Semantic Matching-Based Selection and QoS-Aware Classification of Web Services

Salem Chakhar, Alessio Ishizaka, and Ashraf Labib

Portsmouth Business School, University of Portsmouth, Portsmouth PO1 3DE, UK
{Salem.Chakhar,Alessio.Ishizaka,Ashraf.Labib}@port.ac.uk

Abstract. This paper focuses on Web services matchmaking. It distinguishes three types of matching: functional attribute-level, functional service-level, and non-functional. In this paper, a series of parameterized and highly customizable algorithms are advertised for the different types of matching. A prototype has been developed and used to test the functional attribute-based conjunctive matching using the SME2 environment and the OWLS-TC4 datasets. Results show that the algorithms behave globally well in comparison to similar existing ones.

Keywords: Web service, Service composition, Matchmaking, Quality of Service, Similarity Measure.

1 Introduction

An important issue within web service composition is related to the selection of the most appropriate one among the different candidate web services. In this paper, we propose a semantic matchmaking framework for web service composition. Three types of matching are distinguished in this paper: functional attribute-level, functional service-level, and non-functional. In [5] we discussed functional attribute-level and functional service-level matching. This paper enhances our proposal in [5] by adding generic functional attribute-level and non-functional matching. We also briefly describe the developed prototype and compares the attribute-based conjunctive matching to the ones included in the SEM and SPARQLent frameworks. We used the Semantic Matchmaker Evaluation Environment (SME2) [13] and the OWLS-TC4 datasets to evaluate the performances of the algorithms in respect to several parameters. Results show that our algorithms behaves globally well in comparison to iSEM and SPARQLent.

The paper is structured as follows. Section 2 sets the background. Sections 3, 4 and 5 present different matching algorithms. Section 6 presents performance analysis. Section 7 discusses related work. Section 8 concludes the paper.

2 Background

2.1 Basic Definitions

The following are some basic definitions of a service and other service-specific concepts. Several definitions are due to [7].

Definition 1. *A service S is defined as a collection of attributes that describe the service. Let $S.A$ denotes the set of attributes of service S and $S.A_i$ denotes each member of this set. Let $S.N$ denotes the cardinality of this set.*

Definition 2. *The capability of a service $S.C$ is a subset of service attributes ($S.C \subseteq S.A$), and includes only functional ones that directly relate to its working.*

Definition 3. *The quality of a service $S.Q$, is a subset of service attributes ($S.Q \subseteq S.A$), and includes all attributes that relate to its QoS.*

Definition 4. *The property of a service, $S.P$, is a subset of service attributes ($S.P \subseteq S.A$), and includes all attributes other than those included in service capability or service quality.*

2.2 Service Matching Types and Process

The input for a Web service composition is a set of specifications describing the capabilities of the desired service. These specifications can be decomposed into two groups [4][5]: (i) functional requirements that deal with the desired functionality of the composite service, and (ii) non-functional requirements that relate to the issues like cost, performance and availability. These specifications need to be expressed in an appropriate language. In this paper, we adopt an extended version of Ontology Web Language (OWL) [1] for expressing functional requirements and the Quality of Service (QoS) for non-functional requirements.

In [5], we distinguished three types of service matching: (i) *functional attribute-level matching* that implies capability and property attributes and consider each matching attribute independently of the others; (ii) *functional service-level matching* that considers capability and property attributes but the matching operation implies attributes both independently and jointly; and (iii) *non-functional matching* which focuses on the attributes related to the QoS.

The functional matching takes as input all candidate Web services and produces a set of Web services that meet the user functional matching criteria. Hence, service types that fail to meet the user functional requirements are automatically eliminated. The non-functional matching takes as input a set of Web service instances that meet the functional requirements and classify them into different predefined and ordered quality of service classes.

2.3 Similarity Measure

A semantic match between two entities frequently involves a similarity measure that quantifies the semantic distance between the two entities participating in the match. As in [7], a similarity measure is defined as follows.

Definition 5. *The similarity measure, μ, of two service attributes is a mapping that measures the semantic distance between the conceptual annotations associated with the service attributes. Mathematically,*

$$\mu : A \times A \rightarrow \{Exact,\ Plug\text{-}in,\ Subsumption,\ Container,\ Part\text{-}of,\ Fail\}$$

where A is the set of all possible attributes.

The mapping between two conceptual annotations is called:

- **Exact** map: if the two conceptual annotations are syntactically identical,
- **Plug-in** map: if the first conceptual annotation is specialized by the second,
- **Subsumption** map: if the first conceptual annotation specializes the second,
- **Container** map: if the first conceptual annotation contains the second,
- **Part-of** map: if the first conceptual annotation is a part of the second, and
- **Fail** map: if none of the previous cases applies.

