

Customizable Web Services Matching and Ranking Tool: Implementation and Evaluation

Fatma Ezzahra Gmati¹, Nadia Yacoubi Ayadi¹, Afef Bahri², Salem Chakhar³,
and Alessio Ishizaka³

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: (1) use of strict syntactic matching; (2) use of capability-based matching; (3) lack of customization support; and (4) lack of accurate ranking of matching Web service. The objective of this paper is thus to present the design, implementation and evaluation of the Parameterized Matching-Ranking Framework (PMRF) that fully overcomes the first, third and fourth shortcomings cited above and partially addresses the second one. 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

Web services matchmaking is the operation of discovering and selecting the most appropriate (i.e., that responds better to the user request) Web service among the different available candidates. Different matchmaking frameworks are now available in the literature, including [2][24][26][32][39][40][42]. However, most of these frameworks present at least one of the following shortcomings: (1) use of strict syntactic matching, which generally leads to low recall and precision rates; (2) use of capability-based matchmaking, which is proven [1][10] to be inadequate in practice; (3) lack of customization support; and (4) lack of accurate ranking

of matching Web services, especially within semantic-based matching. These shortcomings are discussed in more detail in the next section.

The objective of this paper is 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 matching 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 [20] and SPARQLent [37], using the OWLS-TC4 datasets, shows that the algorithms supported by PMRF behave globally well in comparison to iSeM-logic-based and SPARQLent.

The design and development of PMRF have been influenced by several existing frameworks, especially [3][6][7][10][20][33]. Although that these proposals are based on semantics, they fail to take into account jointly the previous shortcomings. Indeed, the proposal of [3][20][37] do not support any customization while those of [6][7][10] do not propose solutions for ranking Web services. Some proposals including [5][16] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only. The proposal of [6][7] lack effective implementation. In addition, the authors do not precise how the similarity degree is computed and how the different matching Web services are ranked before provided to the user. Finally, there is a lack of effective evaluation and performance analysis of matching algorithms.

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

In this section, we first discuss each of above-cited shortcomings of existing matchmaking frameworks, namely use of strict syntactic matching, use of capability-based matching, lack of customization support and lack of accurate ranking. Then, we briefly review some similarity measure computing approaches.

2.1 Matching Type

The first and traditional matchmaking frameworks, such as Jini [2], Konark [24] and Salutation [32], 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 [27].

In order to overcome the limitation of strict syntactic matching, some advanced techniques and algorithms have been used such as genetic algorithmic [26] and utility function [40][42]). Alternatively, many authors propose to include the concept of semantics as in [3][5][11][14][16][20][25][33][37][38] to deal with the limitation of strict syntactic matching. The use of ontology eliminates

the issues caused by syntactic difference between terms since matching is now possible on the basis of concepts of ontologies used to describe input and output terms [4].

2.2 Matching Attributes

Most of existing matchmaking frameworks such as [5][14][16][25][33][38] utilize a strict capability-based matchmaking, which is proven [1][10] to be inadequate in practice. Some recent proposals, including [5][16], propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only.

The author in [6] distinguishes three types of service attributes: (i) capability attributes that directly relate to working of the service, (ii) quality attributes related to the service quality, and (iii) property attributes including all the other ones. The authors in [7] extend the works of [6] and [10] propose different match-making algorithms devoted to different types of attributes (capability, property and service quality).

2.3 Customisation Support

An important shortcoming of most of existing Web service matchmaking frameworks is the lack of customization support. To deal with this shortcoming, some authors allow the user to specify some parameters. For instance, the authors in [10] present a parameterized semantic matchmaking framework that exhibits a customizable matchmaking behavior. One important shortcoming of [10] is that the sufficiency condition defined by the authors is very strict since it requires that all the specified conditions hold at the same time. This seems to be very restrictive in practice, especially for attributes related to the service quality.

Recently, the authors in [7] extend the work of [6] and propose a series of algorithms for different types of matching. These algorithms are designed to support a customizable matching process that permits the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed for all matching types.

2.4 Ranking Support

Although the semantic matchmaking permits to avoid the problem of simple syntactic and strict capability-based matchmaking, it is not very suitable for efficient Web service selection. This is because it is difficult to distinguish between a pool of similar Web services [35]. Indeed, since we have a limited number of similarity degrees, semantic matchmaking frameworks will most often face the problem of ties when several Web services have the same similarity degree.

