PMRF: Parameterized Matching-Ranking Framework

Fatma Ezzahra Gmati, Nadia Yacoubi-Ayadi, Afef Bahri, Salem Chakhar, and Alessio Ishizaka

Abstract The PMRF (Parameterized Matching-Ranking Framework) is a highly configurable framework supporting a parameterized matching and ranking of Web services. This paper first introduces the matching and ranking algorithms supported by the PMRF. Next, it presents the architecture of the developed system and discusses some implementation issues. Then, it provides the results of performance evaluation of the PMRF. It also compares PMRF to two exiting frameworks, namely iSeM-logic-based and SPARQLent. The different matching and ranking algorithms have been evaluated using the OWLS-TC4 datasets. The evaluation has been conducted employing the SME2 (Semantic Matchmaker Evaluation Environment) tool. The results show that the algorithms behave globally well in comparison to iSeM-logic-based and SPARQLent.

Fatma Ezzahra Gmati
RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, 2010 Manouba, Tunisia. e-mail: fatma.ezzahra.gmati@gmail.com

Nadia Yacoubi-Ayadi
RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, 2010 Manouba, Tunisia. e-mail: nadia.yacoubi.ayadi@gmail.com

Afef Bahri
MIRACL Laboratory, Higher School of Computing and Multimedia, Technopole Sfax, 3021 Sfax, Tunisia e-mail: afef.bahri@gmail.com

Salem Chakhar
Portsmouth Business School and Centre for Operational Research and Logistics, University of Portsmouth, Portland Street, Portsmouth PO1 3AH, UK. e-mail: salem.chakhar@port.ac.uk

Alessio Ishizaka
Portsmouth Business School and Centre for Operational Research and Logistics, University of Portsmouth, Portland Street, Portsmouth PO1 3AH, UK. e-mail: alessio.ishizaka@port.ac.uk

1 Introduction

The matchmaking is a crucial operation in Web service composition. The objective of the matchmaking is to discover and select the most appropriate (i.e., that responds better to the user request) Web service among the different available candidates. Several matchmaking frameworks are now available in the literature, e.g., [1][14][16][17][19][20][23][24][26][28][27]. 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 low precision of the retrieved services;
2. use of capability-based matchmaking, which is proven [6] to be inadequate in practice;
3. lack of customization and configurability support for both the user and the provider;
4. lack of accurate ranking of matching Web services, especially within semantic-based matching.

Several conceptual and algorithmic solutions to jointly deal with the previous shortcomings are under investigation in an ongoing research project. The first results are given in [9]. The objective of this paper is to present the developed prototype, PMRF (Parameterized Matching-Ranking Framework), supporting the different proposed matching and ranking algorithms. The paper first introduces the matching and ranking algorithms supported by the PMRF. Then, it presents the architecture of the developed system and discusses some implementation issues. Finally, it provides the results of performance evaluation of the PMRF and also compares it to two well-known matchmakers, namely iSeM-logic-based [12] and SPARQLent [21][22].

To evaluate the performance of PMRF, we used seven different configurations with different versions of matching and ranking algorithms. All the algorithms have been evaluated using the OWLS-TC4 datasets. The evaluation has been conducted employing the SME2 (Semantic Matchmaker Evaluation Environment) tool [11]. The results show that the algorithms behave globally well in comparison to iSeM-logic-based and SPARQLent.

The rest of the paper is organized as follows. Section 2 reviews the matching and ranking algorithms. Section 3 presents the architecture of the PMRF. Section 4 studies the performance of the PMRF. Section 5 compares the PMRF to other similar frameworks. Section 6 comments on the users/providers acceptability. Section 7 discusses some related work. Section 8 concludes the paper.

2 Matching and Ranking Algorithms

In this section, we briefly review the matching and ranking algorithms supported by the PMRF.

2.1 Matching Algorithms

The PMRF supports three matching algorithms: trivial, partially parameterized and fully parameterized. These algorithms support different levels of customization. The trivial 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: (i) A first customization consists in allowing the user to specify the list of attributes to consider; (ii) A second customization consists in allowing the user to specify the order in which the attributes are considered; and (iii) 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.

In order to support all the above-cited customizations, we used the concept of Criteria Table, introduced by [6], that serves as a parameter to the matching process. A Criteria Table, C, is a relation consisting of two attributes, $C.A$ and $C.M$. $C.A$ 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 ith tuple of the relation. $C.N$ denotes the total number of tuples in C.

Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be 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 as specified in $C.M$. Formally,

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

According to this definition, only the attributes specified by the user in the Criteria Table are considered during the matching process.

The fully parameterized matching algorithm is formalized in Algorithm 1. This algorithm follows directly from Sentence (1). Algorithm 1 proceeds as follows: (i) Loops over the Criteria Table 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 (for requested Web service S^A) and aAttrSet (for advisable Web service S^A). This operation is implemented by sentences 1 to 10 in Algorithm 1; and (ii) 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 the Criteria Table C there are similar attribute in the advertised service S^A with a sufficient similarity degree) or fail (if the similarity for at least one attribute in the Criteria Table C fails).

Let us now focus on the complexity of Algorithm 1. Generally, we have $S^A.N \gg S^R.N$, hence the complexity of the first outer *while* loop is $O(C.N \times S^A.N)$. Then, the worst case complexity of Algorithm 1 is $O(C.N \times S^A.N) + \alpha$ where α is the

complexity of computing μ. The value of α depends on the approach used to infer $\mu(\cdot,\cdot)$. As underlined in [6], inferring $\mu(\cdot,\cdot)$ by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines, which use the fast Rete [7] pattern-matching algorithm leads to $\alpha = O(|R||F||P|)$ where $|R|$ is the number of rules, $|F|$ is the number of facts, and $|P|$ is the average number of patterns in each rule. In this case, the worst case complexity of Algorithm 1 is $O(C.N \times S^A.N) + O(|R||F||P|)$. Furthermore, we observe, as in [6], that the process of computing $\mu(\cdot,\cdot)$ is the most 'expensive' step of Algorithm 1. Hence, we obtain: $O(C.N \times S^A.N) + O(|R||F||P|) \asymp O(|R||F||P|)$.

Algorithm 1: Fully Parameterized Matching

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
         C, // Criteria Table.
Output: Boolean, // fail/success.
1  while (i ≤ C.N) do
2      while (j ≤ S^R.N) do
3          if (S^R.A_j = C.A_i) then
4              Append S^R.A_j to rAttrSet;
5          j ⟵ j + 1;
6      while (k ≤ S^A.N) do
7          if (S^A.A_k = C.A_i) then
8              Append S^A.A_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;
```

Different versions and extensions of this algorithm are available in [4][5][9]. We remark that Algorithm 1 permits to compute the similarly between a requested Web service S^R and an advertised Web service S^A. In practice, however, matching process should consider all the Web services available in the registry. An extended version of Algorithm 1 that takes into account this fact is given in [9].

2.2 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 is based on the use of a tree data structure. It permits to solve the problem of ties of the first algorithm. In addition, it is computationally better than the rule-based ranking

algorithm. In the present paper, we present the score-based ranking algorithm. We note that the rule-based ranking algorithm is available in [9] while the tree-based algorithm is given in [8].

The score-based ranking approach is implemented by Algorithm 2. The main input of this algorithm is a list mServices of matching Web services. The function ComputeNormalizedScores in Algorithm 2 permits to calculate the scores of Web services. It implements the idea we proposed in [9]. The score-based ranking algorithm uses then a *merge sort procedure* (implemented by lines 3-11 in Algorithm 2) to rank the Web services based on their normalized scores.

The list mServices used as input to Algorithm 2 has the following generic definition:

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

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes and $\mu(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j.

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

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

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

Algorithm 2: Score-Based Ranking

