exceptions
	E2: Equal number and type for exceptions	E3: if the original exception list is non-empty, then the candidate's list is non-empty

which implies that smaller types convey less information than greater types; e.g., casting a `long` type to an `int` type implies precision loss according to Table 2. Thus, to achieve operations matching, it is expected that types of operations in a *required interface* have at least as much precision as types on operations in a *candidate service*.

For example, if $op_R \in I_R$ includes an `int` type, a corresponding operation $op_S \in I_S$ should not have a smaller type (among numerical types) such as `short` or `byte` to be compatible. However, the `String` type is a special case, which is considered as a wildcard type – it is generally used in practice by programmers to allocate different kinds of data (Pasley 2006). Thus, we consider `String` as a supertype of any other built-in type. Besides, complex types imply a special treatment: each field of a complex type from an operation$op_R \in I_R$ must match a field from a complex type in $op_S \in I_S$, considering the aforementioned notions of type equivalence. Extra fields (if any) from the complex type in I_S may be initially left out of any correspondence without causing incompatibility, provided that all required fields in I_R were matched.

Return type Return Type equivalence is then straightforwardly calculated according to the notions of type equivalence. Given two operations $op_R \in I_R$ and $op_S \in I_S$, the highest compatibility (R1) implies equal return types in op_R and op_S. Then, R2 implies equivalent types according to the rules in Table 2. Examples are an `int` type in op_R and a `long` type in op_S, or two complex types with equivalent types in their fields. R3 implies two complex types with (certain) non-equivalent fields, or precision loss; e.g., an `int` type in op_R and a `short` type in op_S. Finally, non-equivalent return type (R0) implies not compatible types; e.g., an `int` type in op_R and a `boolean` type in op_S.

Operation name Operation names (identifiers) do not present structural information to be analyzed. Thus, name equivalence conditions are based upon substring similarity, by considering common naming conventions (Elish and Offutt 2002), as presented in Table 3. In general, developers combine terms in the form <verb> + <noun> for denoting an operation name, such as `getQuote` or `get_quote`, from where the name can be decomposed into words (`get` and `quote`) and a likely string coincidence could be found. Therefore, given two operations $op_R \in I_R$ and $op_S \in I_S$, condition N1 implies identical operation names. N2 implies a substring equivalence, e.g., `get_quote` and `get_user`. N3 implies completely distinct operation names, according to a substring analysis. N0 implies not compatible operation names. It is worth noting that this condition is included for completeness but it is not

Table 2 Subtype equivalence

op_R type	op_S type
char	String
byte	short, int, long, float, double, String
short	int, long, float, double, String
int	long, float, double, String
long	float, double, String
float	double, String
double	String

Table 3 Rules for decomposing operation names

Notation	Rule	Source	Result
Java Beans	Split when changing text case	getZipCode	get Zip Code
Special Tokens	Split when "_" occur	get_Quote	get Quote

applied in practice, to avoid discarding a potentially compatible candidate service only due to a different operation name.

Parameters list The rationale behind structural assessment of parameters is similar to the return type assessment, as it is based upon type equivalence notions and complex types analysis, as discussed at the beginning of this section. However, the complexity of parameters assessment is increased by the number and order of parameters. As shown in Table 1, given two operations $op_R \in I_R$ and $op_S \in I_S$ the most compatible case (P1) implies equal number, type and order for parameters. Similarly, P2 implies equal number and types of parameters, but defined in different order for both operations; e.g., the parameters in op_R are `(int, long)`, while in op_S are `(long, int)`. However, most common situations imply a weaker correspondence between parameter lists: equal number of parameters, with compatible types (P3) – e.g., `(int,long)` and `(double,double)`; and non-equal complex types or precision loss (P4) – e.g., op_R requires two parameters `(long,long)`, while op_S provides `(int,int)`. Finally, non-equivalent parameters (P0) implies that at least one parameter is not compatible; e.g., `(int, long)` and `(int, boolean)`.

Exceptions In the context of Web Services, fault definitions have not become a common practice: undercovering fault information within standard messages is an anti-pattern found in public WSDL interfaces (Rodriguez et al. 2013), which consists in (mis-)communicating error information in output messages from services to invoking clients.

However, by assuming software development best practices, exceptions or faults must be analyzed accordingly. In the structural approach, any operation $op_R \in I_R$ may define exceptions as a name and an associated set of fields (similarly to Java Beans or records in C++), while an operation $op_S \in I_S$ may define a fault as a message that includes a specific attribute which stands for the exception name, and other specific elements that describe the structure of the fault message. Fault messages are detected by the library that parses the WSDL document: the SoaMembrane API.[2] We opted for SoaMembrane because other popular libraries such as Axis2[3] and EasyWSDL[4] either introduce higher noise in the generated Java interface (in the form of RPC calls and configuration burden) or miss certain information of identifiers and types.

The most compatible case for exceptions (E1) implies equal number, type (considering the comprising fields) and order for exceptions. Condition E2 implies equal number and type of exceptions in op_R and op_S. Condition E3 implies that if the exception list in op_R is non-empty, then the exception list in op_S is non-empty – i.e., at least one exception is defined in the required and candidate operations. Finally, condition E0 implies that either op_R defines exceptions and op_S does not; or both define exceptions whose types are not compatible.

Compatibility gap value The final outcome of the Structural Assessment for two interfaces I_R and I_S is a matching list that captures each pair of potentially compatible operations. For example, let us consider I_R with three operations op_{R_i} $1 \le i \le 3$ and I_S with five operations op_{S_j}, $1 \le j \le 5$. Thus, the matching list after the structural assessment might result as follows:

$$(op_{R_1}, [op_{S_1}, op_{S_5}]); (op_{R_2}, [op_{S_2}, op_{S_4}]), (op_{R_3}, [op_{S_1}, op_{S_3}])$$

Additionally, for each pair (op_R, op_S), the equivalence values obtained for the different signature element [R, N, P, E] convey a numeric information about the compatibility degree between these two operations; e.g., the value of an exact equivalence [R1, N1, P1, E1] is 4, as a result from adding the value 1 of each condition. We will refer to this value as $compVal$. Upon this value, an average appraisal value named Compatibility Gap (Garriga et al. 2013) was defined to synthesize the best equivalence value for a pair of interfaces (I_R, I_S) according to their compatible operations.

The Compatibility Gap is calculated according to Formula 1, where:

- the numerator $Min(compVal(op_{R_i}, I_S))$ is the minimum value among the compatibility cases found for the operation op_{R_i}– only the best equivalence (lowest value) is considered for each operation $op_{R_i} \in I_R$; and
- the denominator $N * 4$ is the number of operations in I_R multiplied by 4, i.e. the value of the exact (best)

[2] http://membrane-soa.org/soa-model-doc/1.4/java-api/parse-wsdl-java-api.htm

[3] http://axis.apache.org/axis2/java/core/

[4] http://easywsdl.com/

equivalence. This implies that the best compatibility gap value will be 1 (when all operations present an exact matching).

$$compGap(I_R, I_S) = \frac{\sum_{i=1}^{N} Min(CompVal(op_{R_i}, I_S))}{N * 4} \quad (1)$$

Example Let us consider the required interface I_R with one operation op_R: `long sum(long,long)` and a candidate service interface I_S with one operation op_S: `long add(int, int)`. By means of the structural assessment, the resulting conditions for each signature element are [R1,N3,P4,E1]: both operations have the same return type (`long`), distinct names (considering substring equivalence), precision loss in the parameters (op_R requires two `long` parameters, while op_S provides `int`) and no exceptions. Finally, the resulting Compatibility Gap for this example is calculated according to Formula 1 as: $compGap(I_R, I_S) = (1 + 3 + 4 + 1)/4 = 2.25$

2.2 Semantic service selection

The Semantic-based service selection approach is based on the Semantic Similarity Metric (Pirró 2009). The semantic similarity metric combines the feature-based theory of semantic similarity – a well-known theory in the psychological field proposed in (Tversky 1977) – and the information theory domain. The abstract model of similarity calculates the set of features that is common to two terms, and also the set of differentiating features. Then, the similarity of a term t_1 to a term t_2 is a function of the features common to t_1 and t_2, those in t_1 but not in t_2, and those in t_2 but not in t_1. To add the notion of Information Content (IC) – a measure to quantify the amount of information expressed by a concept – to the compared terms, the author proposes to use the *msca* (Most Specific Common Abstraction) calculated as the intersection of features from t_1 and t_2. Then, the semantic similarity metric is calculated according to Formula 2.

$$sim(t_1, t_2) = 3 * IC(msca(t_1, t_2)) - IC(t_1) - IC(t_2) \quad (2)$$

The IC of term t_1 in relation to t_2 is obtained by subtracting from t_1 the common features. The metric relies in the meaningful and structured organization of its underlying lightweight ontology. We group under the term *lightweight ontologies* or *lightweight semantics,* approaches such as Microformats[5] and RDF[6] tagging, which can be built with minimal tooling, at low cost and a low adoption barrier.

[5] http://microformats.org/

[6] https://www.w3.org/RDF/

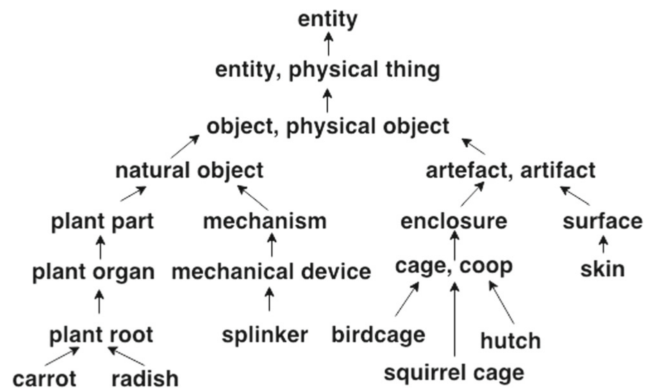

Fig. 1 A small excerpt of the WordNet taxonomic organization

In contrast, *heavyweight ontologies* or *heavyweight semantics* such as OWL-S[7] (Martin et al. 2007) and WSMO[8] (Roman et al. 2005) are formal and rigorous, allowing further machine reasoning and ontology/schema matching upon services and compositions. However, heavyweight ontologies still involve high costs in service interface and query specification, causing service designers to be alienated from their use in practice (Kokash 2006; Garriga et al. 2015). In this context, we adopt lightweight ontologies as they can exploit the semantic information in textual service descriptions (typically WSDL) that are always available in practice.

For the Semantic service selection approach, the lightweight lexical structure to calculate the *msca* is WordNet, where concepts with many hyponyms convey less information than leaf concepts.

WordNet is a large lexical database of the English language. It stores semantic relations that are used to calculate the metric. Figure 1 shows an excerpt of the nouns taxonomy of WordNet. WordNet groups terms in *synsets* (synonym sets) that represent the same lexical concept. Several relations connect different synsets, such as hyperonymy/ hyponymy, holonymy/ meronymy and antonymy (Miller et al. 1990). The Semantic Similarity Metric is based on the levels of hyperonymy/ hyponymy, since they are the most common "*Is a*" relations that connect synsets. For example, according to Fig. 1, *squirrel cage* is a first-level hyponym of *cage*, which in turn is a first level hyponym of *enclosure*. Thus, *squirrel cage* is a second-level hyponym of *enclosure*.

The Semantic Similarity Metric is implemented in the Java WordNet Similarity Library[9] (JWSL) suite of metrics that allows calculating the semantic similarity between two concepts or between two sentences. It exploits the

[7] Ontology Web Language for Services

[8] Web Service Modeling Ontology

[9] http://grid.deis.unical.it/similarity

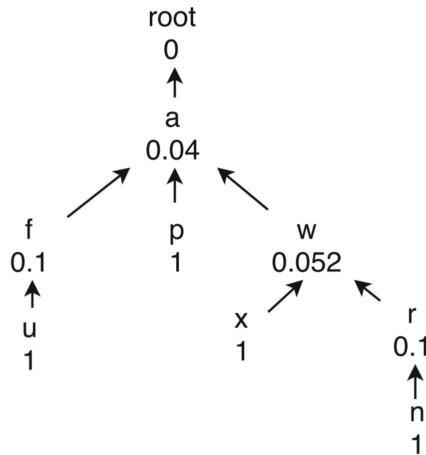

Fig. 2 Example of Semantic Similarity calculation

Lucene[10] indexing framework and includes the WordNet folder structure to avoid installing WordNet separately.