A possible solution to this issue is to use some appropriate techniques and some additional information to rank the Web services delivered by the semantic matching algorithm and then provide a manageable set of ‘best’ Web services to

the user from which s/he can select one Web service to deploy. Several approaches have been proposed to implement this idea [22][29][28].

Table 1 summarizes the main characteristics of the above cited frameworks. As shown in this table, the discussed frameworks fail to jointly take into account all the previously enumerated shortcomings. The proposed system PMRF uses semantic matchmaking, accepts capability and property attributes, support different levels of customization and generates a ranked list of matching Web services. It can be easily extended, based on our previous work [7][8], to support attributes related to the quality of service.

Table 1. Comparison of Matchmaking Frameworks

Matchmaker	Matching Type	Matching Attributes	Customization Support	Ranking Support	Description Language
[1]	Logical	Service quality	No	Yes	OWL
Jini[2]	Syntactic	Capability	No	No	No
Konark[24]	Syntactic	Capability	No	No	XML
Salutation[32]	Logical	Capability	No	Yes	OWL-S
MatchMaker [38]	Syntactic	Capability	No	No	DAMS, UDDI
RACER[25]	Syntactic	Capability	No	No	DAML-S
PSMF[10]	Logical	Capability	Yes	No	DAML-S, WSDL, UDDI
SPARQLent[37]	Logical	Capability	No	Yes	OWL-S
iSeM-logi-based[20]	Logical	Capability	No	Yes	OWL-S, SAWSDL
QoSeBroker[7][8]	Logical	Capability, Property, Service quality	Yes	No	OWL-S
PMRF	Logical	Capability, Property	Yes	Yes	OWL-S

2.5 Similarity Measure Computing Approaches

To overcome the shortcomings of traditional matchmaking frameworks, several authors propose to include the semantics in the matchmaking process [34]. The first semantic matchmaker have been proposed by [33]. The idea of [33] is first to compute similarity between the concepts of the compared services and tag it as Exact, Plugin, Subsumes or Fail if there is no similarity. Then, they aggregate the results into an overall degree of matching using a greedy approach. The first drawback of [33]'s algorithm is the way the degree of match is computed. Indeed, calculating a subsumes relation over a large ontology can be time consuming due to the inference process. The second drawback, as noticed by [3], concerns the problem of false positives and false negatives in the final results, which will lead to low precision and recall rates.

The authors in [10] extend and specify the similarity measure definition proposed in [33] and identify six similarity measures (namely Exact, Plugin, Subsumption, Container, Part-of and Disjoint) that can be used to measure the mapping between two conceptual annotations. However, the authors in [10] do not specify the modeling approach.

The authors in [3] propose an algorithm that relies on a bipartite graph where the vertices in the left side of the bipartite graph correspond to advertised

services while those in the right side correspond to the requested services and edges correspond to the semantic relationships between concepts in left and right sides of the graph. Then, they assign a weight to each edge and finally apply the Hungarian algorithm [23] to identify the complete matching that minimizes the maximum weight in the graph.

Table 2 compares different similarity measure computing approaches with respect to modelling techniques, level of precision and complexity.

Table 2. Comparison of Similarity Measure Computing Approaches

Approach	Modelling Technique	Precision	Complexity
[10]	Unspecified	High	Moderate
[33]	Greedy Algorithm	Low	High
[3]	Bipartite Graph	High	Moderate
Efficient Algorithm	Bipartite Graph	Moderate	Low
Accurate Algorithm	Bipartite Graph	High	Moderate

3 System Architecture

In this section, we first introduce the conceptual and functional architecture of the PMRF. Then, we present the different supported matching, similarity measuring 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 matching 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 the score computing technique and the different ranking algorithms, namely score-based, rule-based and tree-based ranking algorithms (see Section 3.5).