A preferential total order is established on the above mentioned similarity maps.

Definition 6. *Preference amongst similarity measures is governed by the following strict total order:*

$$Exact \succ Plug\text{-}in \succ Subsumption \succ Container \succ Part\text{-}of \succ Fail$$

where $a \succ b$ means that a is preferred over b.

To compute the similarity degrees, we implemented the idea proposed by [2] which starts by constructing a bipartite graph where the vertices in the left side correspond to the concepts associated with advertised services, while those in the right side correspond to the concepts associated with the requested service. The edges correspond to the semantic relationships between concepts. Then, the authors in [2] assign a weight to each edge and then apply the Hungarian algorithm [16] to identify the complete matching that minimizes the maximum weight in the graph. The final returned degree is the one corresponding to the maximum weight in the graph.

3 Functional Attribute-Level Matching

Functional matching is the process of discovering a service advertisement that *sufficiently* satisfies a service request [7]. It is based on the concept of *sufficiency*, which itself is based on the *similarity* measure defined in the previous section.

3.1 Conjunctive/Disjunctive Matching

Let S^R be the service that is requested, and S^A be the service that is advertised. A first customization of functional matching is to allow the user to specify a desired similarity measure for each attribute. A sufficient match exists between S^R and S^A in respect to a given attribute if there exists an identical attribute of S^A and the values of the attributes satisfy the desired similarity measure. A second customization of the matching process is to allow the user specifying which attributes should be utilized during the matching process, and the order in which the attributes must be considered for comparison. In order to support both customizations, we use the concept of Criteria Table, introduced by [7], that serves as a parameter to the matching process.

Definition 7. *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.*

Example 1. Table 1 shows a Criteria Table example.

Table 1. An example Criteria Table

$C.A$	$C.M$
input	Exact
output	Exact
service category	Subsumes

A sufficient functional attribute-level conjunctive match between services is defined as follows.

Definition 8. *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 conjunctive 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,*

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

The algorithm for functional attribute-level conjunctive matching is provided in [5]. A less restrictive definition of sufficiency consists in using a disjunctive rule on the individual matching measures.

Definition 9. *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 disjunctive match exists between S^R and S^A if for at least one attribute in $C.A$ it 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,*

$$\begin{aligned} &\exists_{i,j,k} (C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \\ &\Rightarrow \mathit{SuffFuncDisjMatch}(S^R, S^A). \end{aligned} \tag{2}$$

The algorithm for functional attribute-level disjunctive matching is given in [5].

3.2 Generic Matching

In this section we extend the algorithms proposed in [5] to generic binary connectors by allowing the user to specify the conditional relationships between the capability and property attributes. First, we need to introduce the concept of sufficient single attribute match.

Definition 10. *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 in respect to attribute $S^R.A_i$ if there exists an identical attribute of S^A and the values of the attributes satisfy the desired similarity measure as specified in $C.M_i$. Formally,*

$$\begin{aligned} &\exists_{j,k}(C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i) \\ &\Rightarrow \mathit{SuffSingleAttrMatch}(S^R, S^A, A_i). \end{aligned} \tag{3}$$

The single attribute matching is formalized in Algorithm 1 that follows directly from Sentence (3).

Algorithm 1: SuffSingleAttrMatching

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
         C, // Criteria Table.
         i, // Service attribute index.
Output: Boolean// success/fail.
while (j ≤ S^R.N) do
    if (S^R.A_j = C.A_i) then
        Append S^R.A_j to rAttrSet;
    Assign j ⟵ j + 1; ;
while (k ≤ S^A.N) do
    if (S^A.A_k = C.A_i) then
        Append S^A.A_k to aAttrSet;
    Assign k ⟵ k + 1;
if (μ(rAttrSet[i], aAttrSet[i]) ≽ C.M_i) then
    return success;
return fail;
```

Let now define the sufficient functional generic match.

Definition 11. *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be the criteria table. Let T be a complex logical clause where operands are the attributes related by logical operators (e.g. or, and, not). A sufficient functional generic match between S^R and S^A holds if and only the logical clause T holds. Formally,*

$$\begin{aligned} &\boldsymbol{Parse}(T) \wedge \boldsymbol{Evaluate}(T) \\ &\Rightarrow \mathit{SuffAttrGenericMatch}(S^R, S^A). \end{aligned} \tag{4}$$

*where **Parse** and **Evaluate** are functions devoted respectively to parse and evaluate the logical expression T.*

The functional generic match is formalized in Algorithm 2, which follows directly from Sentence (4).

Example 2. An example of a logical expression is "$T = A_5$ or (A_2 and A_3)". In this example, the matching holds when either (i) the matching in respect to attribute A_5 holds, or (ii) the matching in respect to attribute A_2 and the matching in respect to attribute A_3 hold jointly.

Algorithm 2: SuffAttrGenericMatch

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
         C, // Criteria table.
         T, // Logical expression.
Output: Boolean// success/fail.
if (NOT(Parse(T))) then
    return fail;
T' ⟵ T;
Z ⟵ ∅;
for (each A_l ∈ T') do
    if (A_l ∉ Z) then
        t ⟵ false;
        t ⟵ SuffSingleAttrMatch(S^R, S^A, A_l);
        replace all A_l ∈ T by the value of t;
        Z ⟵ Z ∪ {A_l};
if (Evaluate(T)) then
    return success;
return fail;
```

