QoS-Aware Parameterized Semantic Matchmaking Framework for Web Service Composition

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

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

Abstract: The paper presents a parameterized and highly customizable semantic matchmaking framework. The matchmaking approach on which this framework is based distinguishes three types of matching: functional attribute-level matching, functional service-level matching, and non-functional matching. The functional matching permits to eliminate web services that fail to meet the user functional requirements. The non-functional matching permits to categorize web services instances into different ordered QoS classes. A series of algorithms are advertised for the 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.

1 INTRODUCTION

Individual web services are conceptually limited to relatively simple functionalities modeled through a collection of simple operations. However, for certain types of applications, it is necessary to combine a set of individual web services to obtain *composite* web services (Chakhar et al., 2011). A crucial 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. We distinguish three types of matching: functional attribute-level matching, functional service-level matching, and non-functional matching. The functional attribute-level matching implies capability and property attributes and associates with each attribute a *similarity measure* that should be satisfied by the corresponding attribute in the advertised service. The functional service-level matching supports attribute-level matching and adds a service similarity measure that should be satisfied by the advertised service as a whole. The non-functional matching implies Quality of service (QoS) attributes.

The work presented in this paper is included in a layered system for web service composition that we proposed in (Chakhar, 2012). The matching operation intervenes in different levels of this system, as explained in Section 2.4.

The paper is structured as follows. Section 2 sets the background. Sections 3, 4 and 5 detail the framework. Section 6 provides illustrative examples. Section 7 presents experimental results. Section 8 discusses related work. Section 9 concludes the paper.

2 BACKGROUND

2.1 Example Scenario

This example scenario is freely inspired from a case use scenario described in the WSC Web Services Architecture Usage Scenarios (Hao et al., 2004). A company (travel agent) offers to people the ability to book complete vacation packages: plane/train/bus tickets, hotels, car rental, excursions, etc. Service providers (airlines, bus companies, hotel chains, etc.) propose web services to query their offerings and perform reservations. The user provides a destination and some dates to the travel agent service. The travel agent service inquires service providers about deals and presents them to the user.

2.2 Basic Definitions

We introduce some basic definitions of a service and other service-specific concepts. Several definitions are due to (Doshi et al., 2004).

Definition 1 (**Service**). *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.*

Example 1. *The travel agent company provides a web service,* ***bookVacation****, that is defined by the following attributes:* service category, input, output, preconditions, postconditions, response time, availability, cost, security, *and* geographical location.

Definition 2 (**Service Capability**). *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.*

Example 2. *The capability of* ***bookVacation*** *is:* $S.C = \{$input, output, preconditions, postconditions$\}$.

Definition 3 (**Service Quality**). *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.*

Example 3. *The Quality of* ***bookVacation*** *is:* $S.Q = \{$response time, availability, cost, security$\}$.

Definition 4 (**Service Property**). *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.*

Example 4. *The property of* ***bookVacation*** *is:* $S.P = \{$service category, geographical location$\}$.

2.3 Composition Approach

The key elements of the composition approach that we proposed in (Chakhar, 2012) are: the composition graph, potential executable plans and executable plan. The *composition graph* is an abstract representation of functional requirements provided by the user. It models the *invocation* relationships between the individual web services contained in the composite web service. The set of *potential executable plans* is the composite *service instances* which are obtained by replacing each service type in the composition graph by its instances using a set of transformation rules. The objective of the transformation operation is to include the different semantics of BPEL constructors. Among the different potential executable plans only one—called *executable plan*—should be selected and transformed to a workflow for effective execution.

The composition operation starts by user specification of functional and non-functional requirements and leads to an executable plan that can be handed off to runtime environment for execution. The proposed approach to support the composition operation contains three phases:

1. *Logical composition*: First, the functional requirements provided by the user are used to generate the composition graph.
2. *Physical composition*: Second, the composition graph is transformed to obtain the set of potential executable plans.
3. *Evaluation and selection*: Third, the different potential executable plans are evaluated and compared in order to select one executable plan. The latter is then transformed into a workflow and then deployed, discovered and invoked.

The service composition approach is implemented by a layered system called QoSeBroker (for QoS-enhanced Broker). The architecture of QoSeBroker is given in Figure 1. A detailed description of QoSeBroker is given in (Chakhar, 2012). At this level, we just mention that the matching operation intervenes in the logical and physical composition phases, as briefly explained in the next subsection.

