Improving Web service discovery by a functional and structural approach

Rodrigo Amorim*, Daniela Barreiro Claro*, Denivaldo Lopes†, Patrick Albers‡ and Aline Andrade*
*FORMAS Research Group - Distributed Systems Laboratory (LASID)
Federal University of Bahia (UFBA) - Av. Adhemar de Barros, Salvador/Bahia - Brazil
Email: brzapa@gmail.com, dclaro@ufba.br, aline@ufba.br
†Software Enginnering and Network Laboratory (LESERC)
Federal University of Maranhao, Bacanga Campus/CCET, Sao Luis/Maranhao - Brazil Email: dlopes@dee.ufma.br
‡ESEO - Ecole Superieure d'Electronique de l'Ouest
4 rue M. de la Boulaye, Angers - France Email: patrick.albers@eseo.fr

***Abstract*—Service oriented Architecture (SOA) has been widely used in service computing applications and this fact has been encouraged the publication over the Web in a Web Service format. Whereas the number of Web services published on the Web is growing up, discovery techniques must be improved so as to retrieve more desirable services. Nowadays, the most commonly used technique is semantic filters based on ontological concepts. However, such mechanisms can leave out some important Web services of the matching process, because of their structural relationship not mentioned in an ontology. In order to overcome such problems, some authors have proposed a hybrid approach to combine traditional syntactic and semantic approaches. These proposals remain restricted especially with respect to complexity, precision and time of execution, thus making such solutions in most of cases unfeasible. In this paper, we combine semantic filters based on functional properties with a structural approach, analyzing each neighbor relationship in an ontology. The results showed a considerable improvement in terms of performance and a complexity reduction with respect to other existing techniques. Furthermore, we implement a tool called OWLS-S Discovery in order to simplify the use of our approach by developers.**

I. Introduction

A variety of new approaches and applications such as Cloud Computing and Autonomic Computing use Service Oriented Architecture (SOA) implemented as Web services. Those Web services facilitate the information exchange between applications due to the interoperability of standards such as XML, SOAP and WSDL. However, the discovery of Web services is usually difficult and time consuming.

The discovery process used to find a desirable Web service usually requires some human intervention due to differences between service interfaces and request interfaces. In this scenario, the use of semantic technologies in Web services facilitates the process of automatic discovery. Various languages have been used to allow an unambiguous and computationally interpretable description of semantic Web services, as OWL-S[1], WSMO[2] and SAWSDL[3]. Different semantic mechanisms have been incorporated into these languages to automate such discovery. In particular we can mention OWLS-UDDI matchmaker [4], OWLS-MX [5], WSMO-MX [6] and TUB OWL-S matcher[7]. However these approaches perform a purely semantic-based discovery, except for the OWLS-MX and WSMO-MX which, instead of only exploiting a discovery of semantic-based concepts, perform a syntactic textual analysis of the concepts used in the discovery process.

Our approach presents an algorithm for discovering semantic Web services based on functional semantic and structural analysis in an OWL ontology. This algorithm is composed of two steps, the first one called the functional semantic step and the second called the descriptive semantic step. The functional semantic step was developed following the functional concepts presented in [8]. The second step keeps track of the concepts presented in our previous work [9], which presents a structural analysis using ontology and a dictionary of synonyms for the classification of concepts. Although the proposed algorithm uses OWL-S as a way to specify the services interfaces unambiguously, the algorithm is not tied to it. Experiments were performed in order to validate the efficiency of the presented algorithm. A tool, *OWL-S Discovery*, was developed to disseminate the use of this algorithm for semantic discovery of Web services. We can summarize the main contributions of this paper:

- to present an hybrid approach of our semantic based algorithm for discovering Web Service.
- to present our algorithm that increases both accuracy and performance.
- to validate the efficiency of our proposed algorithm.
- to propose a tool to disseminate this algorithm.

The article is structured as follows: Section 2 presents related work. Section 3 the proposed algorithm is presented. Section 4 discusses the experiments and analyses our results. Section 5 presents our *OWL-S Discovery* tool. Section 6 concludes and gives some research directions.

II. Related Work

Several researches have been carried out to discover semantically similar Web services and OWL-S/UDDI Matchmaker[4], OWLS-MX[5] and SAMT4MDE[9] approaches are the closest to our proposal.

Work in [4] presents the tool called OWL-S/UDDI Matchmaker. In its implementation it takes into account semantic information that is extracted from OWL-S. Such an approach

only considers input and output parameters of a Web service. The degree of similarity between input and output is tied to the relationship between these parameters and the concepts within a domain ontology. The categorization of such concepts follows a similar model as proposed in [8]. Among the existing tools, Matchmaker OWL-S/UDDI provides the best performance in particular for the discovery time; however, our approach achieved better accurate results, it means more desirable web services are retrieved.