3.3 Computational Complexity

Let first focalize on the complexity of Algorithm 1. The complexity of the two *while* loops in Algorithm 1 is equal to $O(S^R.N) + O(S^A.N)$. Since we generally have $S^A.N \gg S^R.N$, hence the complexity of the two *while* loops is equal to $O(S^A.N)$. Then, the worst case complexity of Algorithm 1 is $O(S^A.N)+\alpha$ where α is the complexity of computing μ. The value of α depends on the approach used to infer μ. As underlined in [7], inferring μ by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines which use the fast Rete [8] 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 $O(S^A.N) + O(|R||F||P|)$. Furthermore, we observe, as in [7], that the process of computing μ is the most "expensive" step of the algorithm. Hence, the complexity of Algorithm 1 is $O(S^A.N)+O(|R||F||P|) \asymp O(|R||F||P|)$.

The complexity of Algorithm 2 depends on the complexity of functions **Parse** and **Evaluate**. The complexity of these functions depends on the data structure used to represent the logical expression T (graph, truth tables, etc.). Clearly the complexity of **Evaluate** function is largely greater than the complexity of **Parse** function. Hence, the complexity of Algorithm 2 is $O(|R||F||P|) + O(\gamma)$ where $O(\gamma)$ is the complexity of **Evaluate** function.

4 Functional Service-Level Matching

The functional service-level matching allows the client to use two types of desired similarity: (i) desired similarity values associated with each attribute in the criteria table, and (ii) a global desired similarity that applies to the service as a whole. The service-level similarity measure quantifies the semantic distance between the requested service and the advertised service entities participating in the match by taking into account both attribute-level and service-level desired similarity measures.

Definition 12. *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be a criteria table. Let β be the service-level desired similarity measure. A sufficient service-level match exists between S^R and S^A if (i) 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 $C.M$, and (ii) the value of overall similarity measure satisfies the desired overall similarity measure β. Mathematically,*

$$\begin{aligned}&[\forall i \quad (\textbf{SuffSingleAttrMatch}(S^R, S^A, A_i)) \quad 1 \le i \le C.N] \wedge \\ &\quad [\exists j_1, \cdots, j_i, \cdots, j_N \quad (\zeta(s_{1,j_1}, \cdots, s_{i,j_i}, \cdots, s_{N,j_N}) \succeq \beta)] \\ &\Rightarrow \textbf{SuffFuncServiceLevelMatch}(S^R, S^A),\end{aligned} \tag{5}$$

where ζ is an aggregation rule; and for $i = 1, \cdots, N$ and $j_i \in \{j_1, \cdots, j_N\}$:

$$s_{i,j_i} = \mu(S^R.A_i, S^A.A_{j_i}).$$

The parameter β may be any of the maps given in Section 2.3. The functional service-level matching is given in [5]. The aggregation rule ζ used in the definition above is a tool to combine the similarity measures into a single similarly measure. In [5], we defined ζ as follows:

$$\zeta : F_1 \times \cdots \times F_N \rightarrow \{\text{Exact, Plug-in, Subsumption,Container, Part-of, Fail}\}$$

where F_j={Exact, Plug-in, Subsumption,Container, Part-of, Fail} ($j = 1, \cdots, N$); and N is the number of attributes included in the criteria table.

The similarity maps and the corresponding strict total order given in Section 2.3 still apply here. Since the similarity measures are defined on an ordinal scale, there are only a few possible aggregation rules that can be used to combine similarity measures [5]: Minimum, Maximum, Median, Floor and Ceil. The Floor and Ceil rules apply only when there is an even number of similarity measures (which leads to two median values).

5 QoS-Oriented Classification

The QoS-oriented matching concerns QoS attributes only and applies to service instances that verify functional requirements. The objective of QoS matching is to assign to each instance an overall QoS level. Instead of sorting services

from best to worst, we propose to categorize them into an ordered set of QoS classes $\mathbf{Cl} = \{Cl_1, \cdots, Cl_p\}$, such that the higher the class, the higher the QoS level. The computing of overall QoS level for each instance requires the use of a multicriteria aggregation rule. In this paper, we will use the simple majority with veto support rule.