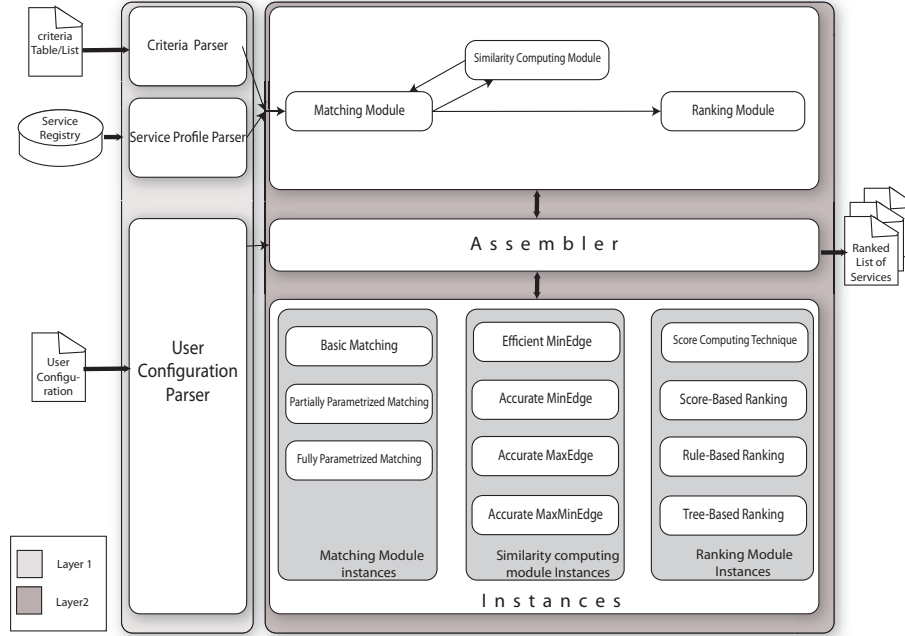


Fig. 1. Conceptual architecture of PMRF

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 matching Web services) to the user.

We can distinguish the following main operations:

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

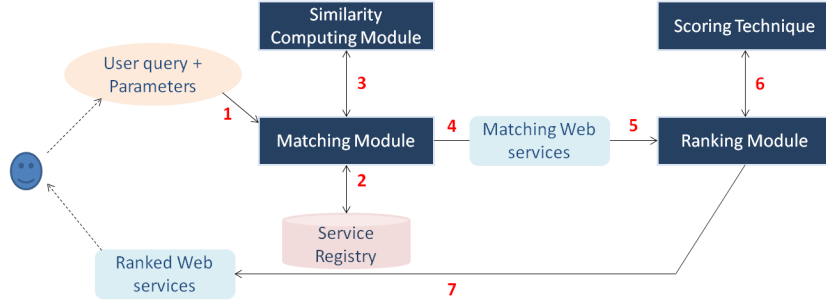


Fig. 2. Functional architecture of PMRF

3.3 Matching Algorithms

The PMRF supports three matching algorithms—basic, partially parameterized and fully parameterized—supporting different levels of customization (see Table 3). 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 3. 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, introduced by [10], that serves as a parameter to the matching process. A Criteria Table, C , is a relation consisting of two attributes, $C.A$ and $C.M$. The $C.A$ describes the service attribute to be compared, and $C.M$ gives the *least preferred similarity measure* for that attribute. Let $C.A_i$ and $C.M_i$ denote the service attribute value and the desired measure in the i th tuple of the relation. The $C.N$ denotes the number of tuples in C .

Let S^R be the service that is requested, S^A be the service that is advertised and C a criteria table. A sufficient match exists between S^R and S^A if for *every*

attribute in $C.A$ there exists an identical attribute of S^R and S^A and the values of the attributes satisfy the desired similarity measure specified in $C.M$. Formally,

$$\begin{aligned} & \forall_i \exists_{j,k} (C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \\ & \Rightarrow \text{SuffMatch}(S^R, S^A) \quad 1 \leq i \leq C.N. \end{aligned} \quad (1)$$

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: Fully Parameterized Matching

```

Input  :  $S^R$ , // Requested service.
           $S^A$ , // Advertised service.
           $C$ , // Criteria Table.
Output: Boolean, // fail/success.
1 while ( $i \leq C.N$ ) do
2   while ( $j \leq S^R.N$ ) do
3     if ( $S^R.A_j = C.A_i$ ) then
4       Append  $S^R.A_j$  to  $rAttrSet$ ;
5      $j \leftarrow j + 1$ ;
6   while ( $k \leq S^A.N$ ) do
7     if ( $S^A.A_k = C.A_i$ ) then
8       Append  $S^A.A_k$  to  $aAttrSet$ ;
9      $k \leftarrow k + 1$ ;
10   $i \leftarrow i + 1$ ;
11 while ( $t \leq C.N$ ) do
12   if ( $\mu(rAttrSet[t], aAttrSet[t]) \prec C.M_t$ ) then
13     return fail;
14    $t \leftarrow t + 1$ ;
15 return success;

```

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 S^R and the potentially advisable service under consideration S^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 S^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 S^R and advertised S^A services. Different versions and extensions of this algorithm are available in [6][7][8][13].