Figure 2 shows an example of intrinsic IC calculation, where the nodes (concepts) in the tree-shaped ontology structure are [root,a,f,p,w,u,x,r,n] and the corresponding intrinsic IC value is shown in each node. The Semantic Similarity metric between two leaf concepts t_1 and t_2 results in $sim(t_1, t_2) = 3 * IC(msca) - 2$. In this case, if the *msca* is near the root in the lightweight ontology (it receives a low IC) the metric returns a lower similarity value than when the *msca* is near the leaves. In particular, if the *msca* is very high (i.e., very near to the root concept) the metric gives a negative value near 2, which can be interpreted as the maximum dissimilarity value. Thus, the similarity value for the pair of terms (*f*,*w*) can be calculated as:

$$\begin{aligned} sim(f, w) &= 3 * IC(a) - IC(f) - IC(w) \\ &= 3 * 0.04 - 0.1 - 0.052 = -0.032 \end{aligned}$$

which can be interpreted as a slight similarity between the pair of concepts (*f*,w) since they share an hyperonym (*a*) that is near the root.

Following the rationale explained in the example, sentence similarity is represented as the assignment problem and then solved through the Hungarian algorithm (Kuhn 1955). This algorithm accepts as input an *n x m* cost matrix, where *n* and *m* are the number of words of the two sentences under analysis. Then, the algorithm solves the assignment problem by comparing all the words to find their best assignment according to the Semantic Similarity metric.

2.2.1 Semantic similarity for operation identifiers

The calculation of semantic similarity for service identifiers (operation and parameter names) is supported by the rationale of sentence similarity. Let op_R and op_S be two operations, then the semantic similarity matrix *S* stores the similarity between all pairs of terms from operation names. A cell S_{ij} from the similarity matrix *S* contains the semantic similarity value $sim(t_i, t_j)$ between the *i-th* term of op_R name and the *j-th* term of op_S name. Finally, the matrix *S* is used as input to the Hungarian algorithm, which returns the higher possible semantic similarity value between the identifiers.

For example, let us consider the operation identifiers op_R = `setAccountSequence` and op_S = `changeInvoiceNumber`. The semantic similarity matrix *S* for op_R and op_S is shown in Table 4, where each cell $s_{ij} = sim(t_i \in op_R, t_j \in op_S)$. The higher the value in s_{ij}, the more related the terms. Then, the Hungarian algorithm determines the best pair-wise assignment for the terms in the operations (which in this example is the diagonal of the matrix) and returns an overall similarity value.

3 Hybrid service selection

The structural service selection method presented in Section 2.1 only considers structural aspects from types in the signature elements of operations. In this paper, the associated Interface Compatibility analysis was conveniently extended to consider not only structural but also semantic aspects from each identifier included in service operations, in particular for operation names, complex return types, parameter names and exception names. Firstly, we discuss the identifiers evaluation algorithm (Section 3.1). Then we detail the structural and semantic assessment for return types (Section 4), parameters (Section 3.3), and exceptions. Application of the identifiers evaluation algorithm for operation names is straightforward, as operation names are simply identifiers. Finally, we introduce a complementary value that reflects the required adaptation effort to integrate a candidate service into a client application, called *Adaptability Gap* (Section 3.5). Again, "semantic" in this context should be understood as relying on not heavyweight but lightweight ontologies.

3.1 Identifiers evaluation

The Identifiers Evaluation Algorithm semantically compares the identifiers of an operation op_R of the required

Table 4 Semantic similarity matrix*S* for the `setAccountSequence` and `changeInvoiceNumber` operations

Term	set	account	sequence
Change	0.33	5×10^{-4}	3×10^{-4}
Invoice	1×10^{-4}	0.65	2×10^{-4}
Number	0.37	0.29	0.60

[10] http://lucene.apache.org

interface I_R and the name of an operation op_S from the candidate service interface I_S. Figure 3 depicts the main steps of the algorithm, which are described below.

Term Splitting and stop words removal Identifiers are usually restricted to a sequence of one or more letters in ASCII code, numeric characters and underscores ("_") or hyphens ("-"). The algorithm supports the rules presented for the structural selection method (Section 2.1), and considers cases that do not strictly follow the conventions described in Table 3. The term splitting step analyzes the operation identifiers. It stores potentially representative terms split by text case changing and special characters. Then, WordNet is used to analyze all the potential terms and to determine the most adequate term separation. When an identifier does not strictly follow naming conventions, some assumptions are made, e.g., an uppercase sequence represents an acronym, to find potentially representative terms.

For example, let us consider the identifier `SQLLogin`. This identifier does not follow the Java Bean naming convention. The preliminary analysis of potentially representative terms gives an uppercase sequence (`'SQLL'`) and a lowercase sequence (`'ogin'`). Then, according to the term splitting algorithm, the sequences (1) and (2) are analyzed with WordNet. As they are not valid words, the algorithm analyzes the last uppercase letter along with the lowercase sequence: `L` + `ogin` = `Login`. As this latter is an existing word in the WordNet dictionary, then `Login` is a term and `SQL` is considered an acronym (which actually stands for Structured Query Language) that is also a term.

Furthermore, stop words are words which are filtered out prior to, or after, processing of natural language data (text) (Rajaraman and Ullman 2011). We defined a stop words list containing articles, pronouns and prepositions from known stop words lists, and each letter of the alphabet. This avoids including each letter of an acronym as a term during term separation. The stop words removal step simply takes a list of terms as input and removes any occurrence of a word belonging to the stop words list.

Stemming Is the process of reducing inflected (or sometimes derived) words to their stem, i.e., base form or root form. Initially, we considered a standard stemmer such as the Porter's Algorithm (Willett 2006). However, such stemmers generate incorrect data, since they merely remove word suffixes. We considered prohibitive the accuracy decrease when using a standard lexical (or syntactical) stemmer, and then we implemented a semantic-aware stemmer based on WordNet. The ad-hoc stemming algorithm receives as input a list of terms. For each term in the list, it verifies whether the term belongs to the WordNet dictionary or not. If found, the corresponding stems are stored. Otherwise, the original term is stored in the result list, thus abbreviations and acronyms will be present in the comparison.

Semantic comparison of term lists After generating the terms lists, the information to calculate their compatibility value must be extracted using the Java Wordnet Interface[11] (JWI) library. This information includes: (1) Number of exact (identical) terms between both lists; (2) Number of synonyms (words with the same meaning); (3) Number of hyperonyms (parents); and (4) Number of hyponyms (children).

To evaluate synonyms, hyperonyms and hyponyms we considered the following aspects. First, we use a single level of hypo/ hyperonymy, to avoid overly altering word meanings. For example, *house* is a first-level hyponym of *building* (thus that relationship is considered by the algorithm) and a third-level hyponym of *thing* (thus that relationship is not considered). Second, we considered total synonymy, where two terms are synonyms if they are interchangeable in the same context without affecting the semantics. For example, *land* and *ground* are total synonyms since they are semantically interchangeable.

Identifiers compatibility calculation At this stage, the following information is available to calculate name compatibility:

- *terms*: total terms between both terms lists.
- *exact*: Number of identical terms.
- *syn*: Number of synonyms among the terms in both lists.
- *hyper*: Number of hyperonyms.
- *hypo*: Number of hyponyms.

Using these values as input the name compatibility is calculated according to Formula 3. Notice that hyperonyms and hyponyms are given a weight w with $0 < w < 1$. In this work, we empirically adjusted $w = 0.5$ – i.e., first level hyponyms and hyperonyms are given half the weight of exact terms and synonyms, since the former imply a slightly different semantics between terms than the latter. Then, following the same rationale, it is possible to consider n-level hypo/hyperonyms, where each level is given half the weight of the previous one: $w_n = w_{n-1} * 0.5$ – i.e., $w = 0.5$, $w_2 = 0.25$, $w_3 = 0.125$, etc. The values of $nameComp$ range between 0, with no identical terms, synonyms or hypo/hyperonyms; to 1, with all terms being identical or synonyms.

$$nameComp = \frac{exact + syn + w * (hyper + hypo)}{terms} \quad (3)$$

[11] http://projects.csail.mit.edu/jwi/

3.2 Return type

Primitive return types only imply a structural assessment. However, for complex return types, the semantic aspects of identifiers should also be considered – i.e., name of the complex type and its fields. Thus, for a pair of operations op_R and op_S with complex return types, the structural and semantic assessment aggregates two values:

- Fields compatibility (*fieldsComp*). Fields in complex types are defined by a name and a type. Thus they are processed similarly to parameter lists (which also are characterized by a name and a type). The procedure is detailed in Section 3.3.
- Name compatibility (*nameComp*). The name of complex return types is assessed through the identifiers evaluation algorithm defined in the previous section.

Finally, the compatibility value between return types of a pair of operations ($op_R \in I_R, op_S \in I_S$) is calculated according to Formula 4. When the return types are both primitive, the calculation is done according to Table 1, thus $retComp = -R$. For complex return types we add 1 to the *nameComp* value, to avoid generating an incompatibility for the return type (*retComp = 0*) only based on the semantics of the identifiers. The case of a complex return type and a primitive return type is considered as not compatible.

$$retComp \begin{cases} -R & when\ comparing\ primitive\ types \\ fieldsComp* & when\ comparing \\ (1 + nameComp) & complex\ types \end{cases} \tag{4}$$

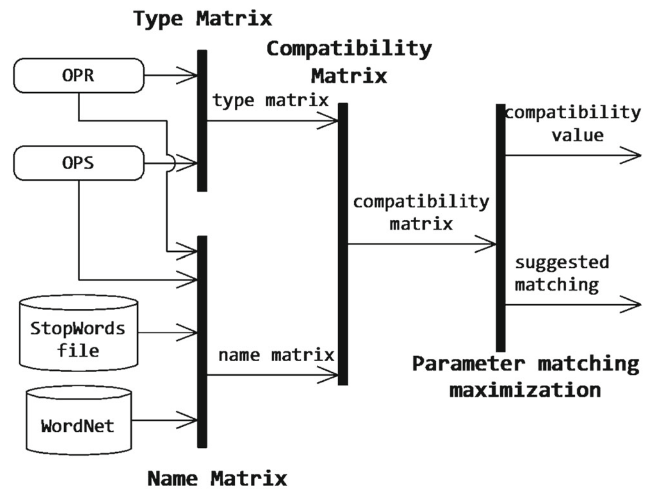

Fig. 4 Steps of the Parameters Matching algorithm of the hybrid service selection approach

3.3 Parameters list

The structural service selection approach presented in Section 2.1 only considers types from operation parameters, but not their names. Moreover, the structural and semantic parameters list evaluation consists of calculating three matrices with parameters information: Type Matrix, Name Matrix and Compatibility Matrix, as shown in Fig. 4. For the three matrices, the cell m_{ij} represents the compatibility value between the *i-th* parameter of the required operation op_R and the *j-th* parameter of an operation op_S from a candidate service. The main steps of the algorithm are described below.

The Name Matrix stores the compatibility values between the name of each parameter from op_R and the

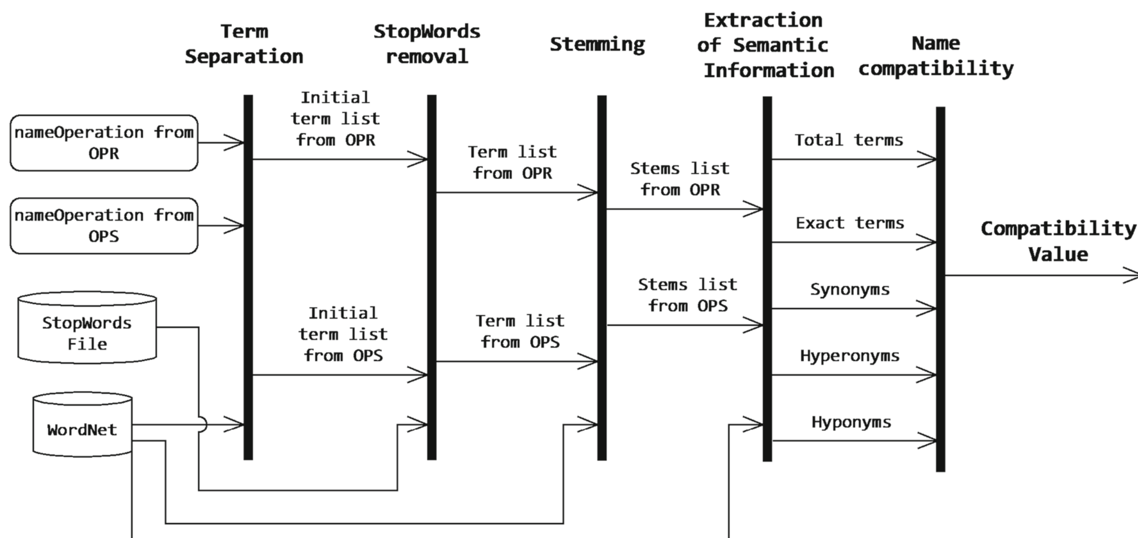

Fig. 3 Steps of the identifiers evaluation agorithm

name of each parameter from op_S. The cell n_{ij} from the name matrix N contains the compatibility value between the name of the *i-th* parameter of op_R and the name of the *j-th* parameter of op_S. This value is the result of applying the name compatibility algorithm presented in Section 3.1. For implementation purposes, we used the Paranamer library[12] to access parameter names of Java methods from compiled files.