```
   Input  : mServices, // List of matching Web services.
            N, // Number of attributes.
   Output: mServices, // Ranked list of Web services.
1  mServices ← ComputeNormalizedScores(mServices,N);
2  r ← length(mServices);
3  while (i ≤ r) do
4      Let row_i be the ith row in mServices;
5      while (j ≤ r) do
6          Let row_j be the jth row in mServices;
7          if (mServices[i,N+2] > mServices[j,N+2]) then
8              tmp ⟵ row_j;
9              row_j ⟵ row_i;
10             row_i ⟵ tmp;
11             update mServices;
12 return mServices;
```

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 of mServices is as follows:

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

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

The second version of mServices is as follows:

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

where S_i^A, S^R and N are as defined above; and $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j. In this case, the similarity measure is 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. This will deteriorate the precision of the ranking algorithm. The tree-based ranking algorithm permits to completely solve the ties problem.

The function ComputeNormalizedScores in Algorithm 2 has a complexity of $O(r(2+N^2))$ where r is the number of Web services and N is the number of attributes. The length in line 2 is assumed to be a built-in function and its complexity is not considered here. The sentences in lines 3-11 in Algorithm 2 implement a merge sort procedure, which at best has a time complexity of $O(r \log r)$ and in worst case, it makes $O(r^2)$. Hence, the overall complexity of Algorithm 2 in best case is $O(r(2+N^2)) + O(r \log r)$ and in worst case is $O(r(2+N^2)) + O(r^2)$.

3 System Architecture and Implementation

In this section, we first present the conceptual and functional architectures of the PMRF. Then, we discuss some implementation issues.

3.1 System Design and Conceptual Architecture

Figure 1 provides the conceptual architecture of the PMRF. The inputs of the system are: the Criteria Table/List, the published Web services repository, the user request and its corresponding Ontologies. The other parameters (namely, the similarity degrees weights and the order functions; see [9]) are computed by the PMRF. The output of the PMRF 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 Criteria Table/List. The result is then passed to the Ranking Module.