3.4 Computing Similarity Degrees

To compute the similarity degree, we extended the solution of [3] 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 [17] 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 degrees between services extend the works of [3] 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.

Algorithm 2: Degree of Match (Efficient Version)

```

Input :  $CA_R$ , // first concept.
          $CA_A$ , // second concept.
Output: degree of match
1 if ( $CA_R \equiv CA_A$ ) then
2   return Exact;
3 else
4   if ( $CA_R \sqsubset_1 CA_A$ ) then
5     return Plugin ;
6   else
7     if ( $CAS_R \sqsupset_1 CA_A$ ) then
8       return Subsumes;
9     else
10      return Fail ;

```

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).

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.

Algorithm 3: Degree of Match (Accurate Version)

```

Input :  $CA_R$ , // first concept.
          $CA_A$ , // second concept.
Output: degree of match//
1 if ( $CA_R \equiv CA_A$ ) then
2    $\sqsubset$  return Exact;
3 else
4   if ( $CA_R \sqsubset_1 CA_A$ ) then
5      $\sqsubset$  return Plugin;
6   else
7     if ( $CA_R \sqsupset_1 CA_A$ ) then
8        $\sqsubset$  return Subsume;
9     else
10      if ( $CA_R \sqsubset CA_A$ ) then
11         $\sqsubset$  return Extended-Plugin;
12      else
13        if ( $CA_R \sqsupset CA_A$ ) then
14           $\sqsubset$  return Extended-Subsume;
15        else
16           $\sqsubset$  return Fail;

```

The second extension [3]'s work concerns the the optimality criterion used to compute the overall similarity value. The optimality criterion used in [3] 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 [12].

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 [13] and [12], respectively.

