Customizable Web Services Matching and Ranking Tool: Implementation and Evaluation

Fatma Ezzahra Gmati[1], Nadia Yacoubi Ayadi[1], Afef Bahri[2], Salem Chakhar[3], and Alessio Ishizaka[3]

[1] RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, Manouba, Tunisia
fatma.ezzahra.gmati@gmail.com, nadia.yacoubi.ayadi@gmail.com
[2] MIRACL Laboratory, High School of Computing and Multimedia, University of Sfax, Sfax, Tunisia
afef.bahri@gmail.com
[3] Portsmouth Business School and Centre for Operational Research & Logistics, University of Portsmouth, Portsmouth, UK
salem.chakhar@port.ac.uk, alessio.ishizaka@port.ac.uk

Abstract. The matchmaking is a crucial operation in Web service discovery and selection. The objective of the matchmaking is to discover and select the most appropriate Web service among the different available candidates. Different matchmaking frameworks are now available in the literature but most of them present at least one of the following shortcomings: (i) use of strict syntactic matching; (ii) use of capability-based matching; (iii) lack of customization support; and (iv) lack of accurate ranking of matching Web services. The objective of this paper is thus to present the design, implementation and evaluation of the Parameterized Matching-Ranking Framework (PMRF). The PMRF uses semantic matchmaking, accepts capability and property attributes, supports different levels of customization and generates a ranked list of Web services. Accordingly, it fully overcomes the first, third and fourth shortcomings enumerated earlier and partially addresses the second one. The PMRF is composed of two layers. The role of the first layer is to parse the input data and parameters and then transfer it to the second layer, which represents the matching and ranking engine. The comparison of PMRF to iSeM-logic-based and SPARQLent, using the OWLS-TC4 datasets, shows that the algorithms supported by PMRF outperform those proposed in iSeM-logic-based and SPARQLent.

Keywords: Web service, Semantic similarity, Matchmaking, Ranking, Implementation, Performance Evaluation.

1 Introduction

Service Oriented Computing (SOC) is a distributed computing paradigm that utilizes services as the basic constructs to support the development of distributed applications, especially in heterogeneous environments. The Service-Oriented Architecture (SOA) is an element of SOC that enables service discovery, integration

and use. Web services are the most successful realization of SOA paradigm [11] that are commonly used in the areas of business-to-business integration, distributed computing and enterprise application integration [18]. A fundamental challenge of SOA paradigm, especially Web services, is service discovery, which is the process of retrieving the service most similar to the user query based on the description of functional and/or non-functional semantics [12]. An important aspect of services discovery is service matchmaking. Web service matchmaking is the method that is used to determine which of available services satisfy at best the requirements of the user while taking into account the user request and the capabilities of available services.

Web services are generally described only syntactically through WSDL and UDDI for service discovery. The UDDI in combination with WDSL provides a basic mechanism for service discovery, but lacks support for automated discovery [11]. The most used automatic service discovery approach relies on the idea of interface matching, which is based on defining the requested service through its expected input-output interface and comparing this expected interface with the input-output interfaces of available services to find matching services. However, the matchmaking based only on service profile is not sufficient, especially in the case of composite services [12]. A possible improvement to overcome such a problem is the use of the information about precondition and effects in order to capture some constraints about the entry and exit points of a service [11].

To support automatic Web service discovery and composition, a number of different semantic languages—such as OWL-S, WSMO and WSDL-S—that allow describing the functionality of services in a machine interpretable form have been proposed. The semantic Web service is a new technology and most of existing Web services still use traditional matching approaches. Accordingly, most of existing matching frameworks still suffer from at least one of the following shortcomings: (i) use of strict syntactic matching, which generally leads to low recall and precision rates [11][12][22]; (ii) use of capability-based matching, which is proven [1][12] to be inadequate in practice; (iii) lack of customization support [3][24]; and (iv) lack of accurate ranking of Web services, especially within semantic-based matching [11][22].

The objective of this paper is hence to present the Parameterized Matching-Ranking Framework (PMRF), which uses semantic matchmaking, accepts capability and property attributes, supports different levels of customization and generates a ranked list of Web services. The PMRF fully overcomes the first, third and fourth shortcomings enumerated earlier and partially addresses the second one. The comparison of PMRF to iSeM-logic-based [14] and SPARQ-Lent [23], using the OWLS-TC4 datasets, shows that the algorithms supported by PMRF behave globally well in comparison to iSeM-logic-based and SPARQ-Lent.

The paper is organized as follows. Section 2 discusses some related work. Section 3 presents the architecture of the PMRF. Section 4 deals with system implementation. Section 5 studies the performance of the PMRF. Section 6 provides the comparative study. Section 7 concludes the paper.

2 Related Work

The first and traditional matchmaking frameworks are based on strict syntactic matching. Such syntactic matching approaches only perform service discovery and service matching based on particular interface or keyword queries from the user, which generally leads to low recall and low precision of the retrieved services [11][12][22]. In order to overcome the shortcomings of strict syntactic matching approaches, some advanced techniques and algorithms have been used such as genetic algorithmic and utility function. Alternatively, many authors propose to include the concept of semantics to deal with these shortcomings. The semantic Web service is a new and active research area. In the rest of this section, we briefly discuss some recent semantic matchmaking frameworks. More detailed reviews on sematic matchmaking approaches are available in, e.g., [8][17][21][25][26].

A matchmaking approach that uses the internal process models of services as primary source of knowledge has been proposed in [11]. The basic idea of this approach is to transform the service matchmaking into model checking in which services are represented as system models and service request as set of formal properties. By using the model checking as a reasoning mechanism, the authors in [11] designed three methods supporting both exact and partial matching.

A framework for content-based semantic Web service discovery that allows users to submit unstructured free text as input has been proposed in [20]. In this framework, a collection of nouns are extracted from the input text and then used for service discovery, after a disambiguation process that makes use of the WordNet lexical database for determining the meaning of the nouns. The proposal of [20] focuses specifically on OWL-S and does not provide a means for ranking the results.

In [15], the authors propose a fuzzy matchmaking approach for semantic Web services to support an automated and veracious service discovery process in collaborative manufacturing environments. The authors first introduce a theoretical framework for fuzzy matchmaking, and a semantic annotation specification of how the needed information of web service attributes can be captured as semantic annotation for WSDL interfaces, operations, faults, and XML Schema. Then, they propose a fuzzy matchmaking algorithm for calculating the fuzzy similarity degree of web services. The developed system has been used in material selection services in the area of collaborative manufacturing.

The author in [3] presents a parameterized and highly customizable semantic matchmaking framework that supports three types of matching: functional attribute level, functional service-level and non-functional. However, the paper addresses only functional matching where a series of algorithms that support a customizable matching process have been proposed. The authors in [5] extend the work of [3] by supporting non-functional matching.