5.1 Classification algorithm

Let first introduce some new concepts.

Definition 13. *A QoS Attribute Table, Q, is a relation consisting of three attributes, $Q.A$, $Q.T$ and $Q.S$. $Q.A$ describes the service attribute to be compared, $Q.T$ gives the attribute type and $Q.S$ specifies the scale type. Two types of attributes are distinguished: gain and cost. The gain attributes are those to be maximized while cost attributes are those to be minimized. The scale may be nominal, ordinal, cardinal or ratio. Let $Q.A_i$, $Q.T_i$ and $Q.S_i$ denote the service attribute value, the attribute type and the scale type of the ith tuple of the relation. Let $Q.N$ be the total number of tuples in Q.*

Example 3. Table 2 shows a QoS Attribute Table example. It specifies the parameters of four QoS attributes: response time (A_1), availability (A_2), security (A_3) and cost (A_4).

Table 2. An example QoS Attribute Table

$Q.A$	$Q.T$	$Q.S$
A_1: Response time	cost	Cardinal
A_2: Availability	gain	Cardinal
A_3: Security	gain	Ordinal
A_4: Cost	cost	Cardinal

Definition 14. *A Boundary Matrix, B, consisting of a pairwise matrix composed of $p-1$ columns B_1,$\cdots$,B_{p-1} and N rows corresponding to the number of QoS attributes.*

Example 4. An example of Boundary Matrix is given in Table 3. It specifies three boundaries in respect to the QoS given in Table 2. Table 3 defines four QoS classes.

The attribute type and scale parameters should be used to control input data, especially the definition of boundaries.

Definition 15. *A Weight Table, W, is a relation consisting of two attributes, $W.A$ and $W.V$. $W.A$ describes the service attribute and $W.V$ specifies the weight of this attribute. Let $W.A_i$ and $W.V_i$ denote the service attribute and the attribute weight value in the ith tuple of relation W. The weights values must sum to 1.*

Table 3. An example Boundary Matrix

$Q.A_i$	B_1	B_2	B_3
Response time	11	9.25	8
Availability	0.2	0.3	0.51
Security	2	3	4
Cost	4	3.5	3

Example 5. An example of Weight Table is given in Table 4.

Table 4. An example Weight Table

$W.A$	$W.V$
Response time	0.325
Availability	0.325
Security	0.175
Cost	0.175

Definition 16. *Let $h \in \{1, \cdots, p\}$. The concordance power for the outranking of advertised service S^A over boundary B_h is computed as follows:*

$$\Phi(S^A, B_h) = \sum_{i \in L_1(S^A, h)} W.V_i, \tag{6}$$

where: $L_1(S^A, B_h) = \{i : S^A.A_i \succeq B_h.A_i \wedge Q.T_i = \text{'gain'}\} \cup \{i : S^A.A_i \preceq B_h.A_i \wedge Q.T_i = \text{'cost'}\} \cup \{i : S^A.A_i = B_h.A_i \wedge Q.S_i = \text{'nominal'}\}$.

Example 6. Let consider the service instances given in Table 5. Based on the definition above we obtain $L_1(s_8, B_1) = \{1, 3, 2, 4\}$, $L_1(s_8, B_2) = \{2, 3, 4\}$ and $L_1(s_8, B_3) = \{4\}$. This leads to: $\Phi(s_8, B_1) = 1$, $\Phi(s_8, B_2) = 0.675$ and $\Phi(s_8, B_3) = 0.175$.

Definition 17. *Let $h \in \{1, \cdots, p\}$. The discordance power for the outranking of advertised service S^A over boundary B_h is computed as follows:*

$$\Psi(S^A, B_h) = \prod_{k=1}^{k=N} Z_k(S^A.A_k, B_h.A_k), \tag{7}$$

where:

$$Z_k(S^A.A_k, B_h.A_k) = \begin{cases} \frac{1 - W.V_k}{1 - \Phi(S^A, B_h)}, & \text{if } W.A_k > \Phi(S^A, B_h) \wedge k \in L_2(S^A, B_h) \\ 1, & \text{otherwise.} \end{cases} \tag{8}$$

with $L_2(S^A, B_h) = \{i : S^A.A_i \prec B_h.A_i \wedge Q.T_i = \text{'gain'}\} \cup \{i : S^A.A_i \succ B_h.A_i \wedge Q.T_i = \text{'cost'}\} \cup \{i : S^A.A_i \neq B_h.A_i \wedge Q.S_i = \text{'nominal'}\}$.