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;
```

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 [13]. 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:

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \dots, \mu(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised service, S^R is the requested service, N the total number of attributes and for $j \in \{1, \dots, N\}$, $\mu(S_i^A.A_j, S^R.A_j)$ is the similarity measure between the requested Web service and the advertised Web service on the j th attribute A_j .

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

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \dots, \mu(S_i^A.A_N, S^R.A_N), \rho'(S_i^A)),$$

where: S_i^A , S^R , N and $\mu(S_i^A.A_j, S^R.A_j)$ ($j = 1, \dots, N$) are as above; and $\rho'(S_i^A)$ is the normalized score of advertised Web service S_i^A . The normalized score $\rho'(S_i^A)$ is computed by **ComputeNormScores**.

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:

$$(S_i^A, \mu_{\max}(S_i^A.A_1, S^R.A_1), \dots, \mu_{\max}(S_i^A.A_N, S^R.A_N)),$$

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

The second version of **mServices** is as follows:

$$(S_i^A, \mu_{\min}(S_i^A.A_1, S^R.A_1), \dots, \mu_{\min}(S_i^A.A_N, S^R.A_N)),$$

where S_i^A , S^R and N are as above; and $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ ($j = 1, \dots, N$) is the similarity measure between the requested Web service and the advertised Web service on the j th 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 [12] 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. 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.

4.2 Customization Support

The parametrization interface of the PMRF is given in Figure 3. 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-based and tree-based). In addition, the PMRF supports different aggregation levels: conjunctive-attribute level, disjunctive-attribute level and service level. The attribute-level matching involves capability and property attributes and consider each matching 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 implies attributes both independently and jointly.

Fig. 3. 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 which version 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 or Attributes List.

A more advanced solution consists in combining all the ideas 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) [19], 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 [19].

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 4 provides an Ontology example (concerning health insurance) that has been used for the experimentations.

5.2 Performance Evaluation Analysis

To study the performance of the different modules supported by the PMRF, we implemented seven plugins (see Table 4) to be used with the SME2 tool. Each

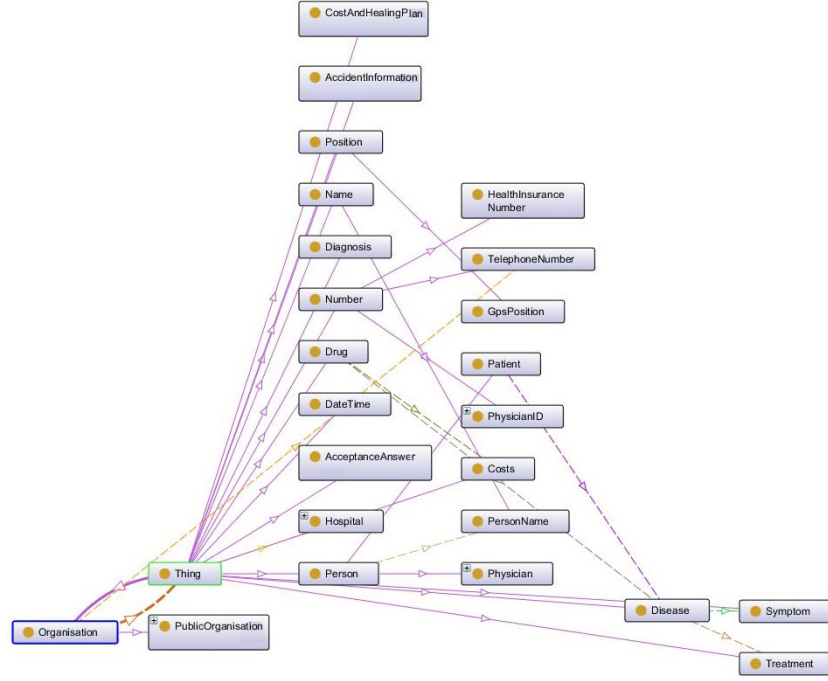


Fig. 4. Ontology example about Health Insurance

Table 4. 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

of these plugins represents a different combination of the matching, similarity computing and ranking algorithms.