The BAX-SET PLUS, proposed in [16], is a multi-agent taxonomy-based method for categorization, search, and retrieval, of semantic Web services. The taxonomic navigation model includes a knowledge module represented by a taxonomic model, an OWL-S extension implemented and a semantic search module for a semantic Web services repository. In this model, user-selected concepts

from a taxonomy are matched against concepts contained in OWL-S service descriptions.

The Semantic Web service discovery framework (SWSD) proposed in [22] comprises a keyword-based discovery process for searching Web services that are described using a semantic language. The framework relies on natural language processing techniques in order to establish a match between a user search query and a semantic Web service description. For matching keywords with semantic Web service descriptions given in WSMO, techniques like part-of-speech tagging, lemmatization, and word sense disambiguation are used. After determining the senses of relevant words gathered from Web service descriptions and the user query, a matching process takes place. In [22], the authors propose three methods for matching sets of words or senses.

The authors in [7] design and develop a semantic framework capable of matching security capabilities of providers and security requirements of customers. The matching process is composed of two steps. The aim of the first phase is to assign a match level to each requirement-capability pair using the well-known concepts of exact, subsume, plugin, and no match introduced by [19]. In the second step, the overall match between the two policies is evaluated in order to identify the capability that matches at best for each requirement. The overall match is then defined to be the minimum among the individual match levels evaluated in the first step for each requirement-capability pair.

The Tomaco [24] is a semantic web service matching algorithm for SAWSDL. The system supports logic-based and syntactic strategies along with hybrid composite strategy. The logic-based strategy in Tomaco considers four matching cases (namely Exact, Desired, LessDesired and Fail) to determine the values of matching degree. The authors in [24] also introduce the Tomaco web application, which aims to promote wide-spread adoption of Semantic Web Services while targeting the lack of user-friendly applications in this field through a variety of configurable matching algorithms.

To improve matching effectiveness, the authors in [6] introduce first a new semantic similarity measure combining functional and process similarities. Then, a service discovery mechanism that utilises the new semantic similarity measure for service matching is proposed. The matchmaking framework is composed of two phases. The first phase uses functional attributes in order to group published Web services into services clusters. The second phase looks to identify the best matching Web services within these matching clusters.

3 System Architecture

In this section, we first introduce the conceptual and functional architectures of the PMRF. Then, we present the different supported matching and ranking algorithms.

3.1 Conceptual Architecture

Figure 1 provides the conceptual architecture of the PMRF. The inputs of the system are the specifications of the requested Web service and the different parameters. The output is a ranked list of Web services. The PMRF is composed of two layers. The role of the first layer is to parse the input data and parameters and then transfer it to the second layer, which represents the matching and ranking engine. The Matching Module filters Web service offers that match with the user specifications. The result is then passed to the Ranking Module that produces a ranked list of Web services. The assembler guarantees a coherent interaction between the different modules in the second layer.

The three main components of the second layer are:

- **Matching Module**: This component contains the different matching algorithms: basic, partially parameterized and fully parameterized matching algorithms (see Section 3.3).
- **Similarity Computing Module**: This component supports the different similarity measure computing approaches: Efficient similarity with MinEdge, Accurate similarity with MinEdge, Accurate similarity with MaxEdge and Accurate similarity with MaxMinEdge (see Section 3.4).
- **Ranking Module**: This component is the repository of score computing and ranking algorithms, namely score-based, rule-based and tree-based ranking algorithms (see Section 3.5).

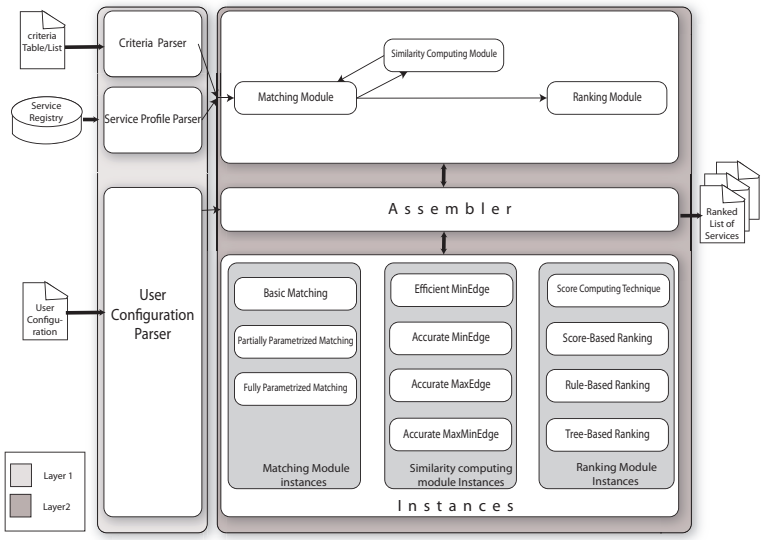


Fig. 1. Conceptual architecture of PMRF

3.2 Functional Architecture

The functional architecture of the PMRF is given in Figure 2. It shows graphically the different steps from receiving the user query (specifications of the requested Web service and the different parameters) until the delivery of the final results (ranked list of Web services) to the user.

We can distinguish the following main operations:

- The PMRF (1) receives the user query including the specifications of the desired Web service and the required parameters;
- The Matching Module (2) scans the Registry in order to identify the Web services matching the user query;
- During the matching process, the Matching Module (3) uses the Similarity Computing Module to calculate the similarity degrees;
- The Matching Module (4) delivers the Web services matching the user query to the Ranking Module;
- The Ranking Module (5) receives the matching Web services and processes them for ranking;
- During the ranking operation, the Ranking Module (6) uses the Scoring Technique to compute the scores of the Web services;
- The Ranking Module (7) generates a ranked list of Web services, which is then delivered by the PMRF to the user.

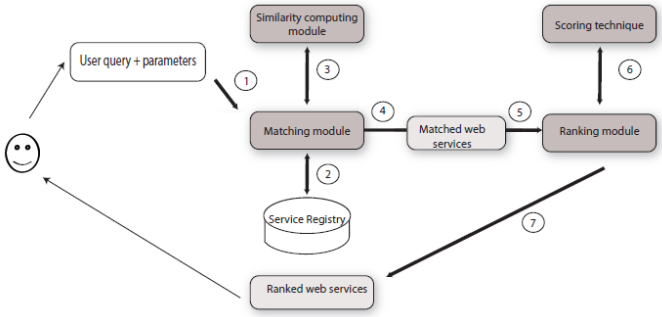


Fig. 2. Functional architecture of PMRF

3.3 Matching Algorithms

The PMRF contains three matching algorithms (basic, partially parameterized and fully parameterized) that support different levels of customization (see Table 1). The basic matching algorithm supports no customization. The partially parameterized matching algorithm allows the user to specify the set of attributes to be used in the matching. Within the fully parameterized matching algorithm,

three customizations are taken into account. A first customization consists in allowing the user to specify the list of attributes to consider. A second customization consists in allowing the user to specify the order in which the attributes are considered. A third customization is to allow the user to specify a desired similarity measure for each attribute. In the rest of this section, we present the third algorithm.