The work in [5] presents a tool called OWLS-MX, which uses a hybrid technique for searching semantic Web services. It uses a reasoner within OWL-DL and a technique for identifying syntactic similarity of concepts. Ontology-based reasoning is similar to the approach used within OWL-S/UDDI Matchmaker, where it only deals with input and output parameters. Their syntactic approach is based on text metrics of similarity from the information retrieval domain. OWLS-MX developers implemented four variations of their algorithm, each one using a different way to calculate syntactic similarity. A comparative study with various techniques of syntactic similarity was done in [10]. Better performing algorithms were used within OWLS-MX by developers. Even though their syntactic similarity presents better results in some experiments in comparison with semantic filters, our approach ensures better accuracy and performance, returning a large set of potential results. The main drawback of their proposal is that a lot of suffix or prefix exists that has no semantic meaning nor relation to retrieve correlate services. Thus, it can lead to a set of false positives (i.e. services that were not supposed to be retrieved).

SAMT4MDE[9] was our first challenge to deal with discovery and correlation issues. SAMT4MDE was based on our previous work [11], [12] which performs an analysis of concepts between metamodels using a dictionary of synonyms. We improved such approach so as to consider that these concepts are as similar as its structural neighbors. Those ideas were incorporated into this work to improve our semantic-based algorithm within structural analysis. However, the complexity of SAMT4MDE is its major drawback. Firstly we changed all metamodels into domain ontology and aggregated some structures which allow one to obtain better results in terms of accuracy and performance.

III. Matching Algorithm

Most of matching algorithms like OWL-S UDDI Matchmaker[4] and OWLSM[7] provide a large number of false positives, i.e. Web Service and request supposed to be similar, but actually they are not. In our approach, false positives are part of the metrics used to determine the efficiency of an algorithm. A large number of false positives can induce a false similarity between concepts thus damaging their accuracy. The use of semantic-based approaches specifically the profile ontology can enable an unambiguous matching and thus minimizing the number of false positives.

The major problem of discovery is to assume that requests and Web service parameters are always identical. This rigid solution ignores many Web services which can partially match a user request. Our algorithm can cope with such problem because it deals with semantic-based parameters within domain ontology. All input and output parameters are mapped to concepts in the domain ontology. In this work, we are only considering simple data types. Thus, we can reduce the number of false positives and consequently we can gain more precise results. Our algorithm is composed of two steps which are presented in the following subsections.

A. Functional Semantic Algorithm

The main idea of this approach is to match all input and output parameters from the service to all input and output parameters of the request respectively. Those input and output parameters are mapped to concepts in the domain ontology. The degree of similarity between such parameters is obtained based on their relationship and inference inside this OWL ontology. Considering this scenario we can observe a superclass *Means of Transportation* which has two subclasses called *Automobile* and *Aircraft*. *Automobile* has as subclass *Taxi* and *Bus*, while *Aircraft* has as subclasses *Air Taxi* and *Airliner Jet* (see Figure 1).

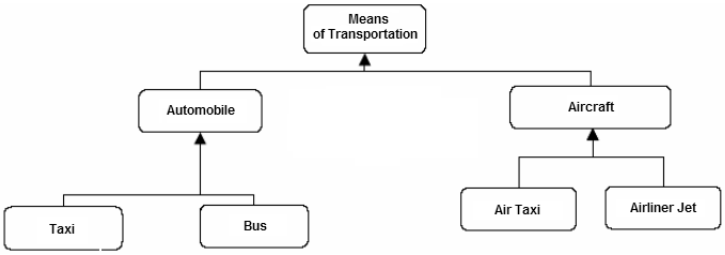

Fig. 1. Hierarchical concept of a transport domain.

In [8], four levels of similarity between parameters are defined.

Exact: Two concepts are classified as exact only if such concepts are exactly identical in the ontology or they have a direct relationship of class and subclass in this ontology. For instance, we have a service with input parameter: *Means of Transportation*; thus the exact degree of match might have as request input parameter: *Means of Transportation* or a subclass of it, i.e. *Automobile* (see Figure 1).

Plug-in: Two concepts are defined as plug-in only if the request input parameter is superclass of the service input parameter, except the direct relationship that is already treated by exact degree of match. For instance, we have a service with input parameter: *Taxi*; thus the plug-in degree of match might have as request input parameter: *Automobile* (see Figure 1).

Subsumes: Two concepts are classified as a subsume degree of match only if the request input parameter is subclass of the service input parameter. For instance,

we have a service with input parameter: *Automobile*; thus the subsume degree of match might have as request input parameter: *Taxi* (see Figure 1).

Fail: This degree of similarity expresses the lack of semantic-based relationship between two concepts.

A complement of the above degree of match is proposed by [13] as a new semantic filter in order to obtain better results. This filter, called a sibling, analyzes two concepts within an ontology which have the same direct parent. In other words, this new filter examines the sibling within an ontology. Thus, we can summarize as follows in descending order the degree of match:

$$\text{Exact} > \text{Plug-in} > \text{Subsumes} > \text{Sibling} > \text{Fail}$$

However, the only use of a semantic-based approach can lack some similarity concepts that cannot be reached by those filters. Thus, an OWL ontology structural analysis can retrieve some similar concepts not yet caught by those semantic-based filters. Our description semantic-based algorithm is a structural approach to analyze a concept inside the domain ontology.