The difference between configurations 1 and 2 is the similarity measure module instance: configuration 1 employs the **Accurate MinEdge** instance while the second employs the **Efficient MinEdge** instance. Figure 5(a) shows the Average Precision and Figure 5(b) illustrates the Recall/Precision plot of configurations 1 and 2. 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 5(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.

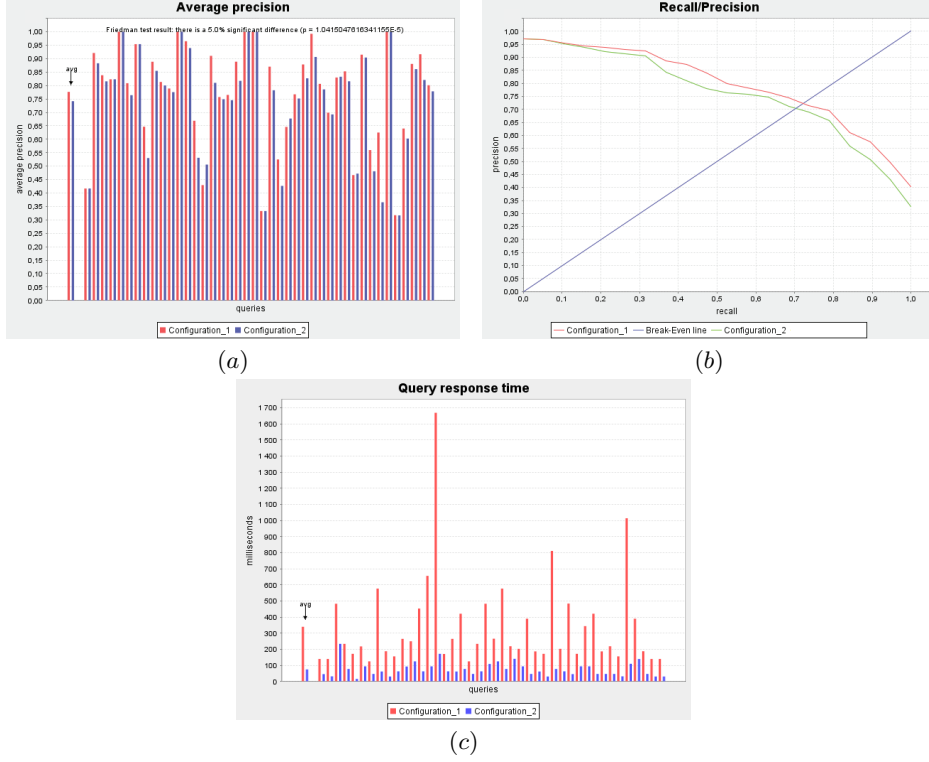


Fig. 5. Config. 1 vs Config. 2: (a) Average Precision, (b) Recall/Precision and (c) Query Response Time

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 6(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 6(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.

The difference between configurations 5 and 6 is the ranking module instance and the similarity measure computing procedure. The first uses the tree-based ranking algorithm while the second employs the rule-based ranking algorithm. Figure 7(a) shows that configuration 5 has a slightly better Average Precision

than configuration 6 while Figure 7(b) shows that configuration 6 is obviously faster than configuration 5.

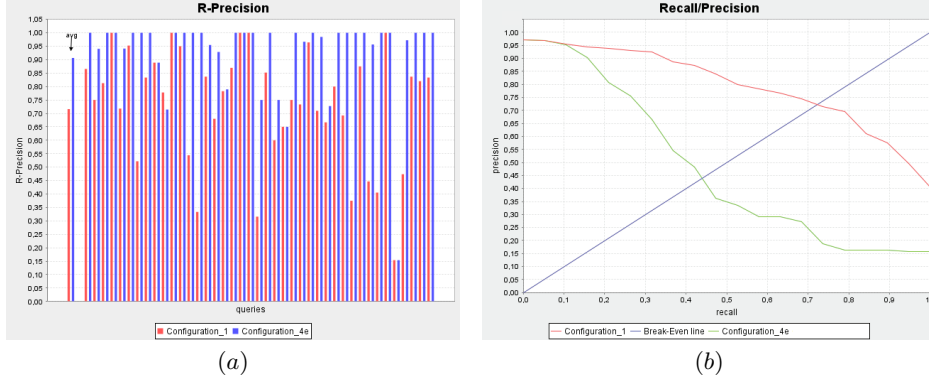


Fig. 6. Config. 1 vs Config. 4: (a) Average Precision and (b) Recall/Precision

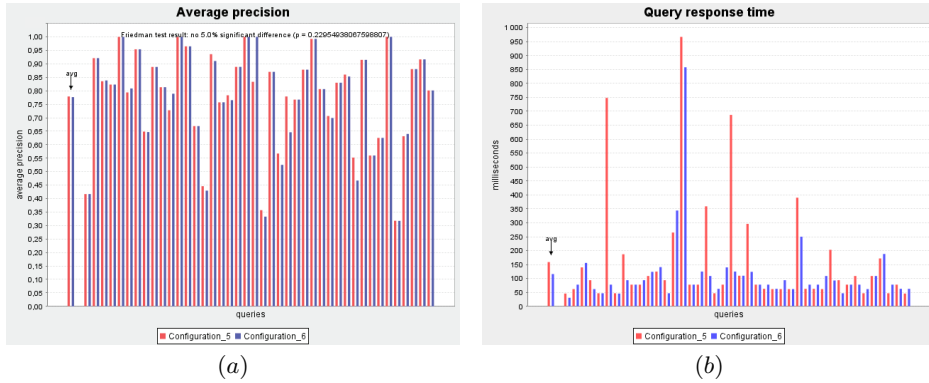


Fig. 7. Config. 5 vs Config. 6: (a) Average Precision and (b) Query Response Time

6 Comparative Study

We compared the results of the PMRF matchmaker with SPARQLent [37] and iSeM [20] frameworks. Configuration 7 Table 4 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 8(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 [33], which gives equal weights to all similarity degrees.

Figure 8(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 [3] for a discussion on the false positives problem).

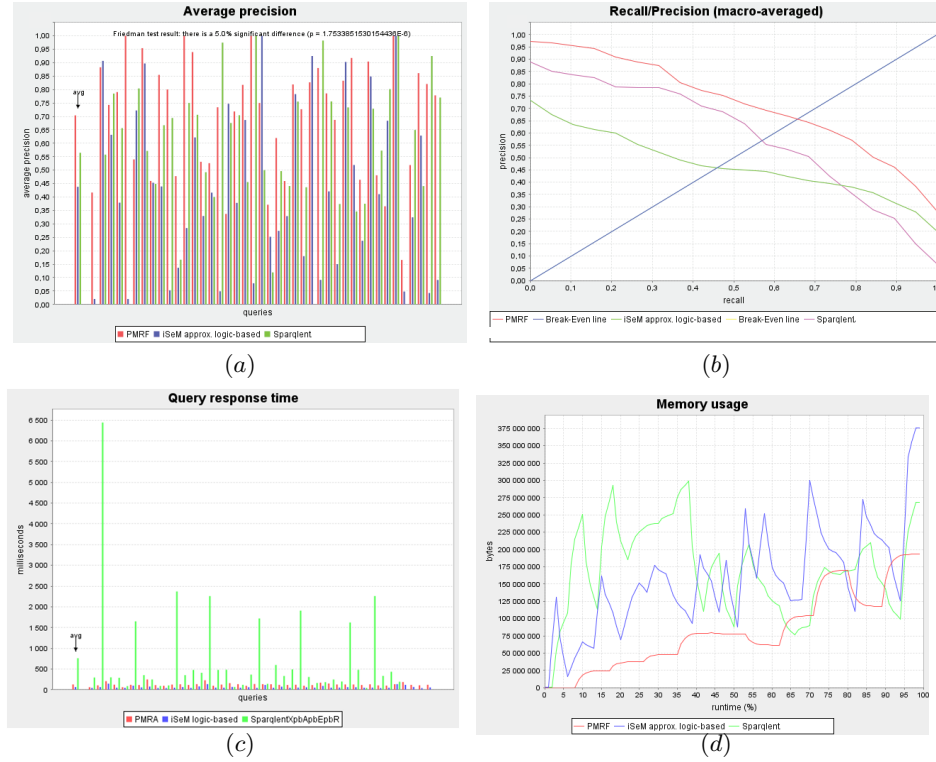


Fig. 8. Comparative study: (a) Average Precision, (b) Recall/Precision, (c) Query Response Time and (d) Memory Usage

The comparison of the Query Response Time of the PMRF, logic-based iSeM and SPARQLent is shown in Figure 8(c). The first column (Avg) gives the average response time for the three matchmakers. 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 is 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 8(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 [19]. 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][21][26][31][36][41]). In the future, we intend to enhance the framework to support QoS attributes for matching and ranking of Web services. The work of [7] 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. [9][18][30][31][43]). In the future, we intend to use a well-known and more advanced multicriteria method, namely the Dominance-based Rough Set Approach (DRSA) [15]. This method is particularly suitable for including the QoS attributes in the matching and ranking process. Furthermore, the DRSA can be seen as case-based reasoning method, which minimizes the cognitive effort required from user.

The last topic relates to the support of the imprecision and uncertainty in matching and ranking 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. K. Arnold, B. O’Sullivan, R.W. Scheifler, J. Waldo, and A. Woolrath. *The Jini Specification*. Addison-Wesley, Reading, MA, 1999.
3. 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.
4. U. Bellur, H. Vadodaria, and A. Gupta. Semantic matchmaking algorithms. In W. Bednorz, editor, *Advances in Greedy Algorithms*, pages 481–502. SInTech, Vienna, Austria, 2008.