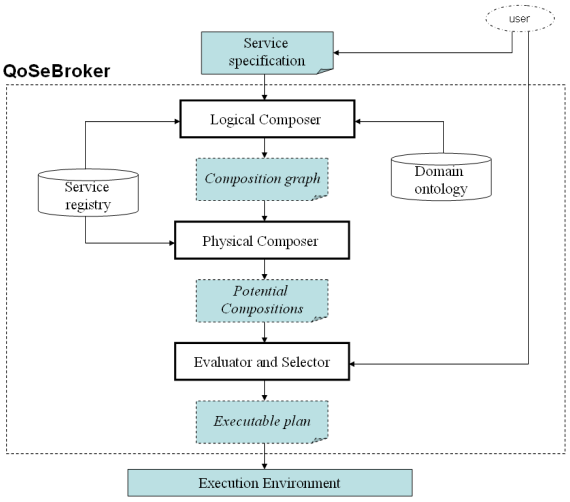

Figure 1: QoSeBroker architecture.

2.4 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 (Chakhar et al., 2011): (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) (Agarwal et al., 2005) for expressing functional requirements and the QoS for non-functional requirements.

Furthermore, we may distinguish three types of service matching (Chakhar, 2013):

- **Functional attribute-level matching**: This type of matching implies capability and property attributes and consider each matching attribute independently of the others.
- **Functional service-level matching**: This type of matching implies capability and property attributes but the matching operation implies attributes both independently and jointly.
- **Non-functional matching**: This type of matching implies QoS attributes.

These different matching types intervene in different composition phases. The functional matching intervenes in the logical composition phase. Indeed, the construction of the composition graph needs first the identification of candidate service types. This operation requires matching the preconditions of a web service with the effects of another up front during filtering by using only functional requirements. The functional matching takes as input all candidate web services and outputs a set of web services that meet the user functional matching criteria. During logical composition phase, service types that fail to meet the user functional requirements are automatically eliminated. At this level, it is important to mention that functional service-level matching supports jointly attribute-level and service-level matching. Hence, the user should select either functional attribute-level matching only or functional service-level matching only.

The non-functional matching intervenes during the physical composition phase. In fact, the first step in this phase is to identify, for each service type in the composition graph, the set of the corresponding instances. For this purpose, the matching operation uses the composition graph and queries the service instances registry to associate to each service type a set of potential instances (at last one instance per type is needed). The second step is then to use non-functional attributes in order to classify web service instances into different QoS ordered classes. It is easy to see that there is no elimination of web service instances at this level.

In (Chakhar, 2013) we discussed functional attribute-level and functional service-level matching. However, the functional attribute-level matching presented in (Chakhar, 2013) assumes only conjunctive or disjunctive logical relationships between matching attributes. This paper enhances our proposal in (Chakhar, 2013) by adding generic functional attribute-level matching and non-functional QoS-oriented matching.

3 FUNCTIONAL ATTRIBUTE-LEVEL MATCHING

Functional matching may be defined as the process of discovering a service advertisement that *sufficiently* satisfies a service request (Doshi et al., 2004). Functional matching is based on the concept of *sufficiency*, which itself is based on the *similarity* measure.

3.1 Similarity Measure

A semantic match between two entities frequently involves a similarity measure. The similarity measure quantifies the semantic distance between the two entities participating in the match. As in (Doshi et al., 2004), a similarity measure is defined as follows.

Definition 5 (Similarity Measure). *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,*

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

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
- **Disjoint** map: if none of the previous cases applies.

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

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

$$\textbf{\textit{Exact}} \succ \textbf{\textit{Plug-in}} \succ \textbf{\textit{Subsumption}} \succ \textbf{\textit{Container}} \succ \textbf{\textit{Part-of}} \succ \textbf{\textit{Disjoint}}$$

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

3.2 Computing the Similarity Measure

The computing of the similarity measure is based on the conceptual annotations associated with the requested and advertised web services. The matching process uses ontological representation of web services to infer the submission hierarchy which leads to the recognition of semantic matches despite their syntactic differences and difference in modeling abstractions between requests and advertisements.