Table 1. Customization Levels for Matching Algorithms

Matching Algorithm	List of Attributes	Order of Attributes	Desired Similarity
Basic			
Partially parameterized	✓		
Fully parameterized	✓	✓	✓

In order to support all the above-cited customizations of the fully parameterized matching, we used the concept of Criteria Table (see [3]) that serves as a parameter to the matching process. A Criteria Table, C, is a relation consisting of two attributes, $C.T$ and $C.M$. The $C.T$ describes the service attribute to be compared, and $C.M$ gives the *least preferred similarity measure* for that attribute. Let $C.T_i$ and $C.M_i$ denote the service attribute value and the desired measure in the ith tuple of the relation. The $C.N$ denotes the number of tuples in C.

Let R be the service that is requested, A be the service that is advertised and C a criteria table. A sufficient match exists between R and A if for *every* attribute in $C.T$ there exists an identical attribute of R and A and the values of the attributes satisfy the desired similarity measure specified in $C.M$. Formally,

$$\begin{aligned} &\forall_i \exists_{j,k} (C.T_i = R.T_j = A.T_k) \wedge \mu(R.T_j, A.T_k) \succeq C.M_i \\ &\Rightarrow \text{SuffMatch}(R, A) \quad 1 \leq i \leq C.N. \end{aligned} \tag{1}$$

The computing of the similarity degrees $\mu(\cdot, \cdot)$ is addressed in Section 3.4. The fully parameterized matching process is formalized in Algorithm 1, which follows directly from Sentence (1). Algorithm 1 proceeds as follows. First, it loops over the attributes in the Criteria Table C and for each attribute it identifies the corresponding attribute in the requested service R and the potentially advisable service under consideration A. The corresponding attributes are appended into two different lists rAttrSet (requested Web service) and aAttrSet (advisable Web service). This operation is implemented by sentences 1 to 10 in Algorithm 1. Second, it loops over the Criteria Table and for each attribute it computes the similarity degree between the corresponding attributes in rAttrSet and aAttrSet. This operation is implemented by sentences 11 to 14 in Algorithm 1. The output of Algorithm 1 is either success (if for every attribute in C there is a similar attribute in the advertised service A with a sufficient similarity degree) or fail (otherwise).

The Criteria Table C used as parameter to Algorithm 1 permits the user to control the matched attributes, the order in which attributes are compared, as well as the minimal desired similarity for each attribute. The structure of partially matching algorithm is similar to Algorithm 1 but it takes as input an unordered collection of attributes with no desired similarities. The basic matching algorithm do no support any customization and the only possible inputs are the specification of the requested R and advertised A services. Different versions and extensions of this algorithm are available in [3][5][10].

Algorithm 1: Fully Parameterized Matching

```
   Input  : R, // Requested service.
            A, // Advertised service.
            C, // Criteria Table.
   Output: Boolean, // fail/success.
 1 while (i ≤ C.N) do
 2     while (j ≤ R.N) do
 3         if (R.T_j = C.T_i) then
 4             Append R.T_j to rAttrSet;
 5         j ⟵ j + 1;
 6     while (k ≤ A.N) do
 7         if (A.T_k = C.T_i) then
 8             Append A.T_k to aAttrSet;
 9         k ⟵ k + 1;
10     i ⟵ i + 1;
11 while (t ≤ C.N) do
12     if (μ(rAttrSet[t], aAttrSet[t]) ≺ C.M_t) then
13         return fail;
14     t ⟵ t + 1;
15 return success;
```

3.4 Computing Similarity Degrees

To compute the similarity degree, we extended the solution of [2] where the authors define four degrees of match, namely Exact, Plugin, Subsumes and Fail as default. During the matching process, the inputs and outputs of the requested Web service are matched with the inputs and outputs of the advertised Web service by constructing a bipartite graph where: (i) the vertices in the left side correspond to advertised services; (ii) the vertices in the right side correspond to the requested service; and (iii) the edges correspond to the semantic relationships between the concepts in left and right sides of the graph. Then, they assign weights to each edge as follows: Exact: w_1, Plugin: w_2, Subsumes: w_3, Fail: w_4; with $w_4 \succ w_3 \succ w_2 \succ w_1$. Finally, they apply the Hungarian algorithm to identify the complete matching that minimizes the maximum weight in the graph. The final returned similarity degree is the one corresponding to the maximum weight in the graph. Then, the selected assignment is the one representing a strict injective mapping such that the maximal weight is minimized.

The algorithms used in PMRF to compute the similarity degree between services extend the works of [2] with respect to two aspects: (i) the way the degree of match between two concepts is computed, and (ii) the optimality criterion used to compute the overall similarity degree. Concerning the computation of the degree of match, two versions are included in PMRF: efficient and accurate. In the efficient version, the degree of match is computed as in Algorithm 2 where: (i) $\equiv$: equivalence relationship; (ii) $\sqsubset_1$: direct child/parent relationship; (iii) and $\sqsupset_1$: direct parent/child relationship. In this first version, only direct related concepts are considered for Plugin and Subsume similarity measures. This will affect the precision of the algorithm since it uses a small set of possible concepts but necessarily improves the query response time (since there is no need to use inference).

Algorithm 2: Degree of Match (Efficient Version)

```
   Input  : K_R, // first concept.
            K_A, // second concept.
   Output: degree of match
 1 if (K_R ≡ K_A) then
 2     return Exact;
 3 else
 4     if (K_R ⊏_1 K_A) then
 5         return Plugin ;
 6     else
 7         if (K_R ⊐_1 K_A) then
 8             return Subsumes;
 9         else
10             return Fail ;
```

In the accurate version, we defined six similarity degrees: Exact, Plugin, Subsume, Extended-Plugin, Extended-Subsume and Fail. The degree of match in this version is calculated according to Algorithm 3 where: (i) $\equiv$: equivalence relationship; (ii) $\sqsubset_1$: direct child/parent relationship; (iii) $\sqsupset_1$: direct parent/child relationship; (iv) $\sqsubset$: indirect child/parent relationship; and (v) $\sqsupset$: indirect parent/child relationship. In Algorithm 3, indirect concepts are considered through Extended-Plugin and Extended-Subsume similarity measures.

The second extension of [2]'s work concerns the the optimality criterion used to compute the overall similarity value. The optimality criterion used in [2] is designed to minimize the false positives and the false negatives. In fact, minimizing the maximal weight would minimize the edges labeled Fail. However, the choice of $\max(w_i)$ as a final return value is restrictive and the risk of false negatives in the final result is higher. To avoid this problem, we propose to consider both $\max(w_i)$ and $\min(w_i)$ as pertinent values in the matching. A further discussion of similarity degree computing is available in [9].