This module produces a ranked list of Web services. The assembler guarantees a coherent interaction between the different modules in the second layer.

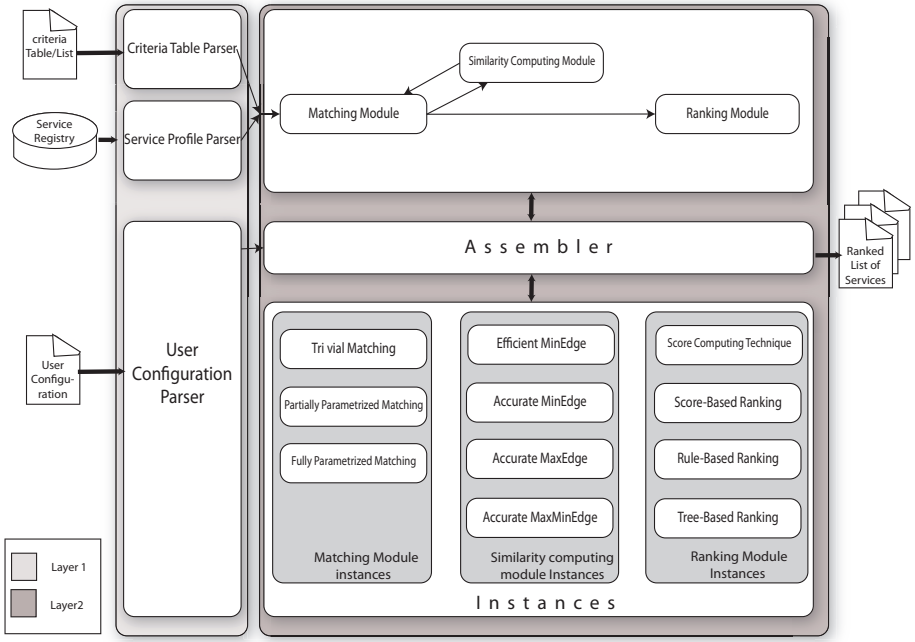


Fig. 1 Conceptual architecture of the PMRF

The three main components of the second layer of the PMRF are:

- **Matching Module**: This component contains the different matching algorithms:
  - Trivial matching algorithm,
  - Partially parameterized matching algorithm,
  - Fully parameterized matching algorithm.
- **Similarity Computing Module**: This component supports the different similarity measure computing approaches:
  - Efficient similarity with MinEdge,
  - Accurate similarity with MinEdg,
  - Accurate similarity with MaxEdge,
  - Accurate similarity with MaxMinEdge.
- **Ranking Module**: This component is the repository of the score computing technique and the different ranking algorithms. It contains the following elements:
  - Score computing technique,

– Score-based ranking algorithm,
– Rule-based ranking algorithm,
– Tree-based ranking algorithm.

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 (including the specifications of the requested Web service and the different parameters) until the delivery of the final results (ranked list of Web services matching the query) 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;
- 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 delivers (7) a ranked list of Web services;
- The PMRF delivers (8) the ranked list of Web services to the user.

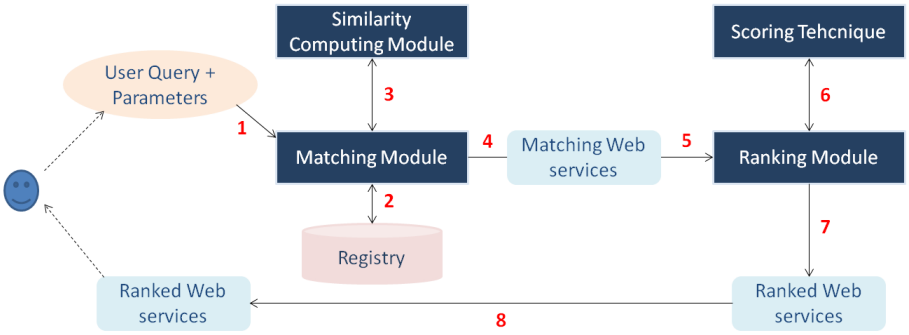


Fig. 2 Functional architecture of the PMRF

3.3 Implementation

To develop the PMRF, we have used the following tools:

- Eclipse IDE (http://eclipse.org/ide/) as the developing platform;
- OWLS-API (http://on.cs.unibas.ch/owls-api/) to parse the OWLS service descriptions;
- OWL-API (http://owlapi.sourceforge.net/) along with the Pellet reasoner (http://clarkparsia.com/pellet/) to perform the inference for computing the similarity degrees.

The inference is one of the main issues encountered during the developing of the PMRF. We perform the following procedure in order to minimize resources consumption, especially memory:

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.

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.

4 Performance Evaluation

In this section, we provide the performance evaluation results.

4.1 Evaluation Framework

To evaluate the performance of the PMRF, we used the SME2 [11], 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