Table 5. Web service instances

s_i	A_1	A_2	A_3	A_4
s_i	A_1	A_2	A_3	A_4
s_8	12.82	0.34296	3	2.74
s_9	10.92	0.15	1	2.08
s_{10}	9.52	0.51	4	2.5

Example 7. Let consider the service instances given in Table 5. Based on the definition above we obtain $L_2(s_8, B_1) = \{1\}$, $L_2(s_8, B_2) = \{1\}$ and $L_2(s_8, B_3) = \{1, 2\}$. This leads to: $\Psi(s_8, B_1) = 0.818$, $\Psi(s_8, B_2) = 0.818$ and $\Psi(s_2, B_3) = 0.670$.

Definition 18. *Let $h \in \{1, \cdots, p\}$. The credibility index for the outranking of advertised service S^A over boundary B_h is computed as follows:*

$$\sigma(S^A, B_h) = \Phi(S^A, B_h) \cdot \Psi(S^A, B_h). \tag{9}$$

Example 8. Based on Examples 6 and 7, we obtain $\sigma(s_8, B_1) = 0.818$, $\sigma(s_8, B_2) = 0.552$ and $\sigma(s_8, B_3) = 0.117$.

Definition 19. *Let S^R be the service that is requested and S^A be the service that is advertised. Let $S^R.Q$ be the list of QoS attributes to be utilized for classification. Service S^A is assigned to QoS class Cl_h if for every QoS attribute of S^R there is exists an identical attribute of S^A and the value of the Credibility index is greater or equal to the credibility threshold $\lambda \in [0.5, 1]$ and $\sigma(S^A, B_h) \geq \sigma(S^A, B_{h'})$ for every $h < h'$. Formally,*

$$\begin{array}{l} Argmax_h[\exists_{j,k}(Q.A_i = S^R.A_j = S^A.A_k) \wedge \sigma(S^A, B_h) \geq \lambda] \\ \Rightarrow QoSClassification(S^R, S^A, Cl_h). \end{array} \tag{10}$$

According to this definition, a service S^A is assigned to class Cl_h if and only if: (i) there is a "sufficient" majority of attributes in favor of assigning S^A to Cl_h, and (ii) when the first condition holds, none of the minority of attributes shows an "important" opposition to the assignment of S^A to Cl_h.

Example 9. Let $\lambda = 0.65$. Based on the data and results of the previous example, we conclude that s_8 in Table 5 is assigned to QoS class level 2 since $\sigma(s_8, B_1) = 0.818 > \lambda$, and $\sigma(s_8, B_2) = 0.552 < \lambda$ and $\sigma(s_8, B_3) = 0.117 < \lambda$.

The algorithm for QoS Classification is given in Algorithm 3. This algorithm compares S^A to each of the boundaries staring from the highest one and stops once a sufficient QoS measure holds. The function `CredibilityIndex` computes the credibility index as in Equation (9).

A more simple version of the classification algorithm consists in using the simple majority only. The algorithm based on the simple majority rule is similar to Algorithm 3. We simply need to replace the test "`CredibilityIndex`$(u, h) \geq \lambda$" by "`ConcordancePower`$(u, h) \geq \lambda$". The function `ConcordancePower` computes the Concordance Power as in Equation (6).

Algorithm 3: QoSClassification

```
Input  : S^A, // Advertised service.
         λ, // Credibility threshold.
         Q, // QoS Table.
         B, // Boundary Matrix.
         W, // Weight Table.
Output: Cl = {Cl_1, ···, Cl_p}// Global QoS classes.
p ⟵ number of QoS classes;
Cl_i ⟵ ∅, ∀i = 1, ···, p;
U ⟵ instances of S^A;
for (all u ∈ U) do
    h ⟵ p − 1;
    assigned ⟵ False;
    while (h ≥ 0 ∧ NOT(assigned)) do
        if (CredibilityIndex(u, h, W) ≥ λ) then
            Cl_{h+1} ⟵ Cl_{h+1} ∪ {u};
            assigned ⟵ true;
        h ⟵ h − 1;
Cl ⟵ {Cl_1, ···, Cl_p};
return Cl;
```

5.2 Computational Complexity

Algorithm 3 runs in $O(2mp \times |U|)$ where U is the set of instances. Note that function `ConcordancePower` runs in $O(m)$ and function `CredibilityIndex` runs in $O(2m)$. The complexity of the simple version of the classification algorithm which is based only on the majority rule and which is not given here, is $O(mp \times |U|)$, where m is the number of QoS attributes and p is the number of QoS classes.