The Type Matrix stores the similarity between all pairs of parameter types from operations op_R and op_S. The notions of type equivalence and subtype equivalence implemented in the structural service selection approach (presented in Table 2) are used to assess parameter types. Then, a cell t_{ij} from the type matrix T contains the compatibility value between the type of the *i-th* parameter of op_R and the type of the *j-th* parameter of op_S. These values complement the name compatibility values (explained above) as a weighting factor based on type similarity. Let P_i and P_j be two parameters under analysis, belonging to op_R and op_S respectively. The cells of T can store three different values:

- If Type(P_i) is identical to Type(P_j) then $t_{ij} = 2$. Identical types increase the name compatibility value for the pair of parameters (P_i, P_j) by a factor of two.
- If Type(P_i) is subtype of Type(P_j) then $t_{ij} = 1.5$. Subtypes increase the name compatibility value for the pair of parameters (P_i, P_j) by a factor of 1.5.
- Otherwise, $t_{ij} = k$, with $0 < k < 1$. Non-related types affect negatively the name compatibility value for the pair of parameters (P_i, P_j), by a factor between 0 and 1 (not included).

Moreover, the goal of the compatibility matrix is to store the compatibility value of all pair-wise combinations of parameters from operations op_R and op_S. The compatibility matrix is generated from the type matrix and the name matrix, thus considering both structural and semantic aspects from parameters. The cell c_{ij} from the compatibility matrix C stores the product between cells t_{ij} and n_{ij} from the type matrix and the name matrix respectively. Thus,

$$c_{ij} = t_{ij} * n_{ij}$$

After calculating the compatibility matrix, the best pair-wise combination of parameters must be selected, i.e., the combination of parameters from op_R and op_S that maximizes their compatibility. Each possible matching is computed taking each row from the compatibility matrix (i.e., op_R parameter) and choose a column (i.e., op_S parameter), without repeating columns – i.e., matching each parameter only once. In this way, it is ensured that each parameter from op_R corresponds to one and only one parameter from op_S.

For each possible pair-wise parameter assignment between op_R and op_S the compatibility value is obtained from the corresponding cell of the compatibility matrix. The value of each possible parameter matching is the sum of all pair-wise assignments that compose it. The matching with the highest value will be the most compatible. Such matching is obtained through the Hungarian algorithm.

The matching that maximizes the compatibility value will be recommended as the best matching and then used to calculate parameter compatibility according to Formula 5, where m_{ij} are the selected cells of the compatibility matrix that maximizes the compatibility values, and α is a configurable constant value that penalizes the number of unused parameters from op_S.

$$paramComp = \frac{\sum m_{ij}}{\#(op_R Param)} - \alpha * \#unusedParamsOp_s \quad (5)$$

The value for the constant α can be tuned to penalize the number of unused parameters in the op_S operation from the candidate service. Each of them represents certain information loss and squandered processing, since it requires adaptation logic to fulfill such parameter when calling the service. Thus, we penalize that situation by the α value, which is initially set to 0.5. With two or more unused parameters, this penalization increases the chance that the candidate service is not compatible. Overall, the outcome of this step is a list of ordered pairs (P_i, P_j) where the first component is a parameter from op_R and the second component is the op_S parameter which maximizes the compatibility between the operations. The advantages of this detailed analysis of parameters lists is twofold. First, it exposes the best possible parameter pairs in terms of interface compatibility. Second, these parameter pairs suggest the needed parameter transformations to adapt the client application, considering the required interface to the interface exposed by the candidate service. For example, given a pair of operations [op_R(`int` p_a, `int` p_b); op_S(`long` p_c, `double` p_d)], the outcome of the parameters evaluation is a list of the potentially compatible parameters in the form [(p_a, p_d); (p_b, p_c)]. Since the parameter types in op_R are both `int`, this suggests that (p_a, p_d) and (p_b, p_c) have compatible names, and the required transformations consist on casting the `int` type of p_a to a `double` type, and the int type of p_b to a `long` type. Finally, with these pairings and transformations, the required operation op_R could be successfully adapted to safely call the service's operation op_S.

[12] https://github.com/paul-hammant/paranamer

3.4 Exceptions

The structural service selection method defines equivalence conditions for exceptions (Table 1) that were also extended by adding lightweight semantics. Firstly, any required operation op_R may define *default* exceptions – i.e., through the Exception type – or *ad-hoc* exceptions. The ad-hoc exceptions defined by developers or third-party libraries should be accessible to analyze their information. Likewise, an operation op_S from a Web Service may define a *fault* as a message with specific parts (fields).

Thus, similarly to parameters evaluation, exceptions evaluation consists of calculating three matrices: type matrix (T), which allocates the likely correspondences between exception types (according to structural conditions defined in Table 1) from op_R and op_S, name matrix (N), which allocates the likely correspondences between exception names, and compatibility matrix (C), which allocates collated information from names and types. For the three matrices, the cell m_{ij} represents the compatibility value between the *i-th* exception of the required operation op_R and the *j-th* exception of an operation op_S from the candidate service. After calculating the compatibility matrix, the best pair-wise combination of exceptions must be selected. The matching with the highest value will be the most compatible. Again, this matching is obtained through the Hungarian algorithm. Finally, exceptions compatibility is calculated according to Formula 6.

$$excepComp = \frac{\sum m_{ij}}{\#excepOp_R} \tag{6}$$

3.5 Adaptability gap

The hybrid selection method allows us to calculate a value that reflects the required adaptation effort to integrate a candidate service with interface I_S into a client application with required interface I_R. This value is called *Adaptability Gap*, and it is calculated according to Formula 7, where N is the number of operations in I_R and *adapMap* is the best value among the equivalence values $adapValue(op_{R_i}, op_{S_j})$. Notice that the adaptability gap extends the compatibility gap value introduced in Section 2.1, by considering structural and semantic information gathered from required interfaces and candidate service interfaces. This information includes the analysis of semantic aspects from each identifier defined in service operations through WordNet, in particular operation names, complex return types, parameter names and exception names. Also, the adaptability gap is a comprehensive value that indirectly foresees the adaptability effort to integrate the candidate service into the client application.

$$adapGap(I_R, I_S) = \frac{\sum_{i=1}^{N} max(adapMap(op_{R_i}, I_S))}{N} \tag{7}$$

The adaptability value *adapValue* between an operation op_R and a potentially compatible operation op_S is calculated upon the structural and semantic compatibility values obtained for the signature elements of operations (identifiers, return type, parameters and exceptions), according to Formula 8. In particular, *nameComp* is the value obtained from the identifiers evaluation algorithm with operation names from op_R and op_S as input.

$$adapValue(op_R, op_S) = nameComp + retComp + parComp + excComp \tag{8}$$

The lower the adaptability gap value between a required interface and a candidate service, the less the effort a developer has to invest into modifying (adapting) his/her client application to invoke the service.

4 Experiments

In this section, we describe the experiments carried out to evaluate the hybrid service selection method presented in Section 3. For the first experiment (Section 4.1), we generated two experimental scenarios by using two data-sets to populate the EasySOC service registry. EasySOC maps queries and services onto vectors in the Vector Space Model (VSM) and uses a query-by-example search engine (Crasso et al. 2011a). The first data-set comprised 1,985 services from the literature (Mateos et al. 2011; Heß et al. 2004; Al-Masri and Mahmoud 2007). The second data-set comprised 1,239 services crawled from the Mashape.com public API repository.[13] Then we compared the retrievability of relevant services according to different service selection approaches (structural, semantic, hybrid, and the Stroulia algorithm).

For the second experiment (Section 4.2), we populated two versions of the EasySOC service registry with the dataset of 1,239 services crawled from Mashape.com to assess the retrievability of relevant services of the hybrid service selection method. Details of the experimental validation are presented in the following sections.

[13] http://www.mashape.com

Table 5 Service selection approaches compared in the experiments

Approach	Structural assessment	Semantic assessment
Structural Interface Compatibility	Types in parameters, return, exceptions	None
Semantic Interface Compatibility	None	Information content of terms in identifiers
Hybrid Interface Compatibility	(Complex) types in parameters, return, exceptions	Similarity of identifiers through WordNet (synonyms, hyponyms, hyperonyms)
Stroulia Algorithm	Recursive similarity analysis of types, messages, operations and services	None

4.1 Experiment 1 – Benchmarking service selection approaches

Methodologically, the first experiment consisted in the following steps. First, a data-set of WSDL specifications was used to populate the EasySOC service registry. A set of 531 syntactic queries was automatically generated from 65 services, randomly selected as relevant services for the data-set and then used to inquiry the service registry. Then, the queries were mutated to add structural information (return types, parameters) by means of mutation operators applied to the original operations from relevant services (see Section 4.1.1). This allowed us to represent each query as a fully-described single-operation Java required interface (I_R). Then, we executed several service selection approaches, described in Table 5, to compare the queries (required interfaces) against the candidate services in the data-set. Assessed selection approaches include the structural, semantic and hybrid approaches described in this paper and a foundational algorithm in the field of Web Services discovery and selection: the Stroulia algorithm (Stroulia and Wang 2005).

Finally we replicated the experiment following the same steps with the second data-set crawled from Mashape.com, generating another experimental scenario. Clearly, the reason of using more than one service data-set is to make the results as much data-set independent as possible.

For this experiment, Java interfaces of candidate services (I_S) were generated through the SoaMembrane API. SoaMembrane provides facilities to parse WSDL specifications capturing almost all of their information. As discussed earlier, other popular libraries such as Axis2 and EasyWSDL either introduce higher noise in the generated Java interface, or miss key information of identifiers and types.

To analyze the results of the experiment (Section 4.1.3), we used a suite of metrics from the IR field – particularly Recall and Normalized Discounted Cumulative Gain (NDCG).

The Stroulia algorithm This work provides a suite of methods to assess Web Service similarity. Given a textual description of the desired service, a structural IR-based method identifies and ranks the most relevant WSDL specifications according to certain structural notions. If a (potentially partial) specification of the desired service behavior is also available, this set of candidates can be further refined by a semantic matching step. For this experiment, we implemented the Stroulia algorithm according to the guidelines given in (Stroulia and Wang 2005).

The structure matching method involves the comparison of the operation sets offered by services based on the structure of the operations input and output messages, which, in turn, is based on comparing the types communicated by these messages. The overall process starts by comparing types in two WSDL specifications. The result of this step is a matrix representing the matching scores, i.e., the similarity degree of all pair-wise combinations of source and target types. The next step in the process matches service messages. The result is a matrix representing the matching scores of all pair-wise combinations of source and target messages, which are computed on the basis of parameter lists similarity in terms of their types. The third step of the process matches service operations. The result of this step is a matrix representing the matching scores of all pair-wise combinations of required and candidate (retrieved) operations. The overall score for two services is computed as the maximal pair-wise correspondence of their operations.

4.1.1 Query set generation

The first step to generate the queries was to extract all the operations defined by the relevant services (531 in total). Then, an interface mutator[14] (developed by our group) was used to generate different structural information about the return type, parameters and exceptions according to several

[14] http://code.google.com/p/querymutator/

mutation operators, by using as input the original structure of the aforementioned operations. Thus, queries consisted in the original operation names and structural information generated by mutating existing return types, parameters and exceptions. Queries for the semantic algorithm imply a special case since they do not include structural information. Thus, an ad-hoc mutation operator was defined for this particular case.

Interface Mutation is a technique to evaluate how well the interactions between various units have been tested (Delamaro et al. 2001). In this experiment, these units are candidate services and required interfaces (queries). Interface Mutation extends mutation testing and is applicable, by design, to software systems composed of interacting units. In a call from a unit *C* to a unit *S*, data can be exchanged in four ways: (1) Data can be passed to *S* via input parameters – i.e., passed by value; (2) Data can be returned to *C* through return values – as in `return` commands in *S*; (3) Data can be passed to *S* and/or returned to *C* through input/ output parameters –i.e., passed by reference; and (4) Data can be passed to *S* and/or returned to *C* through global variables.

In the service integration scenario, let *C* be the client component invoking a service and *S* the invoked service, thus data can be exchanged according to (1) and (2). (3) and (4) are not applicable since global variables and parameters by reference are not shared between a service and its client. When applying Interface Mutation, unlike traditional mutation, the syntactic changes are made only at the interface related points or connections between units (Delamaro et al. 2001). In this experiment, such points are the parameters, return types and exceptions from the queries, which act as the client interface *C*.

In a previous work, we manually expanded the syntactic queries to include structural information (Garriga et al. 2013). In the present experiments, the interface mutation procedure automatizes query generation and (probabilistic) mutation. This means, each mutation operator has a configurable constant p, with $0 \leq p \leq 1$ that indicates the probability of applying that operator to mutate a particular operation. This emulates the (potentially partial) specification of the desired functionality defined by developers during the service discovery process (Stroulia and Wang 2005). After applying the mutation operators, each mutated query can be encapsulated as a Java (required) interface with only one operation, which acts as the required interface (I_R). The mutation operators were defined to generate queries prior to the execution of the three service selection mechanisms, as follows:

Encapsulation A random number of parameters in the parameter list of the interface are encapsulated as fields of a new complex type. The name of the complex type is the concatenation of the name of the operation and the word "Request" as typically done by WSDL generator APIs such as Axis2. The name of the complex parameter is a concatenation of the encapsulated parameters. This operator is analogous to the *Introduce Parameter Object*[15] refactoring proposed by Fowler (Fowler 1999), which is popular in object-oriented programming.