```
public class Matchmaker implements IMatchmakerPlugin {

        PelletReasoner reasoner=new PelletReasoner();
        ServiceTuple query;
        ArrayList<ServiceTuple> offers=new ArrayList<ServiceTuple>();


        public Matchmaker()
        {}

        @Override
        public void input(URL arg0) {
            try {
                    ServiceTuple service=new ServiceTuple(arg0,reasoner);
                    offers.add(service);
                    System.out.println("helloooo");
            } catch (Exception e) {
                    e.printStackTrace();
            }
        }
        @Override
        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
             * ************************************************************************
             */
            Group initialGroup=new Group();
            for(ServiceTuple serviceAd:offers)
            {
                match(query,serviceAd,reasoner);
                initialGroup.addAService(serviceAd);
            }
            /*
             * ************************************************************************
             * We secondly perform the ranking
             * ************************************************************************
             */
            Node<Group>  root= new Node<Group>();
            root.setData(initialGroup);
```

...

Fig. 3 Extract from the Class Matchmaker

paper are: precision and recall, average precision, query response time and memory consumption. The definition of these metrics are given in [11][13].

A series of 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 used is OWLS-TC4, which consists of 1083 Web service offers described in OWL-S 1.1 and 42 queries.

4.2 Performance Evaluation Analysis

In order to study the performance of each instance of the modules supported by the PMRF and describe the difference between them, we implemented seven plugins to be used with the SME2 tool. Each of these plugins represents a different combination of the matching, similarity computing and ranking algorithms. The characteristics of these plugins are summarized in Table 1.

Table 1 Description of the evaluated configurations

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

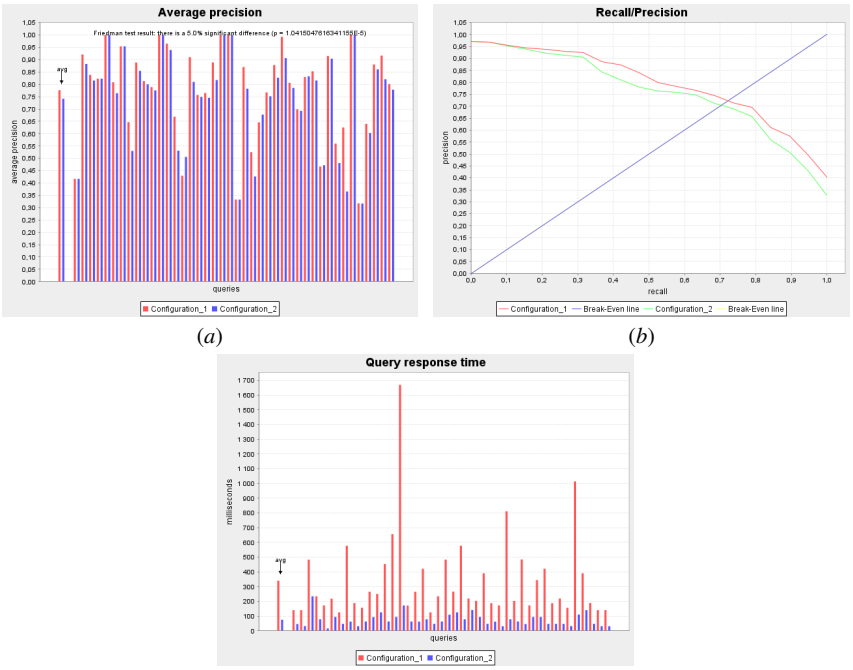
4.2.1 Comparison of Configurations 1 and 2

The evaluation of configurations 1 and 2 yields to the results shown in Figure 4. The difference between the two configurations is the similarity measure module instance. Indeed, the first configuration employs the **Accurate MinEdge** instance while the second employs the **Efficient MinEdge** instance. Figure 4.*a* shows the Average Precision and figure 4.*b* illustrates the Recall/Precision plot. 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 4.*c*, however, configuration 2 is shown to be remarkably faster than configuration 1. This is due to the inference process (which is used in configuration 1) that consumes considerable resources.

4.2.2 Comparison of Configurations 1 and 4

The results of comparison of configuration 1 and 4 are shown in Figure 5. The difference between these two configurations is the matching module instance. The first configuration is based on the trivial matching algorithm while the second uses the fully parameterized matching. Figure 5.*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 high Average Precision value. Figure 5.*b* illustrates the Recall/Precision plot. It shows that configuration 4 has a

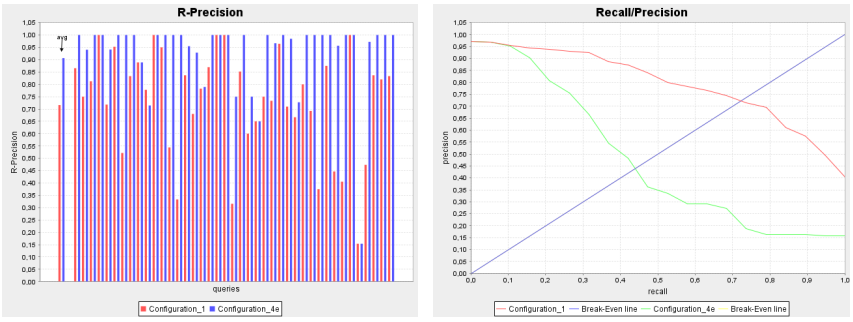
low recall rate. The overly restrictive Criteria Table explains these results, since it fails to return some relevant services.



(*a*) (*b*)

(*c*)

Fig. 4 Configuration 1 vs Configuration 2: (*a*) Average Precision, (*b*) Recall/Precision, (*c*) Query Response Time