5.3 Illustration

Let us consider the list of potential compositions given in Table 6. We assume that these compositions have meet the functional requirements of the user. Table 6 shows the evaluation of the compositions in respect to four QoS attributes (response time (A_1), availability (A_2), security (A_3), and cost (A_4) attributes) given in Table 2. The objective is to classify the compositions into different ordered categories. For the purpose of this example, we assume that the four categories defined by Table 3 and the weights given in Table 4 have been used.

The final classifications obtained by the simple majority and simple majority with veto algorithms where the credibility threshold is $\lambda = 0.65$ are given in Table 6. In this table, we can see that both simple majority and majority with veto algorithms assign instances s_3 and s_{10} to the best QoS class. We remark also that both algorithms assign instances s_5 and s_9 to the worst QoS class. Which is interesting to see in Table 6 is the role of veto effect in the assignment of instances s_8 and s_{13}. Indeed, the QoS of both instances have been decreased (from 3 to 2 for instance s_8 and from 2 to 1 for instance s_{13}) by the majority with veto algorithm. This happens because the weights of attributes which are against the assignment—of instance s_8 to QoS class 3 and instance s_{13} to QoS class 2—have been taken into account.

Table 6. Potential compositions and final classification for λ=0.65

s_i	$A_1(s_i)$	$A_2(s_i)$	$A_3(s_i)$	$A_4(s_i)$	Simple Majority	Majority +Veto
s_1	9.2	0.45946	1	2.48	3	3
s_2	8.12	0.41817	1	2.68	3	3
s_3	8	0.53	4	2.78	4	4
s_4	8.19	0.46967	2	3.24	3	3
s_5	11.15	0.19	1	2.74	1	1
s_6	7.42	0.40317	2	3.38	3	3
s_7	7.72	0.36676	2	3.18	3	3
s_8	12.82	0.34296	3	2.74	3	2
s_9	10.92	0.15	1	2.08	1	1
s_{10}	9.52	0.51	4	2.5	4	4
s_{11}	10.12	0.53294	3	2.68	3	3
s_{12}	10.42	0.48356	1	2.32	2	2
s_{13}	12.52	0.2	1	3.14	2	1
s_{14}	8.42	0.48	1	2.82	3	3
s_{15}	10.32	0.48	4	2.16	3	3

6 Implementation and performance analysis

A prototype called PMCF (Parameterized Matching-Classification Framework) has been implemented. Figure 1 provides the architecture of PMCF. The inputs are the Criteria Table, the published Web services repository, the user request and its corresponding Ontologies. The inputs are parsed by the Semantic Matchmaking Module which filters service offers that match with the Criteria Table. The result should then be passed to the Classification Module which assigns the matching services into different QoS classes. We note that Classification Module is not implemented in the current version of PMCF. We used OWLS API to parse the published Web services list and the user request. The similarity between the concepts inferred using Jena API (see `http://jena.sourceforge.net`). Finally, we note that in the current version, PMCF supports only Input and Output attributes and only the functional attribute-based conjunctive matching.

We used SME2 [13] to evaluate the performance of PMCF. SME2 is an open source tool for testing different semantic matchmakers in a consistent way. SME2 uses OWLS-TC collection to provide the matchmakers with services descriptions, and to compare their answers to the relevance sets of the various queries. Different experimentations was 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 service offers described in OWL-S 1.1 and 42 queries. The implemented Plugin for the experiment required a precise interface, we could not take the Criteria Table as an input, so we assigned to it a default value (Input: Fail, Output:Fail). SME2 gives several metrics to evaluate the performance and effectiveness of a service matchmaker. The metrics that have been considered in this paper are: precision and recall, query response

time and memory time. The definition of these metrics are given in [13][17]. We compared the results of our PMCF matchmaker with SPARQLent approach [24] and and iSEM approach [14]. SPARQLent is a logic-based matchmaker based on the OWL-DL reasoner Pellet to provide exact and relaxed service matchmaking. iSEM is an hybrid matchmaker offering different filter matchings: logic-based, approximate reasoning based on logical concept abduction for matching Inputs and Outputs. We consider only the I-O logic-based matching for the comparison issue. We note that SPARQLent and iSEM consider preconditions and effects of Web services, which is out of the scope of our approach.

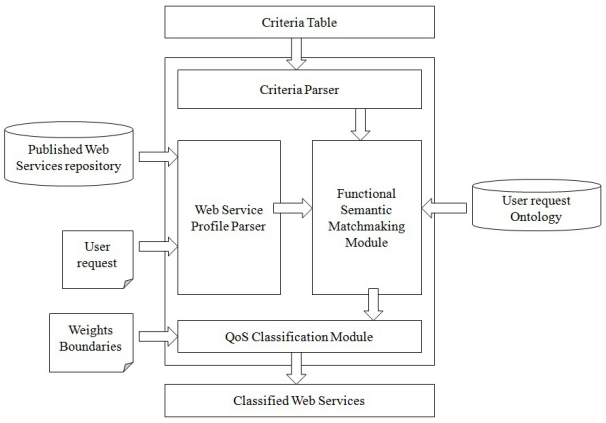

Fig. 1. Architecture of the prototype PMCF

Figure 2.*a* presents the recall precision of PMCF, iSEM logic-based and SPARQLent. This figure shows that PMCF 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.