Algorithm 3: Degree of Match (Accurate Version)

```
Input  : K_R, // first concept.
         K_A, // second concept.
Output: degree of match//
1  if (K_R ≡ K_A) then
2      return Exact;
3  else
4      if (K_R ⊏_1 K_A) then
5          return Plugin;
6      else
7          if (K_R ⊐_1 K_A) then
8              return Subsume;
9          else
10             if (K_R ⊏ K_A) then
11                 return Extended-Plugin;
12             else
13                 if (K_R ⊐ K_A) then
14                     return Extended-Subsume;
15                 else
16                     return Fail;
```

3.5 Ranking Algorithms

The PMRF supports three ranking algorithms: score-based, rule-based and tree-based. The first algorithm relies on the scores only. The second algorithm defines and uses a series of rules to rank Web services. It permits to solve the ties problem encountered by the score-based ranking algorithm. The tree-based algorithm, which is based on the use of a tree data structure, permits to solve the problem of ties of the first algorithm. In addition, it is computationally better than the rule-based ranking algorithm. The score-based ranking is given in Algorithm 4. The rule-based and tree-based ranking algorithms are available in [10] and [9], respectively. The main input of the score-based ranking algorithm is a list mServices of matching Web services. The function ComputeNormScores in Algorithm 4 permits to calculate the normalized scores of Web services. It implements the idea we proposed in [10]. The score-based ranking algorithm uses then an *insertion sort* procedure (implemented by lines 3-7 in Algorithm 4) to rank the Web services based on their normalized scores.

The list mServices used as input to Algorithm 4 has the following generic definition:

$$(A_i, \mu(A_i.T_1, R.T_1), \cdots, \mu(A_i.T_N, R.T_N)),$$

where: A_i is an advertised service, R is the requested service, N the total number of attributes and for $j \in \{1, \cdots, N\}$, $\mu(A_i.T_j, R.T_j)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j.

The list mServices will be first updated by function ComputeNormScores and it will have the following new generic definition:

$$(A_i, \mu(A_i.T_1, R.T_1), \cdots, \mu(A_i.T_N, R.T_N), \rho'(A_i)),$$

where: A_i, R, N and $\mu(A_i.T_j, R.T_j)$ $(j = 1, \cdots, N)$ are as above; and $\rho'(A_i)$ is the normalized score of advertised Web service A_i.

Algorithm 4: Score-Based Ranking

```
Input  : mServices,// List of matching Web services.
         N,// Number of attributes.
Output: mServices,// Ranked list of Web services.
1 mServices ← ComputeNormScores(mServices,N);
2 r ← length(mServices);
3 for (i = 1 to r − 1) do
4 |   j ← i;
5 |   while (j ≥ 0 ∧ mServices[j − 1, N + 2] > mServices[j, N + 2]) do
6 |   |   swap mServices[j, N + 2] and mServices[j − 1, N + 2];
7 |   |   j ← j − 1;
8 return mServices;
```

Based on the discussion in Section 3.4, we designed two versions for computing similarity degrees. Accordingly, two versions can be distinguished for the definition of the list mServices at the input level, along with the way the similarity degrees are computed. The first version is as follows:

$$(A_i, \mu_{\max}(A_i.T_1, R.T_1), \cdots, \mu_{\max}(A_i.T_N, R.T_N)),$$

where: A_i, R and N are as above; and $\mu_{\max}(A_i.T_j, R.T_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **maximum weight** in the matching graph.

The second version of mServices is as follows:

$$(A_i, \mu_{\min}(A_i.T_1, R.T_1), \cdots, \mu_{\min}(A_i.T_N, R.T_N)),$$

where A_i, R and N are as above; and $\mu_{\min}(A_i.T_j, R.T_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **minimum weight** in the matching graph.

To obtain the final rank, we need to use these two versions separately and then combine the obtained rankings. However, a problem of ties may occur since several Web services may have the same scores with both versions. The tree-based ranking algorithm [9] permits to solve this problem.

4 System Implementation

In this section, we first present the different tools and the strategy used to develop PMRF. Then, we present the customization support interface. Finally, we comment on the user/provider acceptability issues.

4.1 Implementation Tools and Strategy

To develop the PMRF, we have used the following tools: (i) Eclipse IDE as the developing platform, (ii) OWLS-API to parse the OWLS service descriptions, and (iii) OWL-API and the Pellet-reasoner to perform the inference for computing the similarity degrees. In order to minimize resources consumption (especially memory), we used the following procedure for implementing the inference operation: (1) A local Ontology is created at the start of the matchmaking process. The incremental classifier class, taken from the Pellet reasoner library, is associated to this Ontology. (2) The service parser based on the OWLs-API retrieves the Uniform Resource Identifier (URI) of the attributes values of each service and the concepts related to these URIs are added incrementally to the local Ontology and the classifier is updated accordingly. (3) In order to infer the semantic relations between concepts, the similarity measure module uses the knowledge base constructed by the incremental classifier. Figure 3 provides an extract from the class Matchmaker. In this figure, we can see the input and output functions. The latter contains the call for the matching and ranking operations.

```
27 public class Matchmaker implements IMatchmakerPlugin {
28
29         PelletReasoner reasoner=new PelletReasoner();
30         ServiceTuple query;
31         ArrayList<ServiceTuple> offers=new ArrayList<ServiceTuple>();
32
33
34         public Matchmaker()
35         {}
36
37         @Override
38         public void input(URL arg0) {
39             try {
40                     ServiceTuple service=new ServiceTuple(arg0,reasoner);
41                     offers.add(service);
42                     System.out.println("helloooo");
43             } catch (Exception e) {
44                     e.printStackTrace();
45             }
46         }
47         @Override
48         public Hashtable<URL, Vector<URL>> query(URL arg0)
49         {
50         Hashtable<URL,Vector<URL>> finalOutput=new Hashtable<URL,Vector<URL>>();
51         try
52         {
53             query=new ServiceTuple(arg0,reasoner);
54             /*
55              * ************************************************************************
56              * We first perform the matching
57              * ************************************************************************
58              */
59             Group initialGroup=new Group();
60             for(ServiceTuple serviceAd:offers)
61             {
62                 match(query,serviceAd,reasoner);
63                 initialGroup.addAService(serviceAd);
64             }
65             /*
66              * ************************************************************************
67              * We secondly perform the ranking
68              * ************************************************************************
69              */
70             Node<Group>  root= new Node<Group>();
71             root.setData(initialGroup);
```

Fig. 3. Extract from the Class Matchmaker

4.2 Customization Support

The parametrization interface of the PMRF is given in Figure 4. The PMRF permits the user to choose the type of algorithm to use and to specify the criteria table to consider during the matching. The PMRF offers three matching algorithms (basic, partially parameterized and fully parameterized) and three ranking algorithms (score-based, rule-base and tree-based). In addition, the PMRF supports different aggregation levels: attribute level and service level. The attribute-level matching involves capability and property attributes and consider each attribute independently of the others. In this type of matching, the PMRF offers two types of aggregation, namely conjunctive and disjunctive, where the individual (for each attribute) similarity degrees are combined using either AND or OR logical operators. The service-level matching considers capability and property attributes but the matching operation involves attributes both independently and jointly.