Table 6 Built-in direct supertyping for java

Type	Direct Supertype
`byte`	`short`
`short`	`int`
`int`	`long`
`long`	`float`
`float`	`double`

Example Let us consider the operation `AddUser (String userName, String password, int sessionId)`. The two first parameters can be encapsulated as `AddUser(AddUserRequest userName_password, int sessionId)`, where the complex type `AddUserRequest` is composed of two `String` fields – `userName` and `password`.

Flatten A random number of complex type parameters in the parameter list of the interface are flattened, generating as many parameters as fields in the complex type. The resulting parameters can be primitive or complex type, according to the types of the fields in the original type.

Example Let us consider the operation `AddUser (UserData userData, int sessionId)` with the complex type `UserData` containing two `String` fields `user` and `password`. The mutated interface after applying the flatten operator could be `AddUser(String user, String password, int sessionId)`.

Upcasting The return type and/or a random number of parameters of numeric types are upcasted to a direct numeric supertype in the Java language, according to Table 6. This operator is similar to the *Encapsulate Downcast*[16] refactoring.

Example Let us consider the operation `void AddUser(String userName, String password, int sessionId)`. The `int` parameter can be upcasted to `long`, generating `void AddUser (String userName, String password, long sessionId)` as the mutated signature.

[15] http://www.refactoring.com/catalog/introduceParameterObject.html
[16] http://www.refactoring.com/catalog/encapsulateDowncast.html

Wildcard Supertype The return type and/or a random number of parameters can be casted to the wildcard supertype `String`, including `void` return types. As stated earlier, the `String` type is generally used as a wildcard in practice to allocate different kinds of data. This is a special, naïve case of the Fowler's *Layer Supertype*[17] pattern. This operator is similar to the Upcasting operator, but upcasting only applies for numeric types, and both operators can be applied with different probability.

Example Let us consider the operation `void AddUser(String userName, String password, int sessionId)`. The Wildcard Supertype operator could generate `String AddUser(StringuserName, String password, String sessionId)`, where both the return type and the parameter `sessionId` have been casted to `String`.

Concatenation The semantic service selection method implies a special case, as it does not consider structural information in the queries. In this case, we defined an ad-hoc mutation operator, which was configured with probability $p = 1$, to generate queries based only upon semantic information. In this case, queries were constructed by obtaining terms from identifiers of original operations using term separation (Section 3.1). Then, the concatenation operator creates a sentence by concatenating the operation name with the parameter names, which are separated by commas, and using the word "with". Finally, the "query" sentence and the "candidate" sentence can be compared using the semantic similarity metric, which was explained in Section 2.2.

Example Let us consider the operation `void AddUser (String userName, String password, int sessionId)`. The Concatenation operator generates a sentence in the form "Add user *with* user name, password, session Id".

4.1.2 Experiment execution

To execute the experiment, two scenarios were defined by combining:

- The two experimental data-sets,
- Original queries (considering only operation names) extracted from the randomly selected relevant services,
- The mutation operators, applied randomly over the signature of original operations to generate queries with unique structural characteristics,
- The EasySOC service discovery registry,
- The structural, semantic and hybrid service selection approaches, and the Stroulia algorithm as the alternatives under analysis.

For this experiment we considered an initial ranked list of the first 10 candidate services retrieved per query by the EasySOC discovery registry. When the relevant service is not retrieved by the discovery registry, it is given the 11th position as input for the other algorithms. The service selection approaches and the Stroulia algorithm were then executed comparing the Java interfaces of candidate services (I_S) and the mutated queries as required interfaces (I_R). Thus, the goal of this experiment is to analyze how the different selection approaches could improve the visibility and retrievability of suitable candidate services in a result list.

Example For example, let us consider the query `getUser` extracted from service `AccountingService`. Thus, AccountingService is the relevant service for that query, and should be retrieved in the topmost positions. Then, the results list (ordered by position) retrieved by the EasySOC registry could be:

{(1, `VomsAdminService`),
(2, `VomsTrustedAdminService`),
(3, `Service6.Accounts`),
(4, `Service7.Accounts`),
(5, *AccountingService*), ... }

As can be seen, the relevant service was retrieved in the fifth position. The first four services also provide an operation named `getUser`, which satisfies the query. Services in position 1 and 2 are from the data-set of relevant services, and the following two services are from the noise data-set. Through the alternative service selection methods and the Stroulia algorithm, this list is reordered considering structural and/ or semantic information both from the mutated query and the services in the list. Thus, the relevant service could be ranked in a better position in the list – e.g., among the first three. In this case, the reordered list obtained by a certain service selection method could be as follows, where the relevant service is in the second position:

{ (1, `VomsAdminService`), (2, AccountingService),
(3, `VomsTrustedAdminService`),
(4, `Service6.Accounts`),
(5, `Service7.Accounts`), ... }

4.1.3 Results

For the two experimental scenarios, we queried the EasySOC registry with the syntactic queries. Then, we

[17] http://martinfowler.com/eaaCatalog/layerSupertype.html

executed the structural, semantic and hybrid service selection methods and the Stroulia algorithm using as input the mutated queries plus the initial service candidate list returned by the registry. We measured and compared results according to two well-known IR metrics, namely Recall and Normalized DCG. Formally, Recall is defined as:

$$Recall = \frac{relevant\ retrieved}{total\ relevant}$$

In particular, for each query in this experiment the numerator of the Recall formula could be 0 (when the relevant service is included within the retrieved results) or 1 (when the relevant service is not included within the retrieved results). The denominator (total relevant services per query) is always 1.

Then we calculate the average Recall for the 531 queries in each position of the results list, considering a window size of 10 results:

$$Average\ Recall = \frac{\sum relevant\ retrieved}{531}$$

On the other hand, the DCG is a measure for ranking quality (Kessel and Atkinson 2016). It measures the usefulness (gain) of an item based on its relevance and position in the provided list. As the DCG increases, the ranked list of results is better. Formally, DCG is defined as:

$$DCG = \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$$

where p is the size of the candidate list, and rel_i indicates if the candidate retrieved in the *i-th* position of the list was relevant. The DCG values for all queries can be averaged to obtain a measure of the average performance of a ranking algorithm, named Normalized DCG (NDCG).

Coming from the IR area, these two metrics have been broadly used in the context of Web Service discovery and selection (Fragoso et al. 2006; Rodriguez et al. 2010). Other well-known metrics such as *Precision* and *F-measure* are not computed for this experiment, due to their particular characteristics. Having exactly one relevant service per query implies that the outcome at each position in the result list is binary: either the relevant service is retrieved, or it is not. Under this condition, Precision and Recall are analogous. Similarly, as F-measure is calculated as the weighted harmonic mean of *Precision* and *Recall*, its calculation gives no information in this context.

A service containing the needed operation (one per query) was selected and associated to the query as the relevant one in order to evaluate the results. Finally, an average of each metric was generated over the total number of queries.

Scenario 1 - Academic data-set In *Scenario 1*, the EasySOC registry was populated with the first data-set of 1,985 services and queried with the 531 queries. Figure 5 depicts the Recall values for the original service registry (EasySOC), the service selection methods and the Stroulia algorithm, by smoothing these results using Bézier curves.

Results show that applying any structural-aware service selection method improves the Recall up to 34 % for the first position of the results list. In the figure, lines above the solid line improved original results and the lines under the solid line decreased original results, for the given positions of the results list.

In this scenario, the hybrid approach outperformed the other alternatives for the first positions of the list, with Recall results up to a 86 % for the third position. This means that, in average, 8.6/10 queries will return the relevant service between the first 3 positions in the list when using the hybrid approach. Although Recall tends to converge when n approaches to 10, the improvements in the first positions are significant since, in general, users first inspect and hence potentially select higher ranked search results, regardless of their actual relevance (Agichtein et al. 2006).

Figure 6 summarizes the results for NDCG in *Scenario 1*. The figure shows that structural and hybrid approaches improved original NDCG (7 % and 19 % respectively). The Stroulia algorithm presents a NDCG 2 % lower than original. Semantic service selection presents a NDCG 22 % lower than the original. We relate this phenomenon with the underlying rationale of the service registry, since EasySOC considers for discovery structural information of services by comparing terms in elements that are present in the query and the candidate services – e.g., operation names or input/output messages. In the context of this experiment, EasySOC only compares terms in syntactic queries against

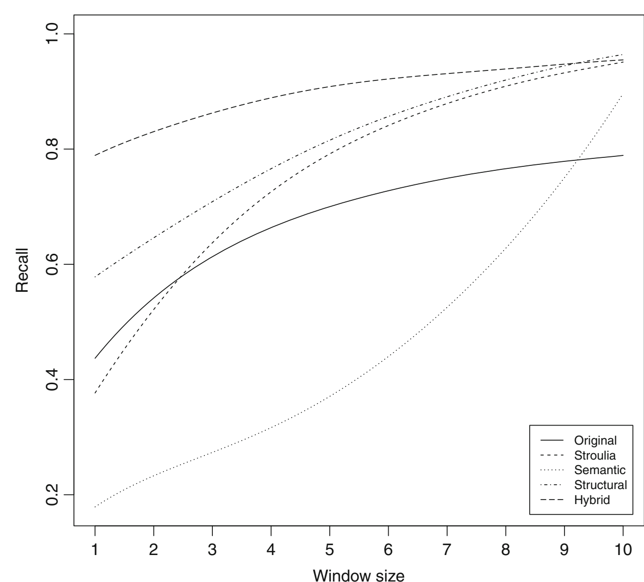

Fig. 5 Recall for *Scenario 1*

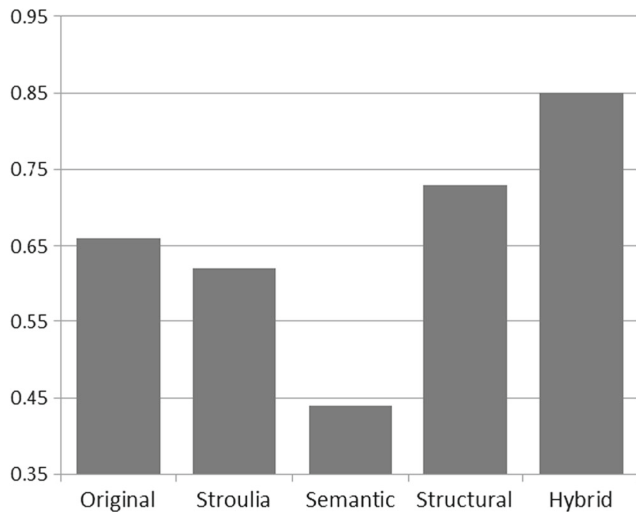

Fig. 6 NDCG for *Scenario 1*

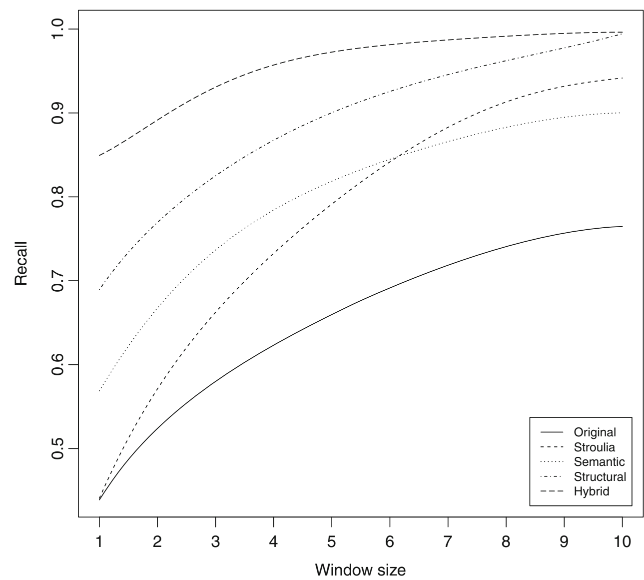

Fig. 7 Recall for *Scenario 2*

operation names of candidate services in the two data-sets. Thus, a purely semantic service selection method considers less information than the original discovery mechanism, and then retrieves less services – i.e., more queries fail in retrieving their relevant service, decreasing Recall and NDCG.

Scenario 2 - Mashape data-set In *Scenario 2*, the EasySOC registry was populated with the second data-set of 1,239 services crawled from Mashape.com, and then queried with the 531 queries. Figure 7 depicts the average Recall-at-n values (i.e., Recall with a window size of n) for the original service discovery registry (EasySOC), the service selection methods and the Stroulia algorithm, again by smoothing these results using Bézier curves.