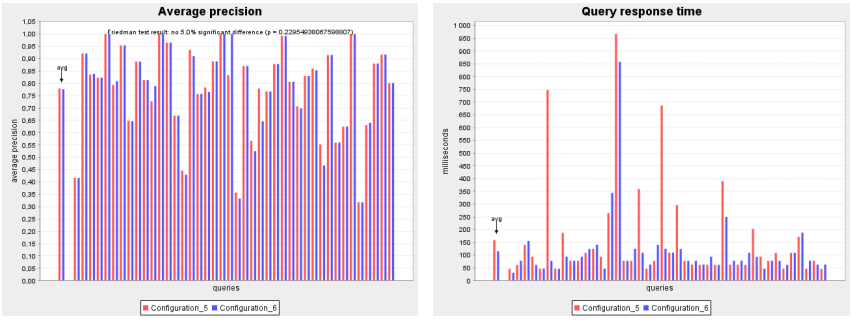


(*a*) (*b*)

Fig. 5 Configuration 1 vs Configuration 4: (*a*) Average Precision, (*b*) Recall/Precision

4.2.3 Comparison of Configurations 5 and 6

Figure 6 show the evaluation results of configurations 5 and 6. The difference between these two configurations is the ranking module instance. The first uses the tree-based ranking algorithm while the second employs the rule-based ranking algorithm. Figure 6.*a* shows that configuration 5 has a slightly better Average Precision than configuration 6. Figure 6.*b* shows that configuration 6 is obviously faster than configuration 5.



(*a*) (*b*)

Fig. 6 Configuration 5 vs Configuration 6: (*a*) Average Precision, (*b*) Query Response Time

5 Comparative Study

We compared the results of the PMRF matchmaker with SPARQLent [21][22] and iSeM [12] frameworks. Configuration 7 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.

5.1 Average Precision

The Average Precision is shown in Figure 7.*a*. This figure shows that PMRF has a more accurate Average Precision than iSeM logic-based and SPARQLent. It is possible to conclude that PMRF has better ranking precision than the two other ap-

proaches. In addition, the ranking generated 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 [18], which gives equal weights to all similarity degrees.

5.2 Recall/Precision

Figure 7.*b* presents the Recall/Precision of 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).

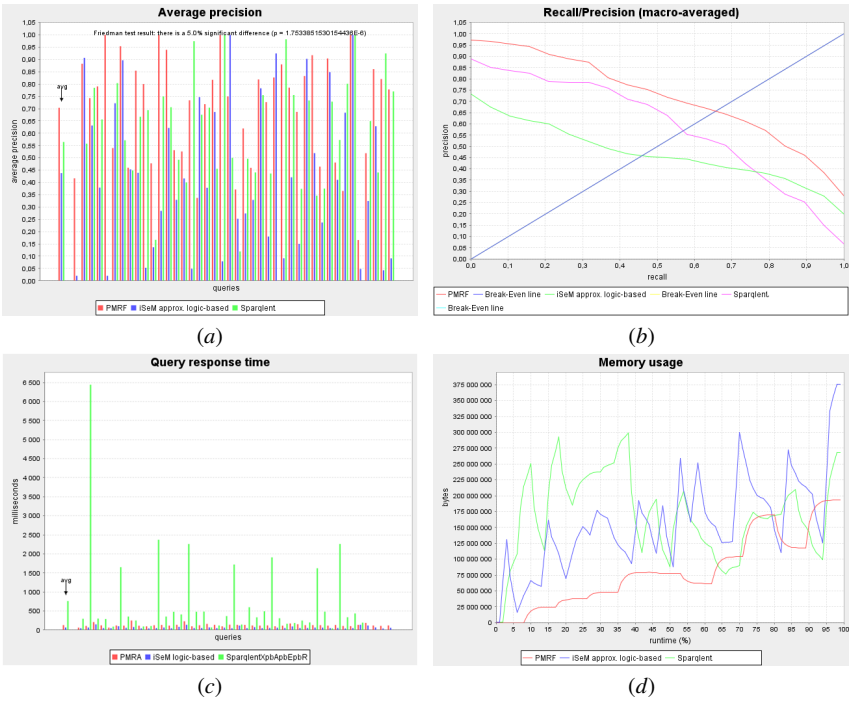


Fig. 7 Comparative Study: (*a*) Average Precision, (*b*) Recall/Precision, (*c*) Query Response Time, (*d*) Memory Usage

5.3 Query Response Time

Figure 7.*c* compares the Query Response Time of the PMRF, logic-based iSeM and SPARLent. The first column (Avg) gives the average response time for the three matchmakers. The experimental results show that the PMRF is faster than SPARQ-Lent (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.

5.4 Memory Usage

Figure 7.*d* shows the Memory Usage for PMRF, iSeM logic-based and SPARQ-Lent. 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.

6 Discussion

An 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 users/providers acceptability and their ability to specify the required parameters, especially the Criteria Table. Indeed, the specification of these parameters may require an important cognitive effort from the user/providers. 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 configurations. Second, a global time limit to the composition process can be used to orient the system towards the use of the accurate version or efficient version of the similarity measure computing algorithm. 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 solutions cited above.

7 Related Work

Several existing frameworks have influenced this research project, especially the proposals of [12][2][18][6][4][5]. Table 2 provides the characteristics of some existing frameworks. Ludwig [16] proposes two matchmaking approaches: one that is based on a genetic algorithm, and the other is based on a memetic algorithm to match consumers with services based on Quality of Service (QoS) attributes. Wang et *al*. [27] propose the use of utility function to evaluate each component service based on the definition given in [28] and then map the multi-dimensional QoS composite Web service to the multi-dimensional multi-choice knapsack. Finally, they use an heuristic algorithm for solving the problem.

Some proposals including [3][10] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only. The proposal of [4][5] lack effective implementation of the proposed matchmaking framework. Indeed, the authors discuss very generally and very briefly the technical issues. In addition, the authors do not precise how the similarity degree is computed and how the different matching Web services are ranked. Finally, there is a lack of effective evaluation and performance analysis of matching algorithms.