The similarity measure, which represents the degree of match, is defined along a discrete scale in which the 'Exact' match is the highest level and is the preferable while 'Fail' is the lowest level and it represents an undesirable result. Note that the labels used to define the different similarity measures can be mapped to numerical values. However, only comparison operations are authorized (since we assumed that these values are ordinal).

Let now explain how the similarity measure μ is computed. For this purpose we consider the travel agent scenario example given in Section 2.1. Figure 2 shows a fragment of the Travel Agent Ontology. We consider an *Advertisement* for the travel agent web service who permits the client to book *Accommodation* in the specified *Destination*. Assume that the *Advertisement* has the following Inputs and Outputs:

Inputs:	*Destination*
Outputs:	*Accommodation*, *Sport*, *Cost*

We consider a *Query* from a client who planning a vacation. The client wants to make reservations for a *Hotel* and an *Excursion* at the specified *Destination*. As shown in Figure 2, the *Hotel* is a subclass of the concept *Accommodation* and *Excursion* is a subclass of the concept *Entertainment*. The Query has the following Inputs and Outputs:

Inputs:	*Destination*
Outputs:	*Hotel*, *Excursion*

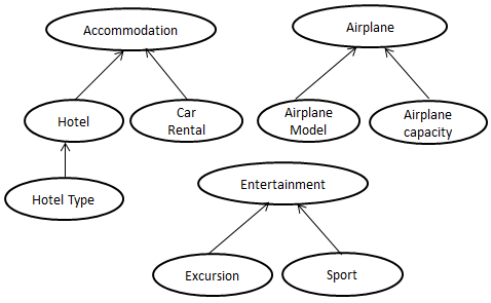

Figure 2: Travel Ontology extract.

Let now focalize on the process of matching *Query* outputs. The same reasoning applies to the process of matching the inputs. The matching process iterates over the list of output concepts of the *Query* and looks to find a match to an output concept in the *Advertisement*. Intuitively, any concept in the output represents a candidate for the match. In this particular example, the initial candidate list is $\{Accommodation, Sport, Cost\}$. The matching process first looks to see if there is a match for *Hotel*. Based on Figure 2, the match between *Hotel* and *Accommodation* will be flagged Exact. Then, the matching process looks a match for *Excursion*. Based on Figure 2, the match for *Excursion* with respect to *Accommodation*, *Sport* and *Cost* will fail (since there is no relationship between these concepts and *Excursion* concept). Hence, we can conclude that $\mu(outputR, outputA)$='Fail' where *outputR* and *outputA* are the output attributes of requested and advertised services, respectively. Using the same reasoning, we obtain an Exact match for the input concept *Destination*, i.e., $\mu(inputR, inputA)$='Exact' where *inputR* and *inputA* are the input attributes of requested and advertised services, respectively.

3.3 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 (Doshi et al., 2004), that serves as a parameter to the matching process.

Definition 7 (**Criteria Table**). *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 5. *Table 1 shows a Criteria Table example.*

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

Definition 8 (**Sufficient Functional Conjunctive Match**). *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,

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

Table 1: An example Criteria Table.

C.A	C.M
input	Exact
output	Exact
precondition	Subsumes
postcondition	Subsumes

The functional attribute-level conjunctive match is formalized in Algorithm 1 in (Chakhar, 2013). A less restrictive definition of sufficiency consists in using a disjunctive rule on the individual matching measures.

Definition 9 (**Sufficient Functional Disjunctive Match**). *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,*

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

The functional attribute-level disjunctive match is formalized in Algorithm 2 in (Chakhar, 2013).

3.4 Generic Matching

In this section we extend the algorithms proposed in (Chakhar, 2013) 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 (**Sufficient Single Attribute Match**). *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,*

$$\exists_{j,k}(C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i) \Rightarrow SuffSingleAttrMatch(S^R, S^A, A_i). \qquad (3)$$

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 $(j \leq S^R.N)$ **do**
  if $(S^R.A_j = C.A_i)$ **then**
    Append $S^R.A_j$ to *rAttrSet*;
  end
  Assign $j \longleftarrow j+1$;
end
while $(k \leq S^A.N)$ **do**
  if $(S^A.A_k = C.A_i)$ **then**
    Append $S^A.A_k$ to *aAttrSet*;
  end
  Assign $k \longleftarrow k+1$;
end
if $(\mu(rAttrSet[i], aAttrSet[i]) \succeq C.M_i)$ **then**
  return success;
end
return fail;

The sufficient functional generic match is then defined as follows.