For this scenario, results show that applying any service selection method improved the Recall up to 41 % for the first position of the results list. In particular, for each service selection method it can be noticed that hybrid, structural and semantic service selection improved original results (by a +41 %, 25 % and 12 % respectively for the first position of the list), and the Stroulia algorithm presents a slightly lower (-1 %) Recall.

In this scenario, the hybrid approach outperformed the other alternatives for the first positions of the list. Figure 8 summarizes the results for NDCG in the second scenario. The figures show that all alternatives for service selection improved original NDCG results up to a 25 %.

4.2 Experiment 2 - Hybrid approach with manual query generation

In this experiment, we evaluated the Hybrid selection approach using manually generated, multi-operation queries and the Mashape.com data-set to populate the EasySOC service discovery registry. In this case, rather than generating queries automatically by mutation, two groups of expert software engineers manually generated the queries by randomly selecting operations from services in the data-set. The procedure is detailed below.

4.2.1 Query set generation

The WSDL specifications of the whole data-set were given to two teams of software engineers. The teams consisted of researchers and PhD. students, both with experience in the industry. Each team randomly selected 15 services as target services. Then, these teams of experts exchanged their selected services. Following, the queries were generated

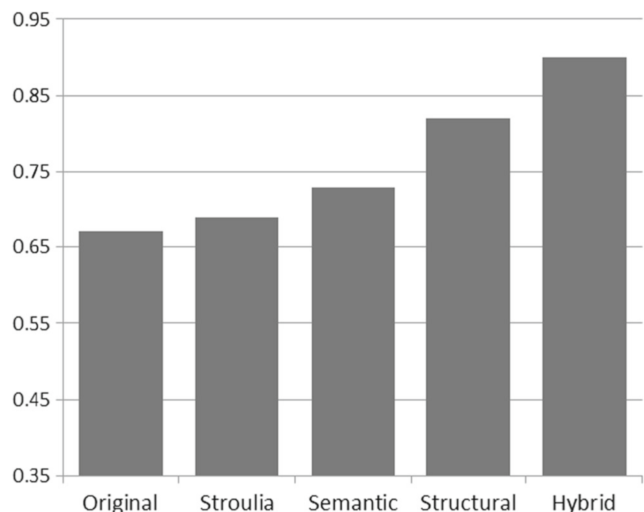

Fig. 8 NDCG for *Scenario 2*

from the 15 services that each team received (selected by the other team). For this, the engineers completely rewrote the signature of the operations in the selected services. The engineers were asked to preserve the intended semantics of the operations while altering the identifiers (i.e., names from operations, parameters and fields in complex types, among others) and data types (in parameters, return and fields in complex types). Then, they splitted these operations in groups of three operations, to generate the queries, since in practice an average Web Service presents three operations (Kil et al. 2006). Thus, each query consisted in up to three related operations, in contrast to the single operation queries in Experiment 1. Each query was associated with the original service from which it was generated as its relevant service (thus having a gold-standard for evaluation).

4.2.2 Experiment execution

In this experiment, we defined the execution scenario as follows. We used two versions of the EasySOC service discovery registry as baseline: the VSM-based original version (Crasso et al. 2008) and a recent version adding query expansion techniques (Crasso et al. 2014). Both versions of the registry were populated with the Mashape.com data-set, which comprised 1,239 services. From the 42 queries with multiple operations generated by experts, a corresponding set of syntactic queries was generated as input to the discovery registries, by concatenating the operation names in each expert query, using the space character as separator. These syntactic queries were then used as input to the service discovery registries. Finally, we performed the hybrid service selection procedure upon the original results obtained from the registries.

4.2.3 Results

For this experimental scenario, we queried both versions of the EasySOC registry with the syntactic queries. Then, we executed the WordNet-JWI implementation of the Interface Compatibility procedure with the experts' queries. Finally, we compared the results against the gold-standard according to the Recall metric.

Figure 9 depicts the average Recall-at-n values for the two versions of the service discovery registry EasySOC and the Hybrid Selection method. The latter increases the Recall results between 19 % and 34 % for the first three positions of the window (i.e. $n \in [1, 3]$).

In this experiment, we used the Mashape.com data-set with a significantly different query generation approach that introduced two teams of software engineers to manually generate multi-operation queries. Results with this configuration confirms the insights of the previous experiments, showing that the hybrid selection method substantially improves original results for Recall in the first positions of results lists.

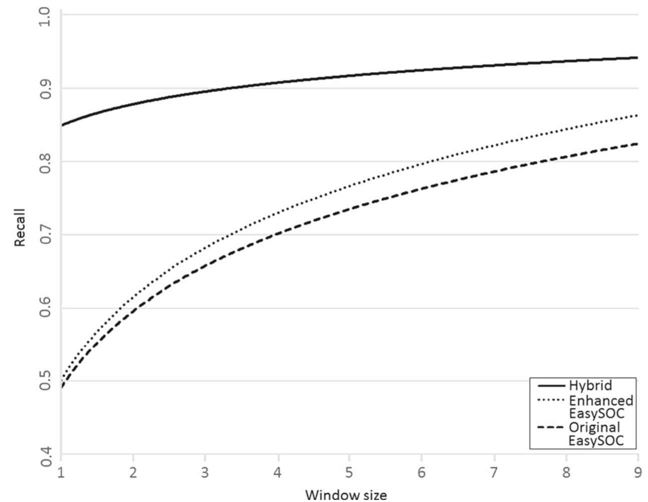

Fig. 9 Recall for Experiment 2

4.3 Threats to validity

In this section we discuss the threats to validity for the experimental procedures and results presented along Section 4.

Internal Threats to validity EasySOC considers structural information of services by comparing terms that are present in the query and the candidate services – e.g., operation names or input/output messages. This could introduce a bias in favor of structural mechanisms, since certain structural information is considered to compare queries against candidate services from the initial discovery step. Thus, a purely semantic service selection method is disadvantageous in this context, since it assesses less information than the original discovery mechanism, and then retrieves less services – i.e., more queries fail in retrieving their relevant service, decreasing Recall and NDCG. However, improved service discovery registries are commonplace nowadays (Crasso et al. 2011b), being a baseline for selection. A possible solution is to combine the information obtained through the semantic approach with a certain structural support – i.e., an hybrid approach.

In addition, the Stroulia-based algorithm did not present significant improvements over original results. The reason for this is twofold. First, EasySOC already uses some concepts of the VSM to represent Web Service descriptions and queries, making the EasySOC registry more efficient with respect to traditional discovery methods, as mentioned in the previous paragraph. Thus, the advantages of using VSM

in the Stroulia-based algorithm are overlapped in the service discovery step of the experiment. Second, to perform the experiment, we implemented the Stroulia algorithm according to the guidelines given in (Stroulia and Wang 2005). It is possible that our implementation of the Stroulia algorithm introduced slight differences w.r.t. the original, which may decrease the performance in terms of retrievability. However, the results obtained in this experiment are similar to those published originally by the authors, thus it is arguable that our implementation did not introduce a significant variation. Additionally, as stated by their authors, similarity assessment methods in the Stroulia algorithm are neither precise nor robust enough to discover the desired service without developer's intervention.

One limitation of this experiment is that, as we mentioned earlier, the second data-set of WSDL documents was automatically generated by a crawler using the information extracted from Mashape.com. In this process, certain identifiers were automatically generated from others. For example, message names were generated using operation names, concatenated to "input/output message" . Therefore, these WSDL documents present two particular properties: on the one hand, they convey less semantic information than the WSDLs from the first data-set, as certain identifiers are a copy of others. However, on the other hand, they are more cohesive as different parts of a WSDL use very similar identifiers. Fewer terms improve the efficiency of the semantic approach, since the Hungarian algorithm receives as input a smaller matrix of terms. This contributes to a better semantic similarity results for Scenario 2 – as shown in Figs. 5 and 6, since the combinations of identifiers to be analyzed are substantially reduced.

It is worth mentioning that a possible threat to the validity of an experiment involving people, such as the one described in Section 4.2, is demand characteristics (Orne 2009). Demand characteristics result from cues in the experimental environment or procedures that lead participants to make inferences about the purpose of the experiment and to respond in accordance with (or in some cases, contrary to) the perceived purpose. Software engineers are inherently problem solvers. When they are told to perform a task, the majority will try to figure out what is expected from them and perform accordingly. Demand characteristics influence a participant's perceptions of what is appropriate or expected and, hence, their behavior (Kirk 1982). In the context of this experiment, the participants may figure out the notion of structural and semantic "similarity" or "difference" among WSDLs. Thus, the participants may generate queries that are too similar to the given WSDL documents, without substantially altering data types identifiers. In order to reduce the learning effect, only a subset of the WSDLs are assigned to each team of experts, and then these WSDLs are exchanged between teams. Thus, this mitigates the aforementioned deduction. Another experimental threat may be the language: even when all participants had some formal English knowledge, they were not native English speakers. The data-set and queries are implemented using concepts of the English language, which could generate a slight noise to understand the WSDL documents and generate mutated queries.

External threats to validity For this experiment, the hybrid service selection method outperformed the other approaches. Considering that the selection of candidate services is performed after a discovery process, this confirms our hypothesis that increasing the visibility of most suitable candidates in a list of previously discovered services may ease the development of client applications. It is important to notice that results can depend on the data-set and queries used, and cannot be merely generalized to other experimental configurations. However, the empirical validation is a common practice for the knowledge area of Web Service discovery and selection. Similar experimental configurations and data-sets were used not only in foundational papers (Stroulia and Wang 2005; Heß et al. 2004; Crasso et al. 2008) but also in recent papers of the field (Mateos et al. 2015b; Tibermacine et al. 2013). For interested readers, our experimental tools[18] and data-sets are available online. The data-set for the Scenario 1 in Experiment 1 was originally published in (Heß et al. 2004) and is still available.[19] The data-set of Scenario 2 is based on the APIs published in the Mashape.com repository[20], which we crawled to generate the corresponding WSDL documents.[21] The interface mutator is also available online.[22]

5 Related work

The idea of accelerating software development by reusing existing code wrapped up in existing software artifacts has been explored in the past. Years ago, Component-Based Software Engineering (CBSE) emerged as the most promising solution to build systems from reusable pre-existing commercial off-the-shelf (COTS) components, reducing development costs and shortening time-to-market (Kung-Kiu and Zheng 2007). However, in this context, it becomes impossible to specify requirements and architectures without asking if the marketplace provides COTS that can be

[18] https://goo.gl/qY5dkb

[19] https://goo.gl/ClZ4ko

[20] https://market.mashape.com/

[21] https://goo.gl/eSl6ZW

[22] http://code.google.com/p/querymutator/

seamlessly integrated to it (George et al. 2008). Thus, COTS selection became particularly important and challenging, and a well-defined, repeatable and (semi-)automated selection process was mandatory. Different approaches strive to solve the component selection problem, notably repository-driven tools that gathered information in two different ways (Hummel et al. 2008): reactive, in which a user actively browses for components, and proactive, in which a tool monitors the user's activities and then suggests potentially useful components according to the context.

From an architectonic perspective, a service-oriented application can be seen as a component-oriented application that is built by composing two different types of components: internal, which are locally embedded in the application, and external, which are statically or dynamically linked to services (Zimmerman et al. 2005). These components may fit into the definition of off-the-shelf (OTS) components (for example, Java Beans), according to the CBSE paradigm, but also may be shaped as common software pieces in an architecture.

The approach presented in this work is based on a previous approach (Flores and Polo 2012; Flores et al. 2010) originally developed to select the most suitable off-the-shelf (OTS) software components as a solution for substitutability of component-based systems. As discussed, Web Services are a special case of software components (Kung-Kiu and Zheng 2007), thus we were able to redefine the previous component selection approach for Web Service selection.

5.1 Identifiers analysis

Identifiers in source code, interfaces and service descriptions convey key information about the intended semantics of an operation, a parameter or a data-type. Programming conventions advise developers to choose meaningful identifier names. Particularly, class names should be descriptive nouns (Butler et al. 2011). Meanwhile, advanced practitioner texts suggest a dedicated approach to identifier naming, following a variety of conventions as a result of praxis. In this section we present several approaches for identifiers analysis and their relation with ours.

The work in (Falleri et al. 2010) automatically classifies a set of identifiers in a WordNet-inspired structure called *lexical view*, which leverages the hierarchical relations between terms in identifiers. The process consists of several steps: term separation, classification into lexical categories or part-of-speech (POS) tagging, application of syntactical rules of English to determine dominant words (those that convey the meaning of the identifier), extraction of implicit important words and organization of the identifiers in a lexical tree. Several steps are similar to our identifiers evaluation algorithm (Section 3.1). As an advantage, this approach performs POS tagging and applies syntactical rules to disambiguate terms, which is a point to be addressed in our work. However, the approach assumes that identifiers are well formed (i.e., that they follow common naming conventions), which is not always the case – as discussed in Section 3.1. Additionally, constructing a lexical tree inspired in WordNet may generate unnecessary overhead and complexity, as WordNet itself is a complete lexical tree which exposes the relationships among almost all words in the English language.