B. Description semantic-based algorithm

The second step of our proposed algorithm was adapted from our previous work [11], [12]. Such approach uses an operator $Match(C_a, C_b) = r, 0 \leq r \leq 1$, which takes two concepts as input (service input and request input) and returns the relationship between them. The similarity analysis of these concepts is used through a dictionary that classifies the concepts. The degree of similarity is given as follows:

- value $1.0(one)$ if concepts are identical, i.e. request input is *Automobile*; service input is also *Automobile*.
- value $0.5(half)$ if concepts are synonymous, i.e. request input is *Automobile*; service input is *Vehicle*.
- value $0.0(zero)$ if concepts are neither synonymous nor identical.

Similarity between two concepts is calculated in the following way: two concepts (c_1 and c_2) are similar as they are structural neighbors (parents, children or siblings). The function $similarity(c_1, c_2)$ is a weighted sum between structural similarity and basic similarity as follows:

$similarity(c_1, c_2) = simBasic(c_1, c_2) * p_1 + simStruct(c_1, c_2) * p_2$; where $0 \leq p_1 < 1, 0 \leq p_2 < 1$ *e* $p_1 + p_2 = 1$.

Weights are p_1 and p_2 which together should be equal to 1. The return value of the function $similarity(c_1, c_2)$ is compared with a minimum threshold set by the user's application. If the result of the function $similarity(c_1, c_2)$ is less than this threshold, two concepts are classified as no-matching, while if the result is greater than the threshold value, those two concepts are related and their similarity is the return value of $similarity(c_1, c_2)$.

The threshold value is an important decision point. If it is very low, between $[0.0, 0.5]$, many concepts can be wrongly matched i.e. false positives can be returned by function $similarity(c_1, c_2)$. On the other hand, if this value is high, between $[0.7, 1.0]$, many concepts which might be matched are not caught, i.e. many false negatives can be discarded. Ideally, studies should be carried on to determine the best value of the threshold. In our approach we did not adjust neither threshold nor weight values; both are empirical values given by the operator user.

The operator returned from the structural and basic function generates a quadruple which combines ancestors, descendants, siblings and leaf nodes. This quadruple maps the whole set of concepts which are going to be analyzed within an ontology (see Figure 2).

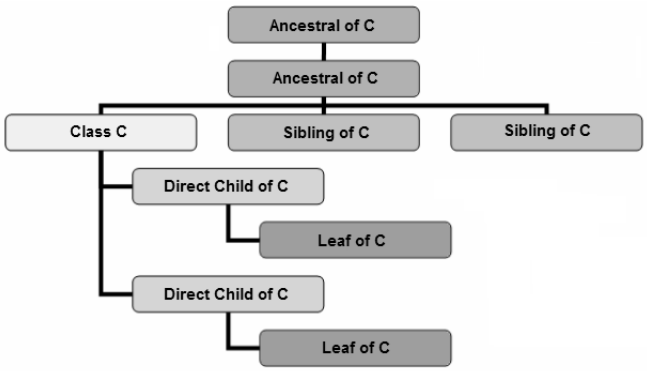

Fig. 2. Structural neighbors of a concept within an ontology.

$SimStruct(c_1, c_2) = p_3 * SimAncestors(c_1, c_2) + p_4 * SimDescendent(c_1, c_2) + p_5 * SimSibling(c_1, c_2) + p_6 * SimLeaf(c_1, c_2)$

In order to clarify how structural neighbors work, let us suppose the calculation of the ancestors of c_1 and c_2. Firstly, all ancestors of both c_1 and c_2 are mapped to concepts inside an ontology. Then, the dictionary analysis to find synonyms is performed and a matrix with all the results is generated. The dimension of such matrix is the cardinality of $Ancestors(c_1)$ multiplied by the cardinality of $Ancestors(c_2)$.

$$m = \frac{\sum_{i=1}^{|Ancestors(c_1)|}(\sum_{j=1}^{|Ancestors(c_2)|} M[i][j])}{|Ancestors(c_1)| * |Ancestors(c_2)|} \quad (1)$$

$$sd = \frac{\sum_{i=1}^{|Ancestors(c_1)|}(\sum_{j=1}^{|Ancestors(c_2)|} M[i][j] - m)^2}{|Ancestors(c_1)| * |Ancestors(c_2)|} \quad (2)$$

$$vc = \frac{sd}{m} \quad (3)$$

Afterwards, the average of all concepts is calculated by equation (1) and their standard deviation(*sd*) is calculated by equation (2). This analysis is to verify if the variance coefficient (*vc*) is greater than the threshold value. Only the similar high values than the threshold are taken into account.

If the variance coefficient (*vc*) is lesser than the threshold value, a scan can be done in the matrix to remove all terms that magnify the value of the standard deviation, i.e. all the concepts below $m * (1 - threshold)$ are removed from the matrix.