Although that these proposals are based on semantics, they fail to take into account jointly the shortcomings of Web services matchmaking enumerated in the introduction. Indeed, the proposal of [2][12][22] do not support any customization while those of [4][5][6] do not propose solutions for ranking Web services.

Table 2 Comparison of Matchmaking Frameworks

Matchmaker	Matching Type	Attributes	Customization	Ranking	Description Language
Jini [1]	Syntactic	Capability	No	No	No
Konark [14]	Syntactic	Capability	No	No	XML
Salutation [17]	Logic-based	Capability	No	Yes	OWL-S
MatchMaker [25]	Syntactic	Capability	No	No	DAMS/UDDI
RACER [15]	Syntactic	Capability	No	No	DAML-S
PSMF [6]	Logic-based	Capability	Yes	No	DAML-S/WSDL/ UDDI
SPARQLent [22]	Logic-based	Capability	No	Yes	OWL-S
iSeM-logic-based [12]	Logic-based	Capability	No	Yes	OWL-S/SAWSDL
QoSeBroker [5]	Logic-based	Capability/QoS/ Property	Yes	No	OWL-S
PMRF	Logic-based	Capability/ Property	Yes	Yes	OWL-S

8 Conclusion

In this paper, we presented a highly customizable framework, called PMRF, for matching and ranking Web services. We briefly reviewed the matching and ranking algorithms supported by the PMRF, provided its conceptual and functional architecture and discussed some implementation issues. We also presented the results of the

performance evaluation of the PMRF using the OWLS-TC4 datasets. The evaluation has been conducted using the SME2 tool [11]. We finally compared PMRF to two exiting frameworks, namely iSeM-logic-based [12] and SPARQLent [21][22]. The results show that the algorithms supported by PMRF behave globally well in comparison to iSeM-logic-based and SPARQLent frameworks.

In the future, we intend to enhance PMRF by (i) including other matching techniques; namely textual matching and Ontology distance calculation; (ii) adapt it to Ontology evolvement in a dynamic Web service environment; iii) make the PMRF useable over the cloud technology; and ((iv) use Artificial Intelligence techniques to reduce the cognitive effort required from the users/providers.

References

1. Arnold, K., O'Sullivan, B., Scheifler, R., Waldo, J., Woolrath, A.: The Jini Specification. Addison-Wesley, Reading, MA (1999)
2. Bellur, U., Kulkarni, R.: Improved matchmaking algorithm for semantic Web services based on bipartite graph matching. In: IEEE International Conference on Web Services, pp. 86–93. Salt Lake City, Utah, USA (2007)
3. Ben Mokhtar, S., Kaul, A., Georgantas, N., Issarny, V.: Efficient semantic service discovery in pervasive computing environments. In: ACM/IFIP/USENIX 2006 International Conference on Middleware, pp. 240–259. Melbourne, Australia (2006)
4. Chakhar, S.: 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), pp. 159–165. Seville, Spain (2013)
5. Chakhar, S., Ishizaka, A., Labib, A.: QoS-aware parameterized semantic matchmaking framework for Web service composition. In: V. Monfort, K.H. Krempels (eds.) WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 1, Barcelona, Spain, 3-5 April, 2014, pp. 50–61. SciTePress (2014)
6. Doshi, P., Goodwin, R., Akkiraju, R., Roeder, S.: Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division (2004)
7. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artificial Intelligence **19**(1), 17–37 (1982)
8. Gmati, F.E., Yacoubi-Ayadi, N., Bahri, A., Chakhar, S., Ishizaka, A.: A tree-based algorithm for ranking Web services. In: V. Monfort, K.H. Krempels (eds.) WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015. SciTePress (2015)