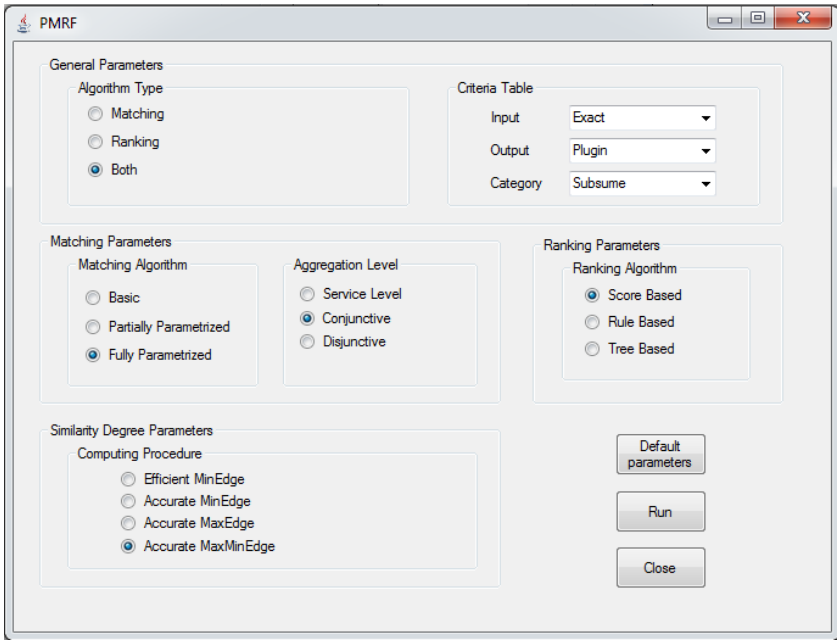


Fig. 4. Parametrization interface

The PMRF also allows the user to select the procedure to use for computing the similarity degrees. Four procedures are supported by the system: efficient similarity with MinEdge, accurate similarity with MinEdge, accurate similarity with MaxEdge and accurate similarity with MaxMinEdge.

4.3 User/Provider Acceptability Issues

One important characteristic of the proposed framework is its configurability by allowing the user to specify a set of parameters and apply different algorithms supporting different levels of customization. This, however, leads to the problem of user/provider acceptability and ability to specify the required parameters, especially the criteria Table. Indeed, the specification of these parameters may require some cognitive effort from the user/provider.

A possible solution to reduce this effort is to use a predefined Criteria Table. This solution can be further enhanced by including in the framework some appropriate Artificial Intelligence techniques to learn from the previous choices of the user.

Another possible solution to reduce the cognitive effort consists in exploiting the context of the user queries. First, the description of elementary services can be textually analysed and based on the query domain, the system uses either the efficient or the accurate versions of the similarity measure computing algorithm. Second, a global time limit to the matchmaking process can be used to orient the system towards the version that should be used. Third, the context of the query in the workflow can be used to determine the level of customization needed and also in the generation of a suitable Criteria Table.

A more advanced solution consists in combining all the idea cited above.

5 Performance Evaluation

In this section, we evaluate the performance of the different algorithms supported by the PMRF.

5.1 Evaluation Framework

To evaluate the performance of the PMRF, we used the Semantic Matchmaker Evaluation Environment (SME2) [13], which is an open source tool for testing different semantic matchmakers in a consistent way. The SME2 uses OWLS-TC collections to provide the matchmakers with Web service descriptions, and to compare their answers to the relevance sets of the various queries. The SME2 provides several metrics to evaluate the performance and effectiveness of a Web service matchmaker. The metrics that have been considered in this paper are: precision and recall, average precision, query response time and memory consumption. The definitions of these metrics are given in [13].

Experimentations have been conducted on a Dell Inspiron 15 3735 Laptop with an Intel Core i5 processor (1.6 GHz) and 2 GB of memory. The test collection OWLS-TC4 that has been used consists of 1083 Web service offers described in OWL-S 1.1 and 42 queries. Figure 5 provides an Ontology example (concerning health insurance) that has been used for the experimentations.

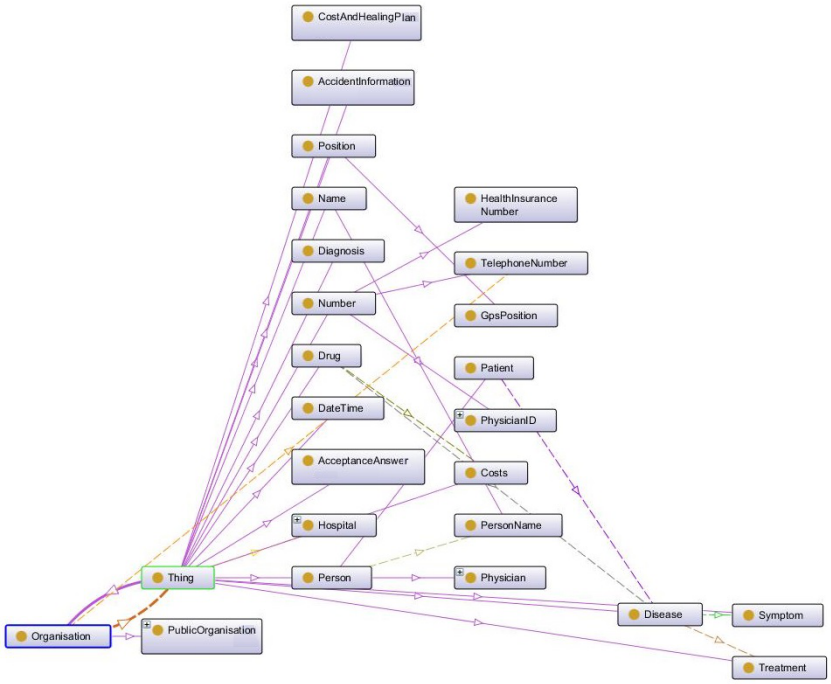


Fig. 5. Ontology example about Health Insurance

5.2 Performance Evaluation Analysis

To study the performance of the different modules supported by the PMRF, we implemented seven plugins (see Table 2) to be used with the SME2 tool. Each of these plugins represents a different combination of the matching, similarity computing and ranking algorithms. Figure 6 shows the main function of the SME2 plugin associated with Configuration 5.