5. S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient semantic service discovery in pervasive computing environments. In *ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 240–259, Melbourne, Australia, 27 November - 1 December 2006.
6. 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.
7. S. Chakhar, A. Ishizaka, and A.W. Labib. QoS-Aware parameterized semantic matchmaking framework for web service composition. In V. Monfort and K.-H. Krempels, editors, *The 10th international conference on web information systems and technologies (WEBIST 2014), volume 1, Barcelona, Spain, April 3-5, 2014*., volume 1, pages 50–61. SciTePress, 2014.
8. 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.
9. L. Cui, S. Kumara, and D. Lee. Scenario analysis of Web service composition based on multi-criteria mathematical goal programming. *Service Science*, 3(4):280–303, 2011.
10. P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized semantic match-making for workflow composition. IBM Research Report RC23133, IBM Research Division, March 2004.
11. P. Fu, S. Liu, H. Yang, and L. Gu. Matching algorithm of Web services based on semantic distance. In *International Workshop on Information Security and Application (IWISA 2009)*, pages 465–468, Qingdao, China, 21-22 November 2009.
12. 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.

13. 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.
14. M. Goncalves, M.-E. Vidal, A. Regalado, and N. Yacoubi Ayadi. Efficiently selecting the best web services. In *The Second International Workshop on Resource Discovery RED 2009, Lyon, France, August 28, 2009. Revised Papers*, volume 6162 of *Lecture Notes in Computer Science*, pages 120–139. Springer, 2010.
15. S. Greco, B. Matarazzo, and R. Slowiński. Rough sets theory for multicriteria decision analysis. *European Journal of Operational Research*, 129(1):1–47, 2001.
16. R. Guo, J. Le, and X.L. Xiao. Capability matching of Web services based on OWL-S. In *Sixteenth International Workshop on Database and Expert Systems Applications*, pages 653–657, 22–26 August 2005.
17. C.-L. Huang. A moderated fuzzy matchmaking for Web services. In *The Fifth International Conference on Computer and Information Technology (CIT 2005)*, pages 1116–1122, 2005.
18. B. Jeong, H. Cho, B. Kulvatunyou, and A. Jones. A multi-criteria Web services composition problem. In *IEEE International Conference on Information Reuse and Integration (IRI 2007)*, pages 379–384, 2007.
19. M. Klusch, M. Dudev, J. Misutka, P. Kapahnke, and M. Vasileski. *SME² Version 2.2. User Manual*. The German Research Center for Artificial Intelligence (DFKI), Germany, 2010.
20. 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.
21. R. Krithiga. QoS-Aware Web service selection using SOMA. *Global Journal of Computer Science and Technology*, 12(10):46–51, 2012.
22. J. Kuck and M. Gnasa. Context-sensitive service discovery meets information retrieval. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom'07)*, pages 601–605, March 2007.
23. H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
24. C. Lee, A. Helal, N. Desai, V. Verma, and B. Arslan. Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 33(6):682–696, 2003.
25. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *The 12th International Conference on World Wide Web, WWW '03*, pages 331–339, New York, NY, USA, 2003. ACM.