Definition 11 (**Sufficient Functional Generic Match**). *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,*

$$\textbf{\textit{Parse}}(T) \wedge \textbf{\textit{Evaluate}}(T) \Rightarrow SuffAttrGenericMatch(S^R, S^A) \qquad (4)$$

*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 6. *A 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.*

3.5 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 (Doshi et al., 2004), inferring μ by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines which use the fast Rete (Forgy, 1982) 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 (Doshi et al., 2004), that the process of computing μ is the most "expensive" step of the algorithms. Hence, we obtain: $O(S^A.N)+O(|R||F||P|) \asymp O(|R||F||P|)$.

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.

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;
end
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};
    end
end
if (Evaluate(T)) then
    return success;
end
return fail;
```

4 SERVICE-LEVEL MATCHING

The 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 (Sufficient Functional Service-Level Match). *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,*

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

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 3.1. The functional service-level matching in formalized Algorithm 4 in (Chakhar, 2013). The aggregation rule ζ used in the definition above is a tool to combine the similarity measures into a single similarly measure. In (Chakhar, 2013), we defined ζ as follows:

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

where F_j={Exact, Plug-in, Subsumption,Container, Part-of, Disjoint} ($j = 1, \cdots, N$); and N is the number of attributes included in the criteria table. The similarity maps and the corresponding strict total order given in Section 3.1 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 (Chakhar, 2013): 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 MATCHING

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 distinguish two types of aggregation rules: (i) simple majority, and (ii) simple majority with veto.

5.1 Simple Majority-Based Matching

To formalize the first version of QoS matching, we need to introduce some new concepts.

Definition 13 (**QoS Attribute Table**). *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 7. *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 (**Boundary Matrix**). *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 8. *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 (**Weight Table**). *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 9. *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 (**Concordance Power**). *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:*

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

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 10. *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 $\Phi(s_8, B_3) = 0.175$.*

Table 5: Web service instances.

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

Definition 17 (**Sufficient Simple Majority-Based QoS Matching**). *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 matching. 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 Concordance Power is greater or equal to the credibility threshold λ and $\Phi(S^A, B_h) \geq \Phi(S^A, B_{h'})$ for every $h < h'$. Formally,

$$Argmax_h[\exists_{j,k}(Q.A_i = S^R.A_j = S^A.A_k) \wedge \Phi(S^A, B_h) \succeq \lambda] \Rightarrow QoSMajorityMatch(S^R, S^A, Cl_h). \quad (6)$$

where $\lambda \in [0.5, 1]$.

The parameter λ, called *credibility threshold*, ensures that at least 50% of attributes are in favor of the assignment of the advertised service to the considered QoS class. Hence, a service S^A is assigned to class Cl_h if and only if there is a "sufficient" majority of attributes in favor of assigning S^A to Cl_h.

This first version of QoS matching is formalized 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 *ConcordancePower*, which computes the concordance power as in Definition (16), is given in Algorithm 4.

Algorithm 3: QoSMajorityMatching

```
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 (ConcordancePower(u, h) ≥ λ) then
            Cl_{h+1} ⟵ Cl_{h+1} ∪ {u};
            assigned ⟵ true;
        end
        h ⟵ h − 1;
    end
end
Cl ⟵ {Cl_1, ··· , Cl_p};
return Cl;
```

Example 11. *Let $\lambda = 0.65$. Based on the previous example, we conclude that s_8 in Table 5 is assigned by Algorithm 3 to QoS class level 3 since $\Phi(s_2, B_3) = 0.175 < \lambda$ and $\Phi(s_8, B_2) = 0.675 > \lambda$.*

5.2 Simple Majority and Veto Based QoS Matching

A further customization consisting in the support of the veto effect by taking into account the opposition of

Algorithm 4: ConcordancePower