POS tagging of program identifiers is also addressed in (Gupta et al. 2013) along with a syntactic term separator for identifiers in source code, which takes into account programming naming conventions to understand the regular, systematic ways a program element is named. The naming conventions are adopted from the object-oriented paradigm to identify different grammatical constructions that characterize a large number of program identifiers; e.g., developers combine terms in the grammatical form <verb> + <noun> for denoting an operation name. POS tagging and syntactical term separation are combined to improve each other, using all possible POS tags and grammatical constructions involving combinations of the basic lexical phrases in a prioritized order, to determine the most appropriate grammatical form of the identifier. This work, as the previous one, provides tools that could be combined to our hybrid approach: POS tagging and improved term separation combined to determine semantics of terms in identifiers more accurately.

Finally, the work in (Butler et al. 2011) combines POS tagging and a hierarchical analysis of words to empirically determine Java naming conventions. The origin of words used in class names is tracked within the name of any super class and implemented interfaces to identify patterns of class name construction related to inheritance. Through the analysis of open source projects, authors found both common and project-specific naming conventions. Then, an automated analysis of class names determines if unconventional named classes are candidates for refactoring, and the kind of refactoring needed – i.e., refactor to a common name or a project-specific name. Although the analysis of related identifiers in a hierarchy of classes is interesting, it is restricted only to class names. Such coarse-grained view hinders the usability of the approach, as it would be desirable to analyze class fields and methods. Also, this kind of hierarchical analysis is restricted to certain languages supporting inheritance (e.g., Java) and is not applicable to others.

5.2 Service selection approaches

The surveys in (Kokash 2006; Crasso et al. 2011b) provide a comparative analysis of existing approaches to improve Web Service discovery considering the typical scenario

where users perform queries against a service registry. This is closely related to service selection, since any discovery method which considers structural and/or semantic aspects performs a partial, preliminary selection among candidates in a registry. Moreover, a service has to be retrieved before it can be selected. Particularly, several discovery approaches use IR techniques in an effort to increase precision of Web Service discovery without involving any additional semantic markup. Although such approaches report concrete improvements, they seem insufficient for automatic retrieval if applied without any complementary technique. To cope with this shortcoming, a strategy based on semantics can consist of formal ontology-based methods, which yet involve high service interface and query specification costs making service designers be alienated from their use in practice (Kokash 2006; Garriga et al. 2015). Indeed, one of the main differences is what such approaches consider/ require as service descriptions. Semantic approaches depend on shared ontologies and unambiguously annotated services, whereas IR-based ones depend on (lighter) textual service descriptions. Although those service discovery systems strive to solve the same problem, they may be appropriate in different environments (Crasso et al. 2011b).

The rest of this section present structural, semantic and hybrid service selection approaches. We consider that IR-powered hybrid approaches are both widely applicable and effective, since they are typically based on available functional descriptions of services and lightweight semantics, which are commonplace. For these approaches, we discuss the similarities and differences with our work.

5.2.1 Structural service selection approaches

The problem of service discovery/ selection may be assessed at structural level in different ways. The work in (Plebani and Pernici 2009) relies in UDDI registries enriched with query by example features. The approach in (Wu and Wu 2005) enrich structural description of services with QoS specifications and other desirable properties that are also considered for service selection. Finally, the Stroulia algorithm which was implemented for the experiments in Section 4 presents a structure matching method that recursively compares the structure of operations, i.e., input and output messages, which is in turn based on comparing the types communicated by these messages (Stroulia and Wang 2005).

However, structural approaches do not leverage the semantic information that is always present in service specifications, in the form of identifiers (e.g., parameter names, operation names), documentation and comments. As suggested by the experimental results in Section 4, this decreases the retrievability of potentially compatible candidate services in structural approaches.

5.2.2 Semantic service selection approaches

At the semantic level, service selection approaches can be broadly divided in two research lines: Semantic description languages for Web Services and IR-powered service selection approaches. The main efforts for defining a standardized semantic description language for Web Services are OWL-S, WSMO and WSDL-S. OWL-S (Martin et al. 2007) is an ontology to automatically discover, invoke, compose and monitor Web Services offering particular properties. WSMO (Roman et al. 2005) represents a conceptual model for Web Services that comprises ontologies, Web Services, goals and mediators. Lastly, WSDL-S (Sivashanmugam et al. 2003) is a semantic extension to WSDL. Leveraging these meta-models, many semantic service selection approaches are supported by ontology/ schema matching techniques. These techniques receive as input two ontologies/ schemes (in this case, meta-models of desired and actual Web Services) and produce as output the relationships holding between them (Shvaiko and Euzenat 2013). Different surveys in the topic of semantic discovery and selection (Cabral et al. 2004; Rambold et al. 2009; Noy 2004) show that most approaches exploit similar ideas, although they use different meta-models (mentioned above). For example, the work in (Li et al. 2006) is based upon WSDL-S, where semantically annotated services are published into a UDDI registry and can be dynamically discovered using ontological concepts.

On the other side, IR-powered approaches – which are mainly syntax-based – augment Web Service descriptions and registries with text mining techniques (Crasso et al. 2011b). These techniques enhance the performance of IR-based approaches by exploiting lightweight semantic information (as opposed to the inherently heavier ontology-based semantic information), for example by means of WordNet to assess terms similarity. For example, the semantic-aware version of the algorithm presented in (Stroulia and Wang 2005) was developed upon the Vector Space Model (VSM) (Salton et al. 1975) and WordNet. This version expands queries and WSDL specifications with WordNet relations: synonyms, direct hyperonyms, hyponyms, and siblings.

5.2.3 Hybrid service selection approaches

Hybrid approaches, as the one presented in this paper, combine structural and semantic aspects to assess Web Services for discovery/ selection.

Web Service similarity to find relevant substitutes for failing Web Services is addressed in (Tibermacine et al. 2013). This approach combines weighted lexical and semantic similarity between identifiers (comprising service names, operations, input/output messages, parameters, and documentation) with structural similarity between

message structures and complex XML schema types, assessed through schema matching techniques and a similarity flooding algorithm, representing complex types as labeled directed graphs. As an advantage, apart from considering types and identifiers, this approach also includes documentation to perform the similarity analysis. The results are promising, although the experimental scenario is restricted to only 29 services. As a disadvantage, we believe that a straightforward comparison of complex types similarity can be performed without dealing with the complexity of an XML schema (Garriga et al. 2013). Although schema matching techniques bring better precision, they are costly and depend upon the availability of the schemes, which is not always the case (Bouchiha et al. 2012). In our approach, the analysis of WSDL documents allows supporting the comparison of complex data types exploiting a lightweight semantic basis.

The work in (Cong et al. 2015) presents a service discovery approach based on hierarchical clustering to tackle the problem of linear complexity of traditional approaches (i.e., those that compare queries against each service published in a registry), which is even worse when using semantic matchmaking. A distance function defined upon the distances between service elements in WordNet is used to divide the repository in clusters. Then, the queries are structurally and semantically compared to clustered services using WSDL specifications and (if available), semantic descriptions in OWL-S or WSMO. This approach prioritizes the acceleration of service discovery in terms of time-per-query at the expense of decreasing Precision/Recall (i.e., retrievability of services). Instead, our approach prioritizes retrievability over time consumption. Another disadvantage is that the repository should be re-clustered when new services are added, potentially disrupting retrieved results for identical queries performed before and after re-clustering. Finally, although the approach supports a wide range of service descriptions (including WSDL), the validation is performed using OWL-S descriptions which are richer in terms of their semantic information, but not widely used in practice, thus often unavailable (Funika et al. 2012; Bouchiha et al. 2012).

The Woogle search engine for Web Services is presented in (Dong et al. 2004). Based on similarity search, Woogle returns similar Web Services for a given query. The search engine combines a clustering algorithm for grouping service descriptions in reduced sets of terms, with structural information about service operations and parameters. After that, similarity between terms is measured using a classical IR metric such as TF/IDF. This approach shares the problems of re-clustering from the previous approach. Additionally, the Woogle clustering algorithm heavily relies in self-descriptive identifiers, particularly parameter names. However, as described in (Mateos et al. 2011), ambiguous or meaningless names are a typical anti-pattern when denoting the main elements of a WSDL documents. The considered structural aspects are limited to name comparison of analogous elements in services and queries (e.g., comparing parameter names against each other and not against operation names). Instead, our approach enhances the similarity evaluation by exhaustively assessing both structural and semantic aspects. In addition, not only service and operation level are assessed, but also we evaluate similarity between parameters (types and identifiers), return and complex types. Performance of Woogle seemed competitive in the experiments performed by the authors, which used a public dataset of 431 services. Their results w.r.t. Recall outperformed other approaches having a Recall of 88 % for operation matching, while our hybrid approach averaged 85 %, although these numbers are not directly comparable due to the different data-sets and experimental configurations. Besides, Woogle allows only single-operation queries, where our approach supports multi-operation queries with competitive results, as shown in Section 4.2.

The structural algorithm in (Plebani and Pernici 2009) features a semantic extension as WSDL specifications often lack details on the real goal of the whole Web Service and their constituting operations as well. The UDDI Registry By Example (URBE), a Web Service retrieval algorithm for substitution purpose, is based on WSDL as the model to define the Web Service interfaces. To make substitution possible, a substitute Web Service has to expose an interface functionally equal or richer than that of the failed Web Service. The approach considers the relationships between the main elements composing a WSDL specification (portType, operation, message, and part). The similarity function is calculated upon operation name similarity and parameters data type similarity, and a maximization function. This function exploits linear programming (rather than the hungarian algorithm) to solve the assignment problem and hence obtain the maximum similarity between the aforementioned elements. The semantic-aware variant of the algorithm (URBE-S) leverages Semantic Annotated WSDLs (SAWSDL) since they are built upon WSDL (the document structure remains the same) and simply add some annotations which better describe operations and parameters. Then, ontology matching techniques are used to compare annotations against OWL descriptions, which results in a hybrid approach when combined with the analysis of data-types. Authors compared the URBE-S registry against Woogle and the Stroulia algorithm, showing that URBE-S outperformed other approaches considering different IR metrics such as Precision and Recall. Although the dataset and query set were smaller than ours, it was considerable (570 services and 27 queries). This is, to the best of our knowledge, the best performing approach for service retrieval based on similarity evaluation. The weak point of URBE-S is, as we mentioned earlier, that providers do not

annotate their services often in practice, since it is costly in terms of time and effort. Even when introducing SAWSDL annotations provides a more accurate description of the elements included in a WSDL description, this requires deep knowledge of the OWL specification, thus increasing the adoption barrier.

5.2.4 Non-functional aspects based service selection

From a non-functional point of view, developers may select their Web Services according to different non-functional aspects, as to what extent they can be trusted in social-based recommendation (Malik and Bouguettaya 2009; Cao et al. 2013; Al-Sharawneh et al. 2011), quality of experience of previous users (Lalanne et al. 2012), and traditional QoS parameters (Oskooei and Daud 2014; Al-Masri and Mahmoud 2007; Lacheheub and Maamri 2016).

The work in (Lalanne et al. 2012) presents an approach to perform Web Service selection using Quality of Experience (QoE). QoE can be defined as the overall acceptability of an application or service, as perceived subjectively by the end-user. By establishing a correlation between the user-perceived quality and traditional QoS parameters such as execution time, availability and accuracy, value-added service providers may be able to provide services which better satisfy the user-expectation of quality. This approach does not consider any functional (structural nor semantic) aspect, but only QoE aspects of the services in their evaluation. We believe that it is possible to complement our hybrid approach with a QoE-based approach for service selection to assess both functional and non-functional aspects.

The work in (Oskooei and Daud 2014) proposes a QoS-based model to provide a mechanism for Web Service selection. The model consists on an ad-hoc ontology with information about several QoS attributes, including maintainability, portability, efficiency, reliability and functionality. Such information is contained in an XML-based description linked to WSDL documents in a service repository. Although the approach seems promising, the work does not address two crucial and related aspects: the impact of manually annotating WSDL services, and the empirical validation about the performance of the approach. Manually annotating services to validate the model is mandatory, in order to generate a WSDL-data-set including the extra QoS information. As stated earlier, this task is costly in terms of time and effort as it may force developers to construct their own ad-hoc ontologies.

The work in (Al-Masri and Mahmoud 2007) proposes a quality-driven discovery of relevant Web Services using well-known QoS attributes such as response time, throughput, availability, accessibility, interoperability analysis and cost. Some of these attributes may be informed by the service provider itself (e.g., cost) while others are measured by a specialized module. Authors assume that, for a given query, there is a set of relevant Web Services that share the same functionality but with different QoS attributes. However, it is not always the case, thus it is necessary to assess functional aspects as well. Besides, our hybrid approach could be weighted with QoS metrics such as the one proposed in (Al-Masri and Mahmoud 2007).