9. Gmati, F.E., Yacoubi-Ayadi, N., Chakhar, S.: Parameterized algorithms for matching and ranking Web services. In: Proceedings of the On the Move to Meaningful Internet Systems: OTM 2014 Conferences 2014, *Lecture Notes in Computer Science*, vol. 8841, pp. 784–791. Springer (2014)
10. Guo, R., Le, J., Xiao, X.: Capability matching of Web services based on OWL-S. In: Sixteenth International Workshop on Database and Expert Systems Applications, pp. 653–657 (2005)
11. Klusch, M., Dudev, M., Misutka, J., Kapahnke, P., Vasileski, M.: SME^2 Version 2.2. User Manual. The German Research Center for Artificial Intelligence (DFKI), Germany (2010)
12. Klusch, M., Kapahnke, P.: 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)
13. Küster, U., König-Ries, B.: Measures for benchmarking semantic Web service matchmaking correctness. In: Proceedings of the 7th International Conference on The Semantic Web:

Research and Applications - Volume Part II, ESWC'10, pp. 45–59. Springer-Verlag, Berlin, Heidelberg (2010)
14. Lee, C., Helal, A., Desai, N., Verma, V., Arslan, B.: 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)
15. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proceedings of the 12th International Conference on World Wide Web, WWW '03, pp. 331–339. ACM, New York, NY, USA (2003)
16. Ludwig, S.: Memetic algorithm for Web service selection. In: Proceedings of the 3rd Workshop on Biologically Inspired Algorithms for Distributed Systems, BADS '11, pp. 1–8. ACM, New York, NY, USA (2011)
17. Miller, B., Pascoe, R.: Salutation service discovery in pervasive computing environments,. White paper, IBM Pervasive Computing (2000)
18. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proceedings of the First International Semantic Web Conference on The Semantic Web, ISWC '02, pp. 333–347. Springer-Verlag, London, UK, UK (2002)
19. Rodriguez-Mier, P., Pedrinaci, C., Lama, M., Mucientes, M.: An integrated semantic Web service discovery and composition framework. IEEE Transactions on Services Computing (2015). Forthcoming
20. Samper Zapater, J., Llido Escriva, D., Soriano Garcia, F., Martinez Dura, J.: Semantic Web service discovery system for road traffic information services. Expert Systems with Applications **42**(8), 3833–3842 (2015)
21. Sbodio, M.: SPARQLent: A SPARQL based intelligent agent performing service matchmaking. In: B. Blake, L. Cabral, B. König-Ries, U. Küster, D. Martin (eds.) Semantic Web Services, pp. 83–105. Springer Berlin Heidelberg (2012)
22. Sbodio, M., Martin, D., Moulin, C.: 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)
23. Sharma, S., Lather, J., Dave, M.: Google based hybrid approach for discovering services. In: IEEE International Conference on Semantic Computing (ICSC 2015), pp. 498–502. IEEE, Anaheim, CA (2015)
24. Srujana, S., Raju, V., Kiran, M.: Semantic Web services discovery using logic based method. In: Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014 - Volume 1, Bhubaneswar, Odisa, India, 14-15 November 2014, pp. 623–629 (2014)
25. Sycara, K., Paolucci, M., van Velsen, M., Giampapa, J.: The retsina mas infrastructure. Autonomous Agents and Multi-Agent Systems **7**(1-2), 29–48 (2003)
26. Syu, Y., Ma, S.P., Kuo, J.Y., FanJiang, Y.Y.: A survey on automated service composition methods and related techniques. In: The IEEE Ninth International Conference on Services Computing (SCC 2012), pp. 290–297. Hawaii,USA (2012)
27. Wang, R., Chi, C.H., Deng, J.: A fast heuristic algorithm for the composite Web service selection. In: Proceedings of the Joint International Conferences on Advances in Data and Web Management, APWeb/WAIM '09, pp. 506–518. Springer-Verlag, Berlin, Heidelberg (2009)
28. Yu, T., Lin, K.J.: Service selection algorithms for Web services with end-to-end qos constraints. In: Proceedings of the IEEE International Conference on e-Commerce Technology (CEC 2004), pp. 129–136 (2004)