Figure 2.*b* compares the Query Response Time of PMCF, logic-based iSEM and SPARLent when answering the 42 queries of OWLS-TC; the first column (Avg) gives the average response time for the three matchmakers. The experimental results show that PMCF is more faster than SPARQLent and slightly less faster than logic-based iSEM. We note that SPARQLent has especially high query response time if the query include preconditions/effects. Moreover, SPARQLent is based on an OWL DL reasoner which is an expensive processing. PMCF and iSEM have close query response time because both consider direct parent/child relations in a subsumption graph which reduce significantly the query processing. The PMCF highest query response time limit is 248ms.

Finally, Figure 2.*c* shows the Memory Usage for PMCF, iSEM logic-based and SPARQLent. It is easy to see that PMCF consumes less memory than iSEM logic-based and SPARQLent. This can be explained by the fact that PMCF does

not require a reasoner neither a SPARL queries in order to compute similarities between concepts.

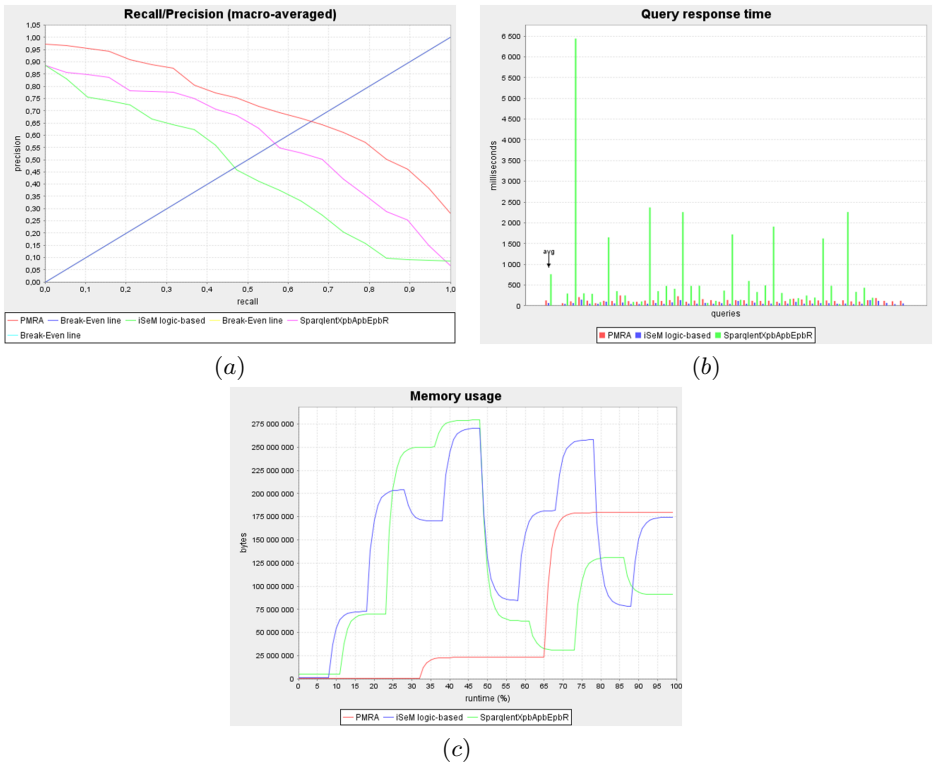

Fig. 2. Results of performance analysis

7 Related Work

In this section we discuss some matchmaking frameworks in respect to several characteristics. The first characteristic is related to the *support of customization* which is an important issue in practice, as recognized by [7]. Most of proposed matching systems ignore this point and only a few ones take into account this aspect. In [7], for instance, the authors present a parameterized semantic matchmaking framework that enables the user to specify the matched attributes and the order in which attributes are compared. In [7], the sufficiency condition defined by the authors is very strict. This problem has been addressed in [5] and in this paper by relaxing matchmaking conditions and supporting three types of matching.

The second characteristic concerns the *type of attributes used in the matching operation*. Most of existing matchmaking frameworks [3][11][18] use only service capability as the criteria for the match. The authors in [7] distinguish two types of matching attributes: capability and property. In [19], the author proposes two approaches to service selection based on QoS attributes. The authors in [23] discuss various techniques of QoS based service selection. The author in [15] proposes a QoS-based web service selection based on a stochastic optimization. In [25], the authors propose a QoS-aware web service selection algorithm based on clustering. The framework presented in this paper identify three types of matching attributes by subdividing property attributes set into two sets of attributes: those directly related to the QoS and those which are not. We think that the proposed framework enhances the above cited proposals, especially the work of [7].