Similarly, the work in (Jiang et al. 2015) combines traditional QoS attributes (specified by providers in published WSDL documents), and user experience derived from the actual usage of services in practice. To avoid users ranking services maliciously, a message board system captures and displays users opinions and suggestions. The main problems of this approach are: functional aspects are not assessed; providers have to define server-side QoS of their services and annotate WSDL documents accordingly, which is costly in terms of time and effort; finally, for the QoS derived from user experience, the feedback obtained from the “wisdom of the crowds” (likes, votes, messages etc.) may be sparse, subjective, and slow to accumulate (Pelleg et al. 2016). Apart from that, the messages may be ambiguous and not automatically analyzable to obtain a discrete valuation of the corresponding service.

Finally, the work in (Choi and Jeong 2014) combines traditional QoS attributes with user-defined weights to generate an overall quality measure that depends on both the individual QoS and the priorities (weights) given to them by each particular user. As determining the weights of each attribute is a multi-criteria decision mechanism, it can be a burden for the users. To reduce such burden, authors propose a quality evaluation system that applies a pairwise comparison matrix and uses the eigenvector of the matrix to determine the priority weight of each quality attribute. Thus, the process assists users when assigning weights to QoS attributes. When each Web Service is evaluated with the weight reflecting a user’s quality preference, the evaluation result for even the same Web Service can be changed according to the preference of the user. This approach is interesting and complete, providing simulation results that confirm their hypothesis. As disadvantages we can mention a) the approach does not consider any functional (structural nor semantic) aspect, only QoS attributes, and b) it assumes that several services are functionally equivalent. We believe that it is possible to combine the detailed analysis of functional aspects in our hybrid approach with the weighted QoS notion to assess both functional and non-functional aspects.

5.3 Conclusions of related work

From the related work, we can broadly classify current approaches for service selection based on functional aspects in three groups: purely structural, IR-based and semantic descriptions-based. Approaches based on non-functional

aspects such as social-based recommendation, quality of experience of previous users, and traditional QoS parameters were also discussed for completeness but are not strictly within the scope of this work for comparison. Compared to our work, as suggested earlier, they represent complementary (not opposite) approaches.

Purely structural approaches are the baseline, as their drawbacks (already discussed) were further addressed by IR-based and semantic descriptions-based approaches. IR-based approaches are mostly structural, but also allow to gather some semantic information from functional specification of services, being WSDL the most widespread standard supported. Additionally, they exploit the rich background inherited from the IR field. Semantic descriptions-based approaches rely on machine-interpretable descriptions by using ontologies to describe services such as OWL-S or WSDL-S. These two meta-models are the main efforts to support automatic service discovery and selection. Other approaches such as pure QoS-aware approaches (Al-Masri and Mahmoud 2007; Own and Yahyaoui 2015) are not focused in functional description matching and thus are out of the scope of this work.

The literature shows that both IR-based and Semantic descriptions-based approaches have different goals and weaknesses/ strengths. IR-based approaches depend on publishers to use self-explanatory and meaningful identifiers and comments, which is not always the case (Mateos et al. 2013). Semantic-based approaches require ontologies and semantically annotated services based on concepts from these ontologies. However, such semantic extensions are not widely adopted as standards yet (Sheng et al. 2014; Chen and Paik 2013; Garriga et al. 2015). Meanwhile, information in WSDL specifications might be rich enough to infer semantics of the specified services. In any case, the cost of formally specifying provided and requested services is inherently higher than that of building conventional, text-based WSDL specifications.

As shown along this section, semantic-based proposals rely on semantic descriptions of services that generally are not available, since publishers must put extra effort into describing services by means of semantic meta-data (Lanthaler and Gutl 2011). A true spread of semantic services could only start when the derived advantages become of clear interest for the industry. Also, the concealment of meaningful information in WSDL documents due to the proliferation of code-first Web Services (Mateos et al. 2015a) hinders service discovery registries from providing accurate results.

For these reasons, this paper proposes a hybrid service selection method, which assesses services mainly exploiting as much structural and semantic information as possible from functional WSDL specifications. Also, our approach is based upon a lightweight semantic support to analyze terms in identifiers through WordNet to lower the adoption barrier.

6 Conclusions

In this paper we presented a novel hybrid service selection approach. Its underlying Interface Compatibility analysis gathers as much information as possible from the most widespread and de-facto standard documents for Web Services: WSDL specifications. Particularly, the underlying algorithms assess names (from operations and parameters identifiers), return types, parameter types and exceptions. These algorithms rely on the lexical database WordNet (through its Java API JWI) and the Paranamer library for accessing parameter names from compiled units, allowing us to solve the shortcomings of the Java Reflection Mechanism. Upon this basis, the Adaptability Gap provides an observable value as a meaningful insight for the adaptation effort for candidate services.

Also, we performed comparative experiments featuring structural, semantic and hybrid service selection methods. Through a set of experiments performed over the EasySOC service discovery registry (Crasso et al. 2011a), we compared three different service selection approaches. These approaches assess semantic and/or structural aspects of candidate Web Services to evaluate and rank candidate Web Services for its likely integration into a client application. The experiments have shown that executing any service selection method over previously discovered services improves the visibility of suitable candidate services w.r.t. original discovery results (from the EasySOC registry) and the Stroulia algorithm. Such improvement is expressed in terms of retrievability through Recall and NDCG metrics. For this experiment, the hybrid service selection method outperformed the other approaches. It is important to notice that results can depend on the data-set and queries used, and cannot be merely generalized to other experimental configurations. However, considering that the selection of candidate services is performed after a discovery process, increasing the visibility of most suitable candidates in a list of previously discovered services may ease the development of client applications. It should be highlighted that lightweight semantic-based methods depend on lexical dictionaries (e.g., WordNet), which are not always reliable. Without an “use context”, the semantic relationships supported by WordNet are way too general and the similarity values cannot be trusted as-is. When addressing a critical domain such as healthcare or financial, a domain-specific and well-founded ontology will be more reliable, regardless of their low availability. In general, the lack of relevant ontologies for the majority of domains is still a problem. Even so, the ontologies (when available) sometimes

are not comprehensive enough to express all the relevant concepts in a domain (Bouchiha et al. 2012; Guizzardi 2005; Lanthaler and Gutl 2011; Crasso et al. 2011b), hindering their applicability in practice.

Experimental results show that structural-aware service selection methods are better in terms of retrievability of suitable candidate services, and also suggest a likely reduction in the subsequent adaptation and integration effort for the candidate services retrieved in the topmost positions. This is due to the inherent importance of structural (e.g., data-types) and semantic aspects (e.g., parameter names) when assessing adaptability. Also, hybrid selection method outperformed other approaches, ensuring that the services retrieved in the first positions are the easiest to adapt and integrate into a client application.

Our current work to improve service selection is twofold. On one side, we are fine tuning the hybrid service selection method. This involves the recursive application of the semantic assessment to nested complex types and their corresponding fields, the usage of different lexical relations supported by the WordNet underlying hierarchy – e.g., hyponyms of hyperonyms or *siblings*, and the replacement (or combination) of such ontology with another one, such as Google n-gram (Michel et al. 2011) or DISCO (De Renzis et al. 2014; Kolb 2009). Particularly, Google n-gram is composed of terms extracted from millions of books from different domains.

In addition, we are currently working in a complementary step for service selection: a *Behavioral Compatibility* analysis that complements the Interface Compatibility analysis addressed in this work by achieving a protocol-level assessment. This analysis is based on assessing the operational behavior execution of services. This is achieved by applying a testing framework, in which a compliance test suite (TS) is generated, based on the required functionality. Preliminary results on a prototype for test generation and reduction are encouraging (Anabalon et al. 2015; Garriga et al. 2011).

Another concern is the composition of candidate services to fulfill functionality, which is particularly useful when a single candidate service cannot provide the whole required functionality. Service composition encompasses roles and functionality to aggregate multiple services into a single composite service, which can be even used as an atomic service in further service compositions. We are expanding the current procedures and models mainly based on business process descriptions and service orchestration to address trustful service composition, building interface or protocol adapters, and pragmatically attending the required testing task that inevitably follows any integration process (Peltz 2003; Weerawarana et al. 2005). Moreover, RESTful services (Fielding 2000) have shown their potential to compose reliable and visible Web-scale applications (Garriga et al. 2016) based on the so-called mashups (Benslimane et al. 2008). However, RESTful services and mashups still suffer from shortcomings on semantically describing, discovering and composing services as well as the absence of a holistic framework covering the entire service life-cycle (Garriga et al. 2016). In this context, we are adapting our approach to support retrieval and selection of services based upon RESTful descriptions.

Acknowledgments This work is supported by grants ANPCyT PICT 2012-0045 and UNCo–Reuse (04-F001). We would like to thank to the anonymous reviewers that contributed to improve the quality of this work with their helpful comments. Also, to Ph.D. Giuseppe Pirró who gently provided the articles and the JWSL library that depict and implement the semantic similarity metric, respectively. Finally, we thank to student Marcos Trotti for the implementation of the query mutator.

References

Agichtein, E., Brill, E., Dumais, S., & Ragno, R. (2006). Learning user interaction models for predicting web search result preferences. In *29th Annual ACM SIGIR International Conference on Research and Development in Information Retrieval* (pp. 3–10): ACM Press.

Al-Masri, E., & Mahmoud, Q.H. (2007). Qos-based discovery and ranking of web services. In *16th International Conference on Computer Communications and Networks, ICCCN'07* (pp. 529–534): IEEE Computer Society Press.

Al-Sharawneh, J., Williams, M., Wang, X., & Goldbaum, D. (2011). Mitigating risk in web-based social network service selection: follow the leader. In *International Conference on Internet and Web Applications and Services. The International Academy*: Research and Industry Association (IARIA).

Anabalon, D., Garriga, M., Flores, A., Cechich, A., & Zunino, A. (2015). Test reduction for easing web service integration. In *Proceedings of the Argentinean Symposium on Software Engineering* (pp. 115–129): SADIO.

Benslimane, D., Dudstar, S., & Sheth, A. (2008). Service mashups: The new generation of web applications. *IEEE Internet Computing*, *12*(5), 13–15.

Bouchiha, D., Malki, M., Alghamdi, A., & Alnafjan, K. (2012). Semantic web service engineering: Annotation based approach. *Computing and Informatics*, *31*(6), 1575–1595.

Butler, S., Wermelinger, M., Yu, Y., & Sharp, H. (2011). Mining java class naming conventions. In *IEEE International Conference on Software Maintenance (ICSM)* (pp. 93–102): IEEE.

Cabral, L., Domingue, J., Motta, E., Payne, T., & Hakimpour, F. (2004). Approaches to semantic web services: an overview and comparisons, In Bussler, C., Davies, J., Fensel, D., Fensel, D., & Studer, R. (Eds.) *The Semantic Web: Research and Applications, volume 3053 of Lecture Notes in Computer Science* (pp. 225–239). Berlin: Springer.

Cao, B., Liu, J., Tang, M., Zheng, Z., & Wang, G. (2013). Mashup service recommendation based on user interest and social network. In *IEEE 20th International Conference on Web Services (ICWS)* (pp. 99–106): IEEE.

Chen, W., & Paik, I. (2013). Improving efficiency of service discovery using linked data-based service publication. *Information Systems Frontiers*, *15*(4), 613–625.

Choi, C.R., & Jeong, H.Y. (2014). A broker-based quality evaluation system for service selection according to the qos preferences of users. *Information Sciences*, *277*, 553–566.

Cong, Z., Fernandez, A., Billhardt, H., & Lujak, M. (2015). Service discovery acceleration with hierarchical clustering. *Information Systems Frontiers*, *17*(4), 799–808.

Crasso, M., Mateos, C., Zunino, A., & Campo, M. (2014). EasySOC: Making web service outsourcing easier. *Information Sciences*, *259*, 452–473.

Crasso, M., Zunino, A., & Campo, M. (2008). Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, *71*(2), 144–164.

Crasso, M., Zunino, A., & Campo, M. (2011a). Combining query-by-example and query expansion for simplifying web service discovery. *Information Systems Frontiers*, *13*(3), 407–428.

Crasso, M., Zunino, A., & Campo, M. (2011b). A survey of approaches to web service discovery in service-oriented architectures. *Journal of Database Management (JDM)*, *22*(1), 102–132.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, *6*(2), 86–93.

De Renzis, A., Garriga, M., Flores, A., Zunino, A., & Cechich, A. (2014). Semantic-structural assessment scheme for integrability in service-oriented applications. In *Latin-american Symposium of Enterprise Computing, held during CLEI'2014*.

Delamaro, M., Maidonado, J., & Mathur, A. (2001). Interface mutation: an approach for integration testing. *IEEE Transactions on Software Engineering*, *27*(3), 228–247.

Dong, X., Halevy, A., Madhavan, J., Nemes, E., & Zhang, J. (2004). Similarity search for web services. In *Proceedings of the International Conference on Very Large Data Bases VLDB* (pp. 372–383): VLDB Endowment.