Once the stop condition is met, the result of the average of the remaining concepts is returned as a result of the similarity between the ancestors of the given two concepts. Thus, our proposed descriptive semantic-based algorithm is made between direct descendants, siblings and leaves providing an outcome from the structural similarity.

The distribution of weighted values in each calculation must be uniform as the calculation of similarity is more flexible and comprehensive, considering all structural neighbors equally important [9]. Thus weights given to basic and structural similarity must also be preferably uniform.

C. Proposed Algorithm

Our proposed algorithm explores the maximum potential of each step. The functional semantic step presents computational simplicity in its implementation, despite the fact that it may contain some flaws that can compromise the return of useful services to users. The descriptive semantic step is more complex computationally and complete, but this step can retrieve services which were not retrieved by the functional step.

This paper suggests the combination of those two steps as follows: in order to simplify the calculation and to improve performance, discovery process starts by running the functional semantic algorithm. In the presence of a failure on matching concepts(i.e. fail degree of match), the descriptive semantic algorithm is triggered as a means to demonstrate the real lack of similarity in the correspondence.

IV. Analysis of proposed algorithm

Among various approaches and algorithms for the discovery of Web services, most use the semantic filters proposed by [8]. Despite the contributions of these semantic filters, they have some drawbacks that other techniques have already dealt with. Within the scenario depicted in Figure 1, we attempt to explain some drawbacks and present possible solutions dealt by our proposed algorithm. Behind such a hierarchical classification, there is the user's request (R) and two services (S) depicted in Figure 3.

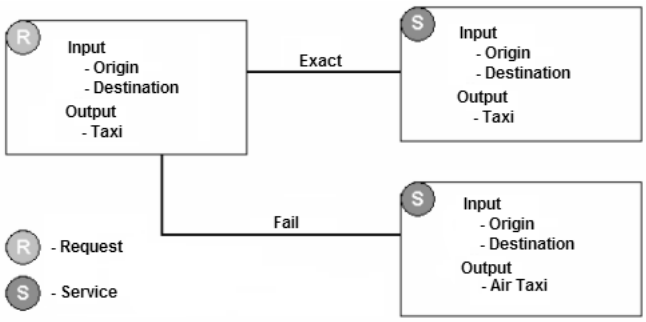

Fig. 3. Matching using semantic filters.

Figure 3 depicts the match between the request and the service taking only into account semantic filters. Within the first upper service, the matching between such service and the request corresponds as *Exact* because both interfaces (request and services) are identical. The second service matches as *Fail*, because there is no relationship between the class "Taxi" and "Air Taxi" in the hierarchy (Figure 1).

In face of such situations, the authors in [5] presented another way to match concepts between request and services. This approach, called syntactic analysis, performs a purely textual search on the words involved to verify that both terms are really disjointed. Within this new scenario, the service which failed previously using only semantic filters, now will not fail anymore. This is due to a syntactic analysis on the concepts "Taxi" and "Air Taxi" which have some string of characters in common and therefore are probably similar. Even so, the syntactic analysis is prone to errors, see the situation illustrated in Figure 4.

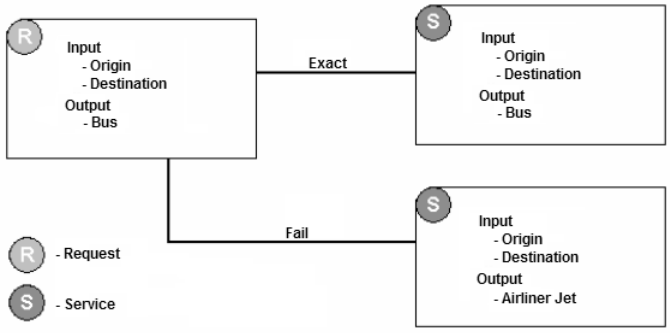

Fig. 4. Matching using semantic and syntactic filters.

A failure can occur by both the semantic filter and the syntactic correspondence (Figure 4). Following the ideas in [5] the syntactic correspondence fails because no textual correspondence can be found between the words "Bus" and "Airliner Jet". However, if the difference between travelling by bus or by airliner jet is very small (one should not take into account this small difference based on price nor time, but goal achieved), these algorithms are discarding a very useful service for a user. Thus, in such situations it is better to leave the decision as to the means of transportation for the user.

In such situations, our descriptive semantic algorithm has an important strength. With the use of a dictionary of synonyms and the analysis of structural neighbors of each concept, it is easy to establish the relationship between these concepts. Even if "Bus" and "Airliner Jet" were not described in the dictionary as synonyms, the analysis of structural neighbors of each concept will reveal that they are related in some way, as they have some structural neighbors in common. In these cases, even if the name of two concepts directly involved in the matching do not correspond, their structural neighbors can actually attest if such concepts are completely disjointed or not.

Unlike syntactic analysis, our descriptive semantic algorithm can be used in conjunction with our semantic filters.

However, our descriptive approach is only triggered on the failure (match fails) of our functional semantic step. This behavior can avoid the undiscriminatory classification of synonyms inside our dictionary, i.e. "bank of sand" and "bank as a financial institution" can be blindly considered as similar concepts.