The third characteristic is related to the *method used to compare the requested and advertised services*. Most of existing proposals use simple syntactic and strict capability-based search. In paper [7], the authors present a semantic matchmaking framework that avoids the limitations of strict capability-based matchmaking and in [9] the authors transform the problem of matching web services to the computation of semantic similarity between concepts in domain ontology using a semantic distance measure. The proposal of [2] improve [22]'s matchmaking algorithm and propose a greedy-based algorithm that relies on the concept of matching bipartite graphs. In this paper, we adopted and extended the semantic matchmaking framework proposed by [7].

The fourth characteristic concerns the *support of the multicriteria evaluation*. There are a few proposals that explicitly support multicriteria evaluation, e.g. [6][12][20][21][26]. Most of them use weighted-sum like aggregation techniques. The authors in [26] use linear programming techniques to compute the optimal execution plans for web service. The author in [20] considers two evaluation criteria (time and cost) and assigns to each one a weigh. The best composition of Web services is then decided on the basis of the optimum combined score and a service selection QoS broker by maximizing a utility function is provided by [21]. We note, however, that this type of methods have two main shortcomings: (i) they accept only numerical data and (ii) may lead to the compensation problem since low values may be counterbalanced by high values. The approach used in this paper accepts any type of data and resolves the compensation problem.

8 Conclusion

We presented a QoS-aware semantic matching framework. The framework supports three types of matching: functional attribute-level matching, functional service-level matching, and QoS-based matching. A series of highly customizable algorithms are advertised for each type of matching.

Several issues need to be further investigated. First, the reduction of the number of parameters required from the user by automatically generating the boundaries of QoS classes. Second, the use of the rough sets theory-based classi-

fication [10] for assigning instances to QoS classes. Third, the use of multicriteria ranking methods instead of the classification approach used in this paper.

Acknowledgments. The authors would like to thank Fatma Ezzahra Gmati (National School of Computer Sciences, University of Manouba, Tunis, Tunisia) for developing the prototype and for conducing the performance analysis. The authors would also like to thank Dr. Nadia Yacoubi-Ayad (National School of Computer Sciences, University of Manouba, Tunis, Tunisia) who co-supervised the developing of the prototype and the performance analysis.

References

1. V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A system for end to end composition of web services. *Journal of Web Semantics*, 3:311–339, 2005.
2. U. Bellur and R. Kulkarni. Improved matchmaking algorithm for semantic web services based on bipartite graph matching. In *IEEE International Conference on Web Services*, pages 86–93, Salt Lake City, Utah, USA, 9-13 July 2007.
3. S. 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.
4. S. Chakhar. QoS-enhanced broker for composite web service selection. In *Eighth International Conference on Signal Image Technology and Internet Based Systems (SITIS 2012)*, pages 533–540, Sorrento-Naples, Italy, 25-29 November 2012.
5. 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.
6. 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.
7. P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division, 2004.
8. C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
9. 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.
10. S. Greco, B. Matarazzo, and R. Slowiński. Rough sets theory for multicriteria decision analysis. *European Journal of Operational Research*, 129(1):1–47, 2001.
11. 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.
12. 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.

13. M. Klusch, M. Dudev, J. Misutka, P. Kapahnke, and M. Vasileski. *SME*^{2} *Version 2.2. User Manual*. The German Research Center for Artificial Intelligence (DFKI), Germany, 2010.
14. M. Klusch and P. Kapahnke. The iSEM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(0):1–14, 2012.
15. R. Krithiga. QoS-aware web service selection using SOMA. *Global Journal of Computer Science and Technology*, 12(10):46–51, 2012.
16. H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
17. U. Küster and B. König-Ries. 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, pages 45–59, Berlin, Heidelberg, 2010. Springer-Verlag.
18. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *12th International World Wide Web Conference*, pages 331–339, Budapest, Hungary, 20-24 May 2003.
19. A.S. Ludwig. Memetic algorithm for web service selection. In *Third Workshop on Biologically Inspired Algorithms for Distributed Systems*, BADS '11, pages 1–8, New York, NY, USA, 2011. ACM.
20. D.A. Menascé. Composing web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90, 2004.
21. 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.
22. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference on The Semantic Web*, pages 333–347, Sardinia, Italy, 09-12 June 2002.
23. 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.
24. 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.
25. 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.
26. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality driven web services composition. In *12th International Conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.