Elish, M.O., & Offutt, J. (2002). The adherence of open source java programmers to standard coding practices. In *6th International Conference on Software Engineering and Applications IASTED* (pp. 374.200–374.207).

Erl, T., Kamarkar, A., Walmsley, P., Haas, H., Yalcinalp, U., Liu, C., Orchard, D., Tost, A., & Pasley, J. (2008). *Web service contract design & versioning for SOA*, 1st edn Vol. 1: Prentice Hall.

Falleri, J.-R., Huchard, M., Lafourcade, M., Nebut, C., Prince, V., & Dao, M. (2010). Automatic extraction of a wordnet-like identifier network from software. In *International Conference on Program Comprehension (ICPC)* (pp. 4–13): IEEE.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. USA: PhD thesis, University of California.

Flores, A., Cechich, A., Zunino, A., & Polo, M. (2010). Testing-based selection method for integrability on service-oriented applications. In *5th International Conference on Software Engineering Advances ICSEA'10* (pp. 373–379). Nice: IEEE Computer Society Press.

Flores, A., & Polo, M. (2012). Testing-based process for component substitutability. *Software Testing, Verification and Reliability*, *22*(8), 529–561.

Fowler, M. (1999). Refactoring: improving the design of existing code. *Pearson Education India*.

Fragoso, O., Santaolaya, R., & Solis, I. (2006). Using case-based reasoning for improving precision and recall in web services selection. *International Journal of Web and Grid Services*, *2*(3), 306–330.

Funika, W., Godowski, P., Pęgiel, P., & Król, D. (2012). Semantic-oriented performance monitoring of distributed applications. *Computing and Informatics*, *31*(2), 427–446.

Garriga, M., Flores, A., Cechich, A., & Zunino, A. (2011). Testing-based process for service-oriented applications. In *30th IEEE International Conference of the Chilean Computer Science Society (SCCC)* (pp. 64–73).

Garriga, M., Flores, A., Cechich, A., & Zunino, A. (2015). Web services composition mechanisms: a review. *IETE Technical Review*, *32*(5), 376–383.

Garriga, M., Flores, A., Mateos, C., Zunino, A., & Cechich, A. (2013). Service selection based on a practical interface assessment scheme. *International Journal of Web and Grid Services*, *9*(4), 369–393.

Garriga, M., Mateos, C., Flores, A., Cechich, A., & Zunino, A. (2016). {RESTFul} service composition at a glance: a survey. *Journal of Network and Computer Applications*, *60*, 32–53.

George, B., Fleurquin, R., & Sadou, S. (2008). A component selection framework for COTS libraries. *Lecture Notes In Computer Sciences*, *5282*, 286–301.

Guizzardi, G. (2005). *Ontological foundations for structural conceptual models*, 1st Ed. The Netherlands: Universal Press. ISBN 90-75176-81-3.

Gupta, S., Malik, S., Pollock, L., & Vijay-Shanker, K. (2013). Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In *International Conference on Program Comprehension (ICPC)* (pp. 3–12): IEEE.

Heß, A., Johnston, E., & Kushmerick, N. (2004). Assam: A tool for semi-automatically annotating semantic web services. In *The Semantic Web–ISWC 2004* (pp. 320–334): Springer.

Hummel, O., Janjic, W., & Atkinson, C. (2008). Code conjurer: Pulling reusable software out of thin air. *IEEE Software*, 45–52.

Jiang, L., Liu, T., & Liu, D. (2015). Objective and subjective qos factors supported web service search method based on extended wsdl. In *International Conference on Geoinformatics* (pp. 1–4): IEEE.

Kessel, M., & Atkinson, C. (2016). Ranking software components for reuse based on non-functional properties. *Information Systems Frontiers*, *18*(5), 825–853.

Kil, H., Oh, S.-C., & Lee, D. (2006). On the topological landscape of web services matchmaking. *VLDB Workshop on Semantic Matchmaking and Resource Retrieval (CEUR)*, *178*, 19–34.

Kirk, R.E. (1982). *Experimental design*: Wiley Online Library.

Kokash, N. (2006). A comparison of web service interface similarity measures. Amsterdam: IOS Press.

Kolb, P. (2009). Experiments on the difference between semantic similarity and relatedness. In *Proceedings of the 17th Nordic Conference on Computational Linguistics - NODALIDA'09*.

Kuhn, H.W. (1955). The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, *2*, 83–97.

Kung-Kiu, L., & Zheng, W. (2007). Software component models. *IEEE Transactions on Software Engineering*, *10*, 709–724.

Lacheheub, M.N., & Maamri, R. (2016). Towards a construction of an intelligent business process based on cloud services and driven by degree of similarity and qos. *Information Systems Frontiers*, *18*(6), 1085–1102.

Lalanne, F., Cavalli, A., & Maag, S. (2012). Quality of experience as a selection criterion for web services (pp. 519–526): IEEE.

Lanthaler, M., & Gutl, C. (2011). A semantic description language for RESTful data services to combat semaphobia. In *Proceedings of the 5th International Conference on Digital Ecosystems and Technologies Conference (DEST)* (pp. 47–53): IEEE.

Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J.A., & Sheth, A. (2006). Designing semantic web processes: The wsdl-s approach. In *Semantic Web Services, Processes and Applications* (pp. 161–193): Springer.

Malik, Z., & Bouguettaya, A. (2009). Rateweb: Reputation assessment for trust establishment among web services. *The VLDB Journal – The International Journal on Very Large Data Bases*, *18*(4), 885–911.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuiness, D., Sirin, E., & Srinivasan, N. (2007).

Bringing semantics to web services with owl-s. *World Wide Web*, *10*, 243–277.
Mateos, C., Crasso, M., Zunino, A., & Coscia, J. (2013). Revising wsdl documents: Why and how, part 2. *IEEE Internet Computing*, *17*(5), 46–53.
Mateos, C., Crasso, M., Zunino, A., & Ordiales, J.L. (2011). Detecting WSDL bad practices in code–first web services. *International Journal of Web and Grid Services*, *7*(4), 357–387.
Mateos, C., Crasso, M., Zunino, A., & Ordiales Coscia, J.L. (2015a). A stitch in time saves nine: Early improving code-first web services discoverability. *International Journal of Cooperative Information Systems*, *24*(02).
Mateos, C., Rodriguez, J.M., & Zunino, A. (2015b). A tool to improve code-first web services discoverability through text mining techniques. *Software: Practice and Experience*, *45*(7), 925–948.
McCool, R. (2005). Rethinking the semantic web. *IEEE Internet Computing*, *9*(6), 86–87.
Michel, J., Shen, Y., Aiden, A., Veres, A., Gray, M., Pickett, J., Hoiberg, D., Clancy, D., Norvig, P., & Orwant, J. (2011). Quantitative analysis of culture using millions of digitized books. *Science*, *331*(6014), 176–182.
Miller, G., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Introduction to wordnet: an on-line lexical database. *International Journal of Lexicography*, *3*(4), 235–244.
Noy, N.F. (2004). Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, *33*(4), 65–70.
Orne, M.T. (2009). *Artifacts in Behavioral Research, chapter Demand characteristics and the concept of quasi-controls*: Oxford University Press.
Oskooei, M.A., & Daud, S.M. (2014). Quality of service (qos) model for web service selection. In *International Conference on Computer, Communications, and Control Technology (I4CT)* (pp. 266–270): IEEE.
Own, H., & Yahyaoui, H. (2015). Rough set based classification of real world web services. *Information Systems Frontiers*, *17*(6), 1301–1311.
Papazoglou, M., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *IEEE Computer*, *40*(11), 38–45.
Pasley, J. (2006). Avoid XML schema wildcards for web service interfaces. *IEEE Internet Computing*, *10*(3), 72–79.
Pelleg, D., Rokhlenko, O., Szpektor, I., Agichtein, E., & Guy, I. (2016). When the crowd is not enough: Improving user experience with social media through automatic quality analysis. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, CSCW '16* (pp. 1080–1090). New York: ACM.
Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, *36*(10), 46–52.
Pirró, G. (2009). A semantic similarity metric combining features and intrinsic information content. *Data & Knowledge Engineering*, *68*(11), 1289–1308.
Plebani, P., & Pernici, B. (2009). Urbe: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, *21*(11), 1629–1642.
Rajaraman, A., & Ullman, J. (2011). *Mining of massive datasets, chapter data mining*: Cambridge University Press.
Rambold, M., Kasinger, H., Lautenbacher, F., & Bauer, B. (2009). Towards autonomic service discovery a survey and comparison. In *IEEE International Conference on Services Computing, SCC* (pp. 192–201): IEEE.
Rodriguez, J.M., Crasso, M., Mateos, C., & Zunino, A. (2013). Best practices for describing, consuming, and discovering web services: A comprehensive toolset. *Software: Practice and Experience*, *43*(6), 613–639.
Rodriguez, J.M., Crasso, M., Zunino, A., & Campo, M. (2010). Improving web service descriptions for effective service discovery. *Science of Computer Programming*, *75*(11), 1001–1021.
Roman, D., Keller, U., Lausen, H., De Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., & Fensel, D. (2005). Web service modeling ontology. *Applied ontology*, *1*(1), 77–106.
Salton, G., Wong, A., & Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613–620.
Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, *280*, 218–238.
Shvaiko, P., & Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, *25*(1), 158–176.
Sivashanmugam, K., Verma, K., Sheth, A.P., & Miller, J.A. (2003). *Adding semantics to web services standards*.
Stroulia, E., & Wang, Y. (2005). Structural and semantic matching for assessing web-services similarity. *International Journal of Cooperative Information Systems*, *14*, 407–437.
Tibermacine, O., Tibermacine, C., & Cherif, F. (2013). Wssim: A tool for the measurement of web service interface similarity. In *Proceedings of the french-speaking Conference on Software Architectures (CAL'13)*. Toulouse.
Tversky, A. (1977). Features of similarity. *Psychological review*, *84*(4), 327.
Weerawarana, S., Curbera, F., Leymann, F., Storey, T., & Ferguson, D. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-Reliable Messaging, and More*: Prentice Hall PTR.
Willett, P. (2006). The porter stemming algorithm: then and now. *Program: electronic library and information systems*, *40*(3), 219–223.
Wu, J., & Wu, Z. (2005). Similarity-based web service matchmaking. In *IEEE International Conference on Services Computing*, (Vol. 1 pp. 287–294): IEEE Computer Society Press.
Zimmerman, O., Tomlinson, M., & Peuser, S. (2005). *Perspectives on web services – applying SOAP, WSDL and UDDI to real-world projects*: Springer.

Martin Garriga received his PhD. degree in Computer Science in 2016 at Faculty of Exact Sciences, UNICEN (Tandil, Argentina). He is a posdoctoral fellow at Politecnico de Milano (Italy) since 2016, and Lecturer Assisstant at Informatics Faculty, UNComa (Neuquén, Argentina) since 2011. His research interests are Service-oriented Architectures, Web Service selection and composition, RESTful Services and Microservices architectures.

Alan De Renzis received his BSc. degree in Systems Engineering in 2013 at the Faculty of Informatics, UNComa. Since then he is a PhD. candidate at Faculty of Exact Sciences, UNICEN. His research interests are Service-oriented Architectures, Web Service discovery and selection and Service metamodels.

Ignacio Lizarralde received his BSc. degree in Systems Engineering in 2013 at the Faculty of Exact Sciences, UNICEN. Since then he is a PhD. candidate at the same institution. His research interests are energy-consumption of mobile devices, mobile and web services, hybrid grids and cloud computing parallelisation.

Andrés Flores received his Ph.D. degree in Informatics from University of Castilla-La Mancha, Spain in 2009. He is Adjunct Professsor at Informatics Faculty, UNComa since 2010, and Researcher at the argentinean National Scientific and Technical Research Council (CONICET) since 2012. His research interests are Software Engineering, Service-oriented Computing, Componentbased Systems, Software Testing.

Cristian Mateos received a PhD. Degree in Computer Science in 2008 at the Faculty of Exact Sciences, UNICEN. He is full time Professor at the Faculty of Exact Sciences, UNICEN University since March 2016, and Researcher at CONICET since 2009. His research interests are parallel and distributed programming, particularly gridifying and cloudifying applications, (mobile) Grid middlewares and platforms, Service-Oriented Computing and Web Services.

Alejandra Cechich received her Ph.D. degree in Informatics from University of Castilla-La Mancha, Spain (2005). She is Full time Professor at Informatics Faculty, UNComa since 1996. Her research interests are Software Engineering, Software Reuse, Software Quality and Architectures.

Alejandro Zunino received his Ph.D. degree in Computer Science at the Faculty of Exact Sciences, UNICEN university, in 2003. He is Full time Professor at Faculty of Exact Sciences, UNICEN since 2006, and Researcher at CONICET since 2005. His research interests are Software Engineering, Mobile Computing, Service-oriented Architectures, and Grid and Cloud Computing.