26. S.A. Ludwig. Memetic algorithm for Web service selection. In *The 3rd Workshop on Biologically Inspired Algorithms for Distributed Systems, BADS '11*, pages 1–8, New York, NY, USA, 2011. ACM.
27. Q. Lv, J. Zhou, and Q. Cao. Service matching mechanisms in pervasive computing environments. In *International Workshop on Intelligent Systems and Applications (ISA 2009)*, pages 1–4, May 2009.
28. Z. Maamar, S.K. Mostefaoui, and Q.H. Mahmoud. Context for personalized Web services. In *The 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, pages 166b–166b, Jan 2005.

29. U.S. Manikrao and T.V. Prabhakar. Dynamic selection of Web services with recommendation system. In *The International Conference on Next Generation Web Services Practices (NWeSP 2005)*, pages 117–121, August 2005.
30. D.A. Menascé. Composing Web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90, 2004.
31. D.A. Menascé and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 422–430, 2007.
32. B.A. Miller and R.A. Pascoe. Salutation service discovery in pervasive computing environments,. White paper, IBM Pervasive Computing, February 2000.
33. 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.
34. P.R. Reddy, A. Damodaram, and A.V.K. Prasad. Capability matching of Web services based on OWL-S. In S.C. Satapathy, P.S. Avadhani, and A. Abraham, editors, *Heterogeneous Matchmaking Approaches for Semantic Web Service Discovery Using OWL-S. International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012)*, volume 132 of *Advances in Intelligent and Soft Computing*, pages 605–612, Visakhapatnam, India, 2005. Springer Berlin Heidelberg.
35. W. Rong, K. Liu, and L. Liang. Personalized Web service ranking via user group combining association rule. In *The IEEE International Conference on Web Services (ICWS 2009)*, pages 445–452, July 2009.
36. M. Sathya, M. Swarnamugi, P. Dhavachelvan, and G. Sureshkumar. Evaluation of QoS based Web-service selection techniques for service composition. *International Journal of Software Engineering*, 1(5):73–90, 2011.
37. 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.
38. K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The retina mas infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.
39. Y. Syu, S.-P. Ma, J.-Y. Kuo, and Y.-Y. FanJiang. A survey on automated service composition methods and related techniques. In *The IEEE Ninth International Conference on Services Computing (SCC 2012)*, pages 290–297, Hawaii,USA, 24–29 June 2012.
40. R. Wang, C.-H. Chi, and J. Deng. A fast heuristic algorithm for the composite Web service selection. In *The Joint International Conference on Advances in Data and Web Management, APWeb/WAIM '09*, pages 506–518, Berlin, Heidelberg, 2009. Springer-Verlag.
41. Y. Xia, P. Chen, L. Bao, M. Wang, and J. Yang. A QoS-Aware Web service selection algorithm based on clustering. In *IEEE International Conference on Web Services (ICWS)*, pages 428–435, 2011.
42. T. Yu and K.-J. Lin. Service selection algorithms for Web services with end-to-end QoS constraints. In *The IEEE International Conference on e-Commerce Technology (CEC 2004)*, pages 129–136, July 2004.
43. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality driven Web services composition. In *The 12th International Conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.