Table 2. Configurations Used for Comparison

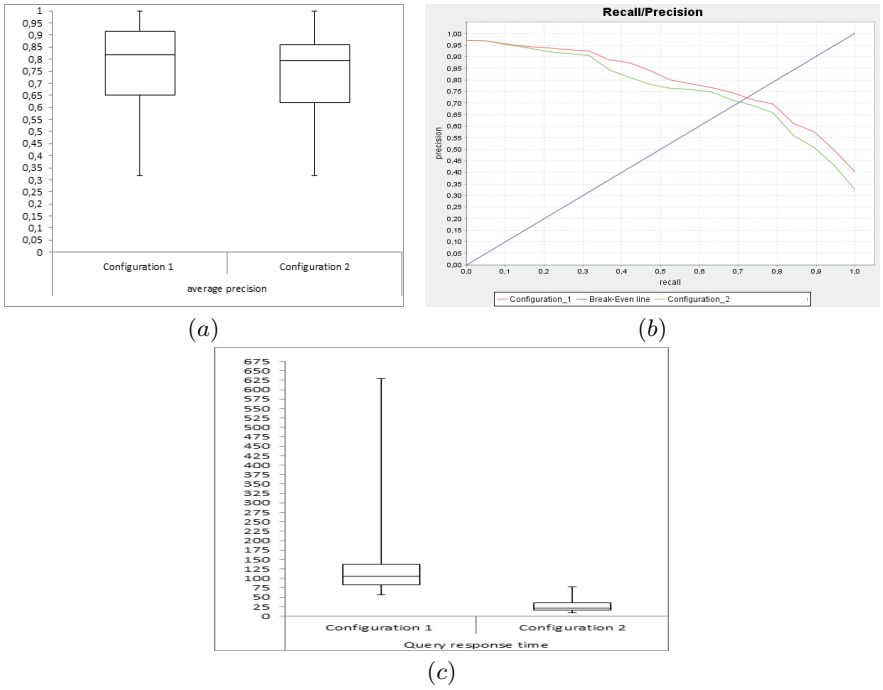
Configuration Number	Similarity Measure	Matching Algorithm	Ranking Algorithm
1	Accurate MinEdge	Basic	Basic
2	Efficient MinEdge	Basic	Basic
3	Accurate MaxEdge	Basic	Basic
4	Accurate MinEdge	Fully Parameterized	Basic
5	Accurate MaxMinEdge	Basic	RankMinMax
6	Accurate MinEdge	Basic	Rule Based
7	Efficient MinEdge	Basic	Rule Based

The difference between configurations 1 and 2 is the similarity measure module instance: configuration 1 employs the **Accurate MinEdge** instance while

```
public Hashtable<URL, Vector<URL>> query(URL arg0)
        {
        Hashtable<URL,Vector<URL>> finalOutput=new Hashtable<URL,Vector<URL>>();
        try
        {
            query=new ServiceTuple(arg0,reasoner);
            /*
             * ************************************************************************
             * We first perform the matching
             * In the same time we create the initial group
             * the initial group represents the data of the root node
             * It contains all services
             * ************************************************************************
             */
            Group initialGroup=new Group();
            for(ServiceTuple serviceAd:offers)
            {
                match(query,serviceAd,reasoner);
                initialGroup.addAService(serviceAd);
            }
            /*
             * ************************************************************************
             * We secondly perform the ranking
             * ************************************************************************
             */
            /*
             * We Create the node we fill it with the initial group and set it as a root for
             the tree
             */
            Node<Group>  root= new Node<Group>();
            root.setData(initialGroup);

            Tree tree=new Tree();
            tree.setRootElement(root);
            /*
             * We instantiate a sorting instance, it is the class that will perform all the
             ranking procedures
             */
            Sorting sort=new Sorting();
            /*
             * We generate the tree
             */
            ArrayList<Node<Group>> firstLevelChildren= sort.rankMinMaxArb(root,2);
            root.setChildren(firstLevelChildren);

            for(Node<Group> firstLevelChild: firstLevelChildren)
            {
                if(!firstLevelChild.getData().hasSingleService())
                {
                    ArrayList<Node<Group>> secondLevelChildren=
                    sort.rankMinMaxArb(firstLevelChild,1);
                    firstLevelChild.setChildren(secondLevelChildren);
                    for(Node<Group> secondLevelChild: secondLevelChildren)
                    {
                        if(!secondLevelChild.getData().hasSingleService())
                        {
                            ArrayList<Node<Group>> thirdLevelChildren =
                            sort.rankMinMaxArb(secondLevelChild,3);
                            secondLevelChild.setChildren(thirdLevelChildren);
                        }
                    }
                }
            }
          /*
           * We generate the final ranked list of services
           */
           List<Node<Group>> rankedServices=new ArrayList<Node<Group>>();
           tree.walk(root, rankedServices);
           /*
            * We assign the ranked services into the final data structure
            */
            Vector<URL> output=new Vector<URL>();
            for(Node<Group> node:rankedServices)
            {
                output.add(node.getData().getSingleService().serviceDocument);
            }
            finalOutput.put(arg0, output);

          }
          catch (Exception e) {
              e.printStackTrace();
          }
          return finalOutput;
      }
```

Fig. 6. Main function of the SME plugin associated with configuration 5

the second employs the **Efficient MinEdge** instance. Figure 7(a) shows the Average Precision and Figure 7(b) illustrates the Recall/Precision plot of configurations 1 and 2.

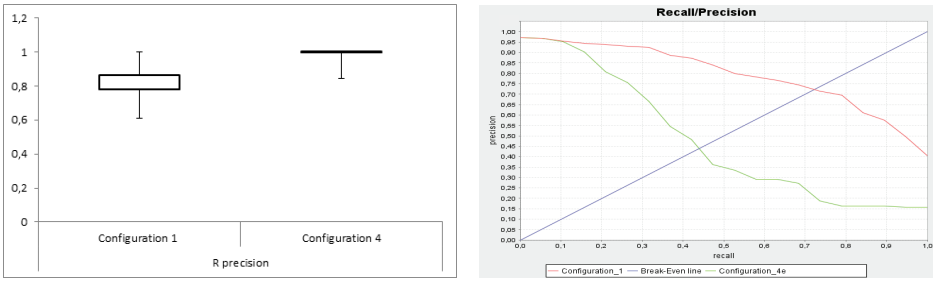


(a) (b) (c)

Fig. 7. Config. 1 *vs* Config. 2: (a) Average Precision, (b) Recall/Precision and (c) Query Response Time

We can see that configuration 1 outperforms configuration 2 for these two metrics. This is due to the use of logical inference, that obviously enhances the precision of the first configuration. In Figure 7(c), however, configuration 2 is shown to be remarkably faster than configuration 1. This is due to the inference process used in configuration 1 that consumes considerable resources.