V. Proof of Complexity

In this section, we present our algorithm including both functional and descriptive semantic steps. We also analyze the complexity of our algorithm and some advantages provided by our approach.

As previously presented in Section 4, semantic matching which uses semantic filters requires that all parameters of the request input match all input service parameters (the same for output parameters). In this case, for each input and output parameter of the request a match is performed through the filter with each input and output parameter of the service respectively (see Algorithm 1).

Algorithm 1 Functional Semantic step

```
1: List listInputReq;
2: List listInputServ;
3: for (int i=0; i ≤ listInputReq.size(); i++) do
4:    for (int j = 0; j ≤ listInputServ.size(); j++) do
5:       GrauSim result = analyzeInputBySemanticFilters(listInputReq(i), listInputServ(j));
6:    end for
7: end for
8: public GrauSim analyzeInputBySemanticFilters(argumentReq, argumentServ)
9: if childDirect(argumentoReq, argumentoServ) then
10:   return EXACT;
11: else if childIndirect(argumentReq, argumentServ) then
12:   return PLUG-IN;
13: else if childIndirect(argumentServ, argumentReq) then
14:   return SUBSUMES;
15: else if parentsInCommon(argumentReq, argumentServ) then
16:   return SIBLING;
17: else
18:   return FAIL;
19: end if
```

For the processing of each instruction, a cost $T_x \geq 0$ is associated, where x means the line of instruction. In Functional Semantic step (algorithm 1), for processing lines 1-2, a cost of T_0 is fixed because this processing refers to the reading of data from files that describe services involved in the computing (OWL-S files). Line 3 has processing that occurs $T_{03}(i+1)$ times, while line 4 has processing that repeats $T_{04}(j+1)$ times in execution time.

A necessary condition for invocation of such algorithm is that the total number of input parameters of a service should be less or equal than the number of input parameters of a request. It makes no sense to execute this algorithm so as to analyze a request which a Web service is unable to provide.

Thus, in the worst case, the quantity of request parameters must be the same as of a running service. With this assumption, i and j is incremented (lines 3 and 4) to the same value n:

$T_0 + T_{03}(n+1) * T_{04}(n+1)$

The execution of Line 5 carries a search in a domain ontology for two concepts in the interface, one is the request and the other is the Web service. This execution is performed in the same manner as in the descriptive semantic step, except by the classification: in one case it corresponds to semantic filters and in the other is carried by the dictionary of synonyms. A fixed value g is the cost of computing:

$T_0 + T_{03}(n+1) * T_{04}(n+1) * g$

$T_0 + T_{03} * T_{04} * g(n^2 + 2n + 1)$

Having K as constant and $K \geq \{T_0, T_{03}$ and $T_{04}\}$

$T_0 + T_{03} * T_{04} * g(n^2 + 2n + 1) \leq K + K^2 * gn^2 + 2K^2gn + K^2g$

$K + K^2 * gn^2 + 2K^2gn + K^2g \leq Kn^2 + K^2 * gn^2 + 2K^2gn^2 + K^2gn^2$

$K + K^2 * gn^2 + 2K^2gn + K^2g \leq Kn^2(1 + 4Kg)$

The term of the highest degree of the polynomial describing this algorithm is Kn^2. Therefore, as stated by [14], regardless the complexity of g and added the computation of the execution in the order of $O(n^2)$, it is possible to conclude that, in the worst case, this algorithm (Algorithm 1) is $O(n^2)$. The same is carried on for outputs. In this way, if two blocks of code are sequentially executed and each one has a complexity of $O(n^2)$, thus all algorithms have the same complexity $O(n^2)$.

As stated previously, the descriptive semantic step performs an analysis on all parameters (request - service and input - output). However, the process is quite different because it also examines structural neighbors in the domain ontology that contains the concepts of examined interfaces (see Algorithm 2).

Algorithm 2 Descriptive Semantic step

```
1: List listInputReq;
2: List listInputServ;
3: for (int i=0; i ≤ listInputReq.size(); i++) do
4:    for (int j = 0; j ≤ listInputServ.size(); j++) do
5:       DegreeSim result1 = analyzeAncestors(listInputReq(i), listInputServ(j));
6:       DegreeSim result2 = analyzeLeaf(listInputReq(i), listInputServ(j));
7:       DegreeSim result3 = analyzeDirectChild( listInputReq(i), listInputServ(j));
8:       DegreeSim result4 = analyzeSibling(listInputReq(i), listInputServ(j));
9:    end for
10: end for
```

The methods *analyzeAncestors*, *analyzeLeaf*, *analyzeDirectChild* and *analyzeSibling*(lines 5 - 8) are similar methods such as *analyzeInputBySemanticFilters* from the functional semantic step that searches all the domain ontology. However, the difference is that the first four methods return a list of values between 0.0 and 1.0 for the structural neighbors of both the input arguments while the method used to analyze functional semantic returns, as seen, only a value that classifies it in a semantic filter.

For methods that return a list of structural neighbors of the two input arguments, the worst case occurs when the method returns the largest number of elements of the tree as a result of the ontological analysis of the neighbors.

Thus, in a worst case, all four methods of descriptive semantic analysis return two lists of size $m-1$ that correspond to the structural neighbors of each argument passed as parameter, where m is the total number of concepts of the ontology.

Thus, there is a cost g replicated for each $m-1$ elements returned from each argument within the four methods invoked.

$$T_{[05-08]} = 4g(m-1)(m-1)$$

Taking both algorithms together there is:

$T_0 + T_{03}(n+1) * T_{04}(n+1) * 4g(m-1)(m-1)$
$T_0 + T_{03} * T_{04} * (n^2+2n+1) * 4g(m-1)(m-1)$
$T_0 + T_{03} * T_{04} * (n^2+2n+1) * 4g(m^2-2m+1)$
$T_0 + g * T_{03} * T_{04}(4n^2m^2 - 8mn^2 + 4n^2 + 8nm^2 - 16nm + 8n + 4m^2 - 8m + 4)$

Having K as constant and $K \geq \{T_0, T_{03}$ and $T_{04}\}$

$T_0 + g * T_{03} * T_{04}(4n^2m^2 - 8mn^2 + 4n^2 + 8nm^2 - 16nm + 8n + 4m^2 - 8m + 4) \leq K + 4gK^2n^2m^2 - 8gK^2mn^2 + 4gK^2n^2 + 8gK^2nm^2 - 16gK^2nm + 8gK^2n + 4gK^2m^2 - 8gK^2m + 4gK^2$

The term of highest degree of the polynomial describing the algorithm is $4gK^2n^2m^2$. Therefore, based on [14], regardless of complexity g added with the execution of the computation in the order of $O(n^2m^2)$, it is possible to conclude that, in the worst case, this algorithm (see Algorithm 2) is $O(n^2m^2)$ (idem for output parameters).

The only condition for both computations to be similar would be when all methods to analyze structural neighbors return a list of neighbors of size 1 or 0. Such situation (0 or 1 value) can happen only if a domain ontology has two concepts. In real world, this is completely unworkable because it is almost impossible to model a domain with only two concepts in an ontology.

Another factor which affects the performance of descriptive semantic step is that the classification is given by an external file, a dictionary of synonyms, while in the functional semantic step the classification process is intrinsic within the ontology mapped by the semantic filters.

Using the descriptive algorithm as the sole means to discover concepts brings a performance reduction of the discovery process due to the complexity of calculation. In this sense, it is better to combine the descriptive and functional algorithm, as proposed in this paper. The descriptive semantic algorithm would only be invoked if a match fails within the functional approach. Moreover, through the descriptive algorithm it is possible to obtain the classification of new results that have been discarded by the functional semantic algorithm. Hence, in the worst case, the complexity of the overall approach is $O(n^4)$.

VI. Accuracy Analysis

For measuring the accuracy of our algorithm we used a collection of Web services (OWLS-TC). This collection has more than 500 services covering several application domains. Almost all services were retrieved from public IBM UDDI Registers and/or semi-automatically parsed from WSDL to OWL-S[15].

In order to analyze the improvement of accuracy obtained by our algorithm, we used the individual precision-recall curves technique. Precision and Recall are two indicators of effectiveness [16]. Precision is the fraction of relevant retrieved documents. A relevant result can be defined as a result that would be useful if it were returned to the user. Precision may be formalized as:

$Precision$ = |Relevant Documents ∩ Retrieved Documents|/|Retrieved Documents|

Recall is the fraction of the relevant documents that are succefully retrieved. Recall may be formalized as:

$Recall$ = |Relevant Documents ∩ Retrieved Documents|/|Relevant Documents|

In this paper both indicators were used to measure the effectiveness of our algorithm in relation to other approaches. We made a test set for each algorithm described in our related work (see section II). Firstly, we evaluate Paolucci's approach [8] and its complement by the use of a sibling filter [13]. Afterwards, we evaluate the hybrid algorithm OWLS-MX M4[5] and finally our proposed approach. Figure 5 depicts those four evaluations including our proposed algorithm.

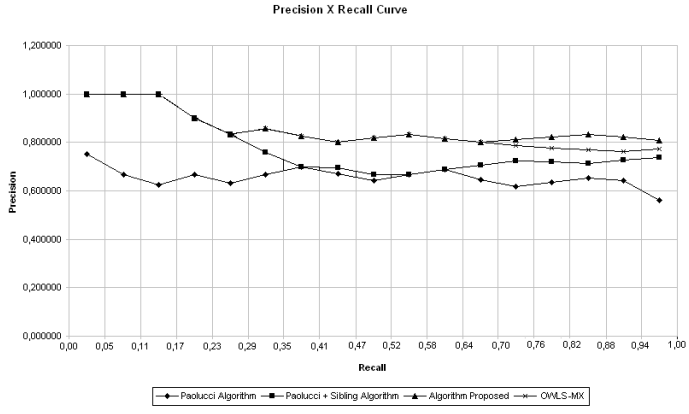

Fig. 5. Performance Evaluation - Recall X Precision

Within these experimental results we can provide the following considerations:

- Paolucci's approach was the less effective in comparison with others. This low effectiveness is due to the sole use of semantics so as to discover relevant concepts in domain ontology. Despite this technique improves the performance of this algorithm, it degrades effectiveness.
- As can be observed in Figure 5, the addition of sibling filters provides an improvement in its effectiveness. This new filter enhances Paolucci's original algorithm to return relevant concepts that were not retrieved before.
- Due to its hybrid discovery technique the OWLS-MX has better precision than the others. This approach uses both the semantic and syntactic analysis to retrieve relevant concepts, thus increasing considerably the effectiveness of this algorithm as shown in Figure 5. For a deeper analysis of this algorithm we suggest reading [5].
- In spite of the fact that OWLS-MX provides relevant results, section IV presented some of its major drawbacks. Our proposed algorithm goes beyond, further on returning relevant results as OWLS-MX, it presents other important relations between concepts due to its structural analysis. The contribution of our descriptive semantic algorithm can be highlighted in Figure 5. The line that denotes our proposed algorithm has more precision than OWLS-MX and as a consequence the other approaches.

VII. *OWL-S Discovery* TOOL

In order to implement both steps (descriptive and semantic) in a single tool, we developed a tool under GPL license, called *OWL-S Discovery*[1]. *OWL-S Discovery* was developed using a Java platform which allows run it on several plataforms like Windows, Unix, Mac etc. The *OWL-S Discovery* discovers an OWL-S Web service based on the user's request also described in OWL-S. This tool used the OWL-S API, which is strongly coupled with the Jena framework [17].

In order to demonstrate the use of *OWL-S Discovery* tool, let us suppose a scenario where a user needs the price of a certain type of car (see Figure 6).

The user should inform three parameters so as to discover similar concepts. The first parameter is a file containing the user's request (OWL-S format). This file describes the input and output parameters requested by the user. The location (url or local directory) of OWL-S file should be informed in the request field presented at *OWL-S Discovery* interface. The second parameter corresponds to the location of the Web service repository. This repository contains all Web service descriptions which match the user's request in the request field. The third parameter corresponds to a dictionary of synonyms and this field requires the location of a dictionary.

As all the above information is filled into *OWL-S Discovery* tool, the user should configure the two steps (functional and descriptive) to start the discovery of services. In "semantic step" control panel users should select which semantic filter they want to use. Users can also choose all filters. Similarly, in "descriptive step" control panel, users set the weight of each search coefficient and the minimum threshold for acceptance of correspondences.

Once all the fields are filled, users click on the discover button and the results are presented inside the console panel.

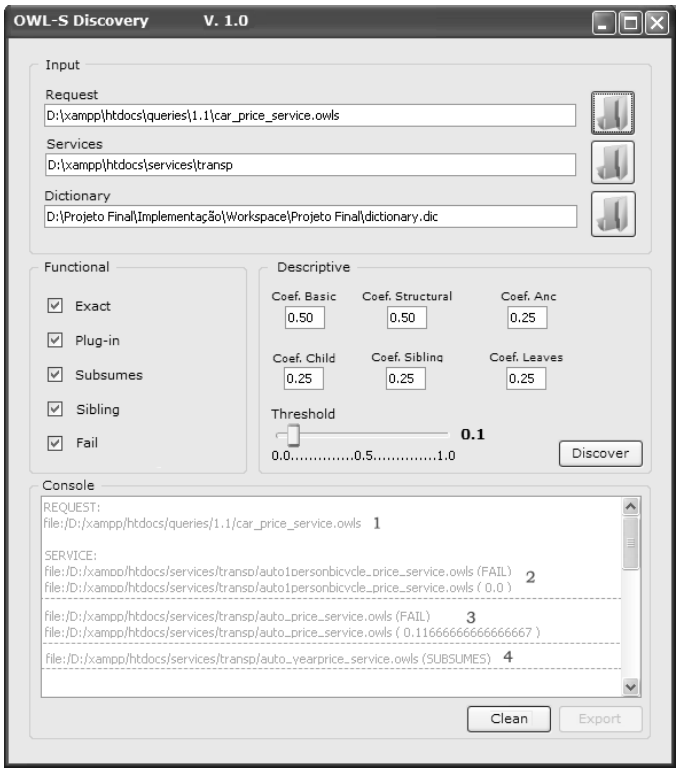

Fig. 6. OWL-S *Discovery* interface.

Results are displayed on the console as follows: on the first-line, our tool prints the user's request (OWL-S file), see number 1 in Figure 6. In subsequent lines, our tool presents the classification of each Web service according to the options selected by the user.

The general format of results is made by the name of the Web service (OWL-S file name) followed by their classification, which is one of the semantic filters (functional step) or a number that ranges from 0.0 to 1.0 (descriptive step) in the presence of a failure in the semantic step.