```
Input  : S^A, // Advertised service.
         h, // Integer (Boundary index).
Output: Φ(S^A, B_h)// Concordance Power.
s ⟵ 0;
for (all Q.A_i ∈ Q.A) do
    switch Q.T_i do
        case ('gain')
            if (S^A.A_i ≥ B_h.A_i) then
                s ⟵ s + W.V_i;
            end
        end
        case ('cost')
            if (S^A.A_i ≤ B_h.A_i) then
                s ⟵ s + W.V_i;
            end
        end
        case ('nominal')
            if (S^A.A_i = B_h.A_i) then
                s ⟵ s + W.V_i;
            end
        end
    end
end
return s;
```

minority attributes, i.e., attributes that do not belong to set L_1. The definition of QoS matching with support of veto effect requires the introduction of some new concepts.

Definition 18 (Discordance Power). *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:*

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

where:

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

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

Example 12. *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 19 (Credibility Index). *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:*

$$\sigma(S^A, B_h) = \Phi(S^A, B_h) \cdot \Psi(S^A, B_h).$$

Example 13. *Based on Examples 10 and 12, 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 20 (Sufficient Majority With Veto Based QoS Matching). *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 matching. 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* λ *and* $\sigma(S^A,B_h) \geq \sigma(S^A,B_{h'})$ *for every* $h < h'$. *Formally,*

$$Argmax_h[\exists_{j,k}(Q.A_i = S^R.A_j = S^A.A_k) \wedge \sigma(S^A,B_h) \geq \lambda] \Rightarrow QoSMajorityVetoMatching(S^R,S^A,Cl_h). \quad (7)$$

where $\lambda \in [0.5,1]$.

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.

The algorithm for the second version of QoS matching is similar to Algorithm 3. We simply need to replace the test "$ConcordancePower(u,h) \geq \lambda$" by "$CredibilityIndex(u,h) \geq \lambda$". The function *CredibilityIndex*, which computes the credibility index as in Definition (19), is given in Algorithm 5.

Algorithm 5: CredibilityIndex

```
Input  : S^A, // Advertised service.
         h, // Integer (Boundary index).
Output: σ(S^A,B_h)// Credibility Index.
z ⟵ 1;
s ⟵ ConcordancePower(S^A,h);
for (all Q.A_i ∈ Q.A) do
    switch Q.T_i do
        case ('gain')
            if (S^A.A_i < B_h.A_i ∧ W.A_i > s) then
                z ⟵ z · (1−W.A_i)/(1−s);
            end
        end
        case ('cost')
            if (S^A.A_i > B_h.A_i ∧ W.A_i > s) then
                z ⟵ z · (1−W.A_i)/(1−s);
            end
        end
        case ('nominal')
            if (S^A.A_i ≠ B_h.A_i ∧ W.A_i > s) then
                z ⟵ z · (1−W.A_i)/(1−s);
            end
        end
    end
end
return s · z;
```

Example 14. *Let* $\lambda = 0.65$. *Based on the data and results of the previous example, we conclude that* s_8 *in Table 5 is assigned by majority with veto Algorithm 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 previous example shows that the QoS of instance s_8 has decreased (from 3 to 2) in comparison to Example (11). This because the weights of attributes which are against the assignment of instance s_8 to QoS class 3 have been taken into account.

5.3 Computational Complexity

Algorithm 3 runs in $O(mp \times |U|)$, where m is the number of QoS attributes and p is the number of QoS classes. The second version of Algorithm 3 based on simple majority with veto and which is not provided in this paper, runs in $O(2mp \times |U|)$. Note that Algorithm 4 runs in $O(m)$ and Algorithm 5 runs in $O(2m)$.

6 ILLUSTRATION

Consider again the example given in Section 3.2. Assume that the Criteria Table is as in Table 6. Based on the discussion given in Section 3.2, the desired similarity specified by this table holds only for *Destination* and *Hotel*. Hence, the attribute-level conjunctive matching algorithm will fail but the attribute-level disjunctive matching algorithm will success.

Table 6: Criteria Table.

C.A	*C.M*
input	Exact
output	Exact

Let now assume that the user has specified the following logical expression:

expression:	*Destination* and (*Hotel* or *Excursion*)

The desired similarity specified by the Criteria Table holds only for *Destination* and *Hotel*. But since the expression "*Destination* and (*Hotel* or *Excursion*)" will be flagged true, the functional generic matching algorithm will success.

Let now focalize on service-level functional matching. Suppose that the *Advertisement* has the following Inputs, Outputs and Categories