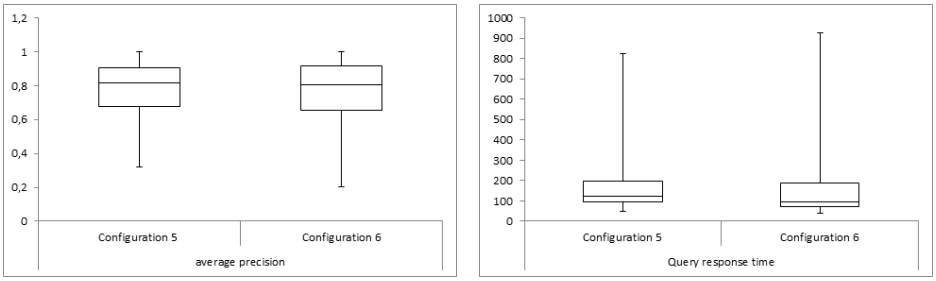
The configurations 1 and 4 use different matching module instances. The first configuration is based on the basic matching algorithm while the second uses the fully parameterized matching. Figure 8(a) shows the Average Precision metric results. It is easy to see that configuration 4 outperforms configuration 1. This is due to the fact that the Criteria Table restricts the results to the most relevant Web services, which will have the best ranking leading to a higher Average Precision. Figure 8(b) illustrates the Recall/Precision plot. It shows that configuration 4 has a low recall rate. The overly restrictive Criteria Table explains these results, since it fails to return some relevant services.



(a) (b)

Fig. 8. Config. 1 *vs* Config. 4: (*a*) Average Precision and (*b*) Recall/Precision

The difference between configurations 5 and 6 is the ranking module instance and the similarity computing procedure. The first uses the tree-based ranking algorithm while the second employs the rule-based ranking algorithm. Figure 9(*a*) shows that configuration 5 has a slightly better Average Precision than configuration 6 while Figure 9(*b*) shows that configuration 6 is obviously faster than configuration 5.



(a) (b)

Fig. 9. Config. 5 *vs* Config. 6: (*a*) Average Precision and (*b*) Query Response Time

6 Comparative Study

We compared the results of the PMRF matchmaker with SPARQLent [23] and iSeM [14] frameworks. Configuration 7 Table 2 was chosen to perform this comparison. The SPARQLent is a logic-based matchmaker based on the OWL-DL reasoner Pellet to provide exact and relaxed Web services matchmaking. The iSeM is an hybrid matchmaker offering different filter matchings: logic-based, approximate reasoning based on logical concept abduction for matching Inputs and Outputs. We considered only the I-O logic-based in this comparative study.

We note that SPARQLent and iSeM consider preconditions and effects of Web services, which are not considered in our work.

The Average Precision is given in Figure 10(a). This figure shows that the PMRF has a more accurate Average Precision than iSeM logic-based and SPARQLent, leading to a better ranking precision than the two other frameworks. In addition, the generated ranking is more fine-grained than SPARQLent and iSeM. This is due to the score-based ranking that gives a more coarse evaluation than a degree aggregation. Indeed, SPARQLent and iSeM approaches adopt a subsumption-based ranking strategy as described in [19], which gives equal weights to all similarity degrees.

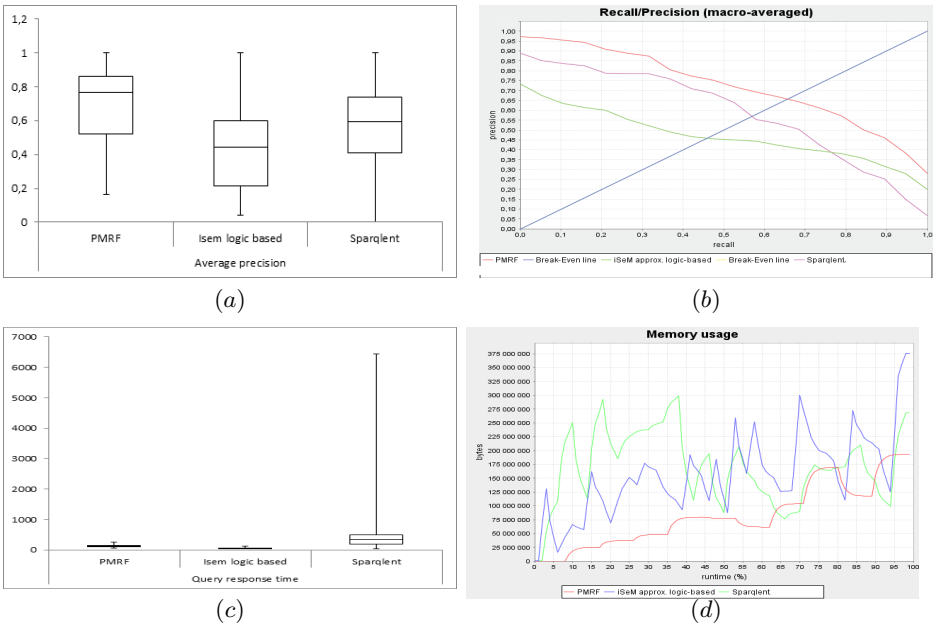


Fig. 10. Comparative study: (a) Average Precision, (b) Recall/Precision, (c) Query Response Time and (d) Memory Usage

Figure 10(b) presents the Recall/Precision of the PMRF, iSeM logic-based and SPARQLent. This figure shows that PMRF recall is significantly better than both iSeM logic-based and SPARQLent. This means that our approach is able to reduce the amount of false positives (see [2] for a discussion on the false positives problem).

The comparison of the Query Response Time of the PMRF, logic-based iSeM and SPARQLent is shown in Figure 10(c). The experimental results show that the PMRF is faster than SPARQLent (760ms for SPARQLent versus 128ms for PMRF) and slightly less faster than logic-based iSeM (65ms for iSeM). We note that SPARQLent has especially high query response time if the query include

preconditions/effects. The SPARQLent is also based on an OWL DL reasoner, which has an expensive processing. PMRF and iSeM have close query response time because both consider direct parent/child relations in a subsumption graph, which reduces significantly the query processing. The PMRF highest query response time limit is 248ms.

Figure 10(d) shows the Memory Usage for PMRF, iSeM logic-based and SPARQLent. It is easy to see that PMRF consumes less memory than iSeM logic-based and SPARQLent. This can be explained by the fact that the PMRF does not require a reasoner (in the case of Configuration 7) neither a SPARQL queries in order to compute similarities between concepts. We note, however, that the memory usage of the PMRF increases monotonically in contrast to SPARQLent.

7 Conclusion and Future Work

In this paper, we presented a highly customizable framework, called PMRF, for matching and ranking Web services. The conceptual and algorithmic solutions on which PMRF relies permit to fully overcome the first, third and fourth shortcomings of existing matchmaking frameworks. The second shortcoming is partially addressed in this paper. All the algorithms have been evaluated using the OWLS-TC4 datasets. The evaluation has been conducted employing the SME2 tool. The results show that the algorithms behave globally well in comparison to iSeM-logic-based and SPARQLent.