$<$ *name of owls Web service* $>$ $(classification)$

Three different situations may arise. The first situation (number 2 in Figure 6) presented in the console panel is a failure in both steps (functional and descriptive).

file:/.../auto1personbicycle_price_service.owls (FAIL)
file:/.../auto1personbicycle_price_service.owls (0.0)

Another situation (number 3 in Figure 6) is when the functional semantic step fails and our descriptive semantic step executes and then it finds some similarities between the concepts.

file:/.../auto_price_service.owls (FAIL)
file:/.../auto_price_service.owls (0.116667)

[1]Available to download at: http://sourceforge.net/projects/owls-discovery/

Finally there is a situation (number 4 in Figure 6) where the functional semantic step does not fail, therefore our descriptive semantic step is not triggered.

file:/.../auto_yearprice_service.owls (SUBSUMES)

Thus we presented all possible situations covered by our approach, highlighting our main contributions.

VIII. Conclusion and Future Work

We presented an approach for Web service discovery using a combination of two algorithms and the *OWL-S Discovery* tool. Our approach is entirely semantic because both algorithms (functional and descriptive) can lead to semantics in domain ontology. Even if the descriptive semantic approach can retrieve the same results as the functional semantic, its individual use has a computational cost that can make it unviable. Thus, in our approach, the descriptive semantic algorithm is only invoked if the functional semantic reports a failure (match fails) between the correspondence of two concepts.

A deeper analysis can highlight that the descriptive approach can deal with situations not previously covered by purely semantic approaches nor other techniques such as syntactic textual analysis of the concepts. We argue that discovery based on semantic filters in combination with an ontological structural analysis can increase accuracy and performance for retrieving better similar correspondences. To the best of our knowledge no other work has combined such two approaches and successfully presented better results.

We performed some experiments so as to validate our approach. Firstly we analyzed the complexity of our algorithm and we conclude that it was better to have a hybrid solution than to use each algorithm separately. Further we analyze the improvement of accuracy obtained by our algorithm based on two indicators of effectiveness: Precision and Recall. We concluded that our approach had better results than previous work. Finally we developed a tool called OWL-S Discovery to disseminate our algorithm and facilitate its usage.

As future work, we would like to expand our approach by using preconditions and effects so as to improve accuracy and also use Wordnet [18] as a dictionary which will give us a standardized and complete synonyms set.

References

[1] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. R. Payne, "OWL-S: Semantic Markup for Web Services," *Internet Computing, IEEE*, vol. 1, pp. 72–81, 2004.

[2] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Wsmo-web service modeling ontology," *DERI Working Draft*, vol. 14, pp. 77–106, 2005.

[3] J. Kopecky, T. V. C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.

[4] M. Paolucci, T. Kawamura, and J. Blasio, "A preliminary report of a public experiment of a semantic service matchmaker combined with a uddi business registry," in *1st International Conference on Service Oriented Computing (ICSOC 2003), Trento, Italy*, 2003.

[5] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM New York, NY, USA, 2006, pp. 915–922.

[6] M. Klusch, P. Kapahnkeand, and F. Kaufer, "Evaluation of wsml service retrieval with wsmo-mx," in *ICWS'08: Proceedings of the 2008 IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 401–408.

[7] M. Jager, G. Rojec-Goldmann, and G. Muhl, "Ranked Matching for Service Descriptions Using OWL-S." in *GI/VDE Fachtagung Kommunikation in Verteilten Systemen KiVS, Kaiserslautern*, 2005.

[8] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," *Proceedings of the 1st International Semantic Web Conference (ISWC2002)/LNCS*, vol. 1, pp. 333–347, 2002.

[9] J. G. S. Junior, D. Lopes, D. B. Claro, and Z. Abdelouahab, "A step forward in semi-automatic metamodel matching: Algorithms and tool," *International Conference on Enterprise Information Systems (ICEIS 2009), LNBIP*, vol. 24, pp. 137–148, 2009.

[10] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, 2003.

[11] D. Lopes, S. Hammoudi, and Z. Abdelouahab, *Schema Matching in the context of Model Driven Engineering: From Theory to Practice*. Springer, Sobh, Tarek; Elleithy, Khaled. (Org.)., 2006, pp. 219–227.

[12] D. Lopes, S. Hammoudi, J. Sousa, and A. Bontempo, "Metamodel matching: Experiments and comparison," in *International Conference on Software Engineering Advances (ICSEA 2006)*. IEEE Computer Society Press, 2006.

[13] J. Samper, F. Adell, and L. Berg, "Improving semantic web service discovery," *Journal of networks*, vol. 3, no. 1, 2008.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," 2001, mIT press Cambridge, MA.

[15] S. W. Central, "Owls-tc: An owl-s service retrieval test collection," http://projects.semwebcentral.org/projects/owls-tc/, 2008.

[16] C. J. V. Rijsbergen, "Information Retrieval," *Buttersworth, London*, 1976.

[17] B. McBride, "Jena: A semantic web toolkit," *IEEE Internet Computing*, vol. 6, no. 6, pp. 55–59, 2002.

[18] C. Fellbaum *et al.*, *WordNet: An electronic lexical database*. MIT press Cambridge, MA, 1998.