There are several topics that need to be addressed in the future. The first topic concerns the support of non-functional matching. In this respect, several existing approaches consider attributes related to the Quality of Service (QoS) in the matching process (e.g. [1]). In the future, we intend to enhance the framework to support QoS attributes for matching and ranking of Web services. The work of [5] could be a start point.

The second topic focuses on the use of multicriteria evaluation. Indeed, there are few proposals that explicitly use multicriteria evaluation to support matching and ranking of Web services (e.g. [4]). In the future, we intend to use a well-known and more advanced multicriteria method, namely the Dominance-based Rough Set Approach (DRSA), which is particularity suitable for including the QoS attributes in the matching process.

The last topic relates to the support of the imprecision and uncertainty in matching and ranking of Web services. In this paper, we assumed that the data and user parameters are crisply defined. In the future, we intend to enhance the proposed framework by conceiving and developing algorithms and tools that support the imprecision and uncertainty aspects in Web services matching and ranking.

References

1. A. Alnahdi, S. H. Liu, and A. Melton. Enhanced web service matchmaking: A quality of service approach. In *2015 IEEE World Congress on Services*, pages

341–348, New York, USA, June 27 - July 2 2015.

2. U. Bellur and R. Kulkarni. Improved matchmaking algorithm for semantic Web services based on bipartite graph matching. In *IEEE International Conference on Web Services*, pages 86–93, Salt Lake City, Utah, USA, 9-13 July 2007.
3. S. Chakhar. Parameterized attribute and service levels semantic matchmaking framework for service composition. In *Fifth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2013)*, pages 159–165, Seville, Spain, 27 January - 1 February 2013.
4. S. Chakhar, S. Haddad, L. Mokdad, V. Mousseau, and S. Youcef. Multicriteria evaluation-based framework for composite web service selection. In R. Bisdorff, L.C. Dias, V. Mousseau, M. Pirlot, and P. Meyer, editors, *Evaluation and decision models with multiple criteria*, International handbooks on information systems series, pages 167–200. Springer, 2015.
5. S. Chakhar, A. Ishizaka, and A.W. Labib. Semantic matching-based selection and QoS-Aware classification of web services. In V. Monfort and K.-H. Krempels, editors, *Proceedings of the 10th international conference, WEBIST 2014, Barcelona, Spain, 3-5 April, 2014, Revised Selected Papers:*, Lecture Notes in Business Information Processing, pages 96–112. Springer, Switzerland, 2015.
6. F. Chen, M. Li, H. Wu, and L. Xie. Web service discovery among large service pools utilising semantic similarity and clustering. *Enterprise Information Systems*, 11(3):452–469, 2017.
7. G. Di Modica and O. Tomarchio. Matchmaking semantic security policies in heterogeneous clouds. *Future Generation Computer Systems*, 55:176–185, 2016.
8. D.H. Elsayed and A. Salah. Semantic web service discovery: A systematic survey. In *The 11th International Computer Engineering Conference (ICENCO 2015)*, pages 131–136, Dec 2015.
9. F.-E. Gmati, N. Yacoubi Ayadi, A. Bahri, S. Chakhar, and A. Ishizaka. A tree-based algorithm for ranking web services. In V. Monfort and K.-H. Krempels, editors, *The 11th International Conference on Web Information Systems and Technologies (WEBIST 2015), Lisbon, Portugal, May 20-22:*, pages 170–178. SciTePress, 2015.
10. F.-E. Gmati, N. Yacoubi Ayadi, and S. Chakhar. Parameterized algorithms for matching and ranking web services. In R. Meersman, H. Panetto, T. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T. Sellis, editors, *On the move to meaningful internet systems, OTM 2014 conferences:confederated international conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31*, Lecture Notes in Computer Science, pages 784–791. Springer, Berlin Heidelberg, 2014.
11. A. Günay and P. Yolum. Service matchmaking revisited: An approach based on model checking. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):292–309, 2010.
12. M. Khater, S.E. Habibeche, and M. Malki. Behaviour approach for composite OWL-S services discovery. *International Journal of Business Information Systems*, 25(1):55–70, 2017.
13. M. Klusch, M. Dudev, J. Misutka, P. Kapahnke, and M. Vasileski. *SME*^{2} *Version 2.2. User Manual*. The German Research Center for Artificial Intelligence (DFKI), Germany, 2010.
14. M. Klusch and P. Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.

15. M. Liu, W. Shen, Q. Hao, J. Yan, and L. Bai. A fuzzy matchmaking approach for semantic web services with application to collaborative material selection. *Computers in Industry*, 63(3):193–209, 2012.
16. J.A.G. Luna, I.D.T. Pardo, and J.A.J. Builes. BAX-SET PLUS: A taxonomic navigation model to categorize, search and retrieve semantic web services. In F.L. Gaol, editor, *Recent progress in data engineering and internet technology*, volume 156 of *Lecture Notes in Electrical Engineering*, pages 359–364. Springer, 2013.
17. H. Nacer and D. Aissani. Semantic web services: Standards, applications, challenges and solutions. *Journal of Network and Computer Applications*, 44:134–151, 2014.
18. T. Narock, V. Yoon, and S. March. A provenance-based approach to semantic web service description and discovery. *Decision Support Systems*, 64:90–99, 2014.
19. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *The First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, 2002. Springer-Verlag.
20. D. Paulraj and S. Swamynathan. Content based service discovery in semantic web services using WordNet. In P.S. Thilagam, A.R. Pais, K. Chandrasekaran, and N. Balakrishnan, editors, *Advanced Computing, Networking and Security - International Conference, ADCONS 2011, Surathkal, India, December 16-18, 2011, Revised Selected Papers*, volume 7135 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2012.
21. G. Priyadharshini, R. Gunasri, and B.B. Saravana. A survey on semantic web service discovery methods. *International Journal of Computer Applications*, 82(11):8–11, 2013.
22. J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin. Semantic web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11):4660–4671, 2013.
23. M.L. Sbodio, D. Martin, and C. Moulin. Discovering semantic Web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):310–328, 2010.
24. T.G. Stavropoulos, S. Andreadis, N. Bassiliades, D. Vrakas, and I. Vlahavas. The tomaco hybrid matching framework for sawsdl semantic web services. *IEEE Transactions on Services Computing*, 9(6):954–967, 2016.
25. E. Toch, I. Reinhartz-Berger, and D. Dori. Humans, semantic services and similarity: A user study of semantic web services matching and composition. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(1):16–28, 2011.
26. D. Tosi and S. Morasca. Supporting the semi-automatic semantic annotation of web services: A systematic literature review. *Information and Software Technology*, 61:16–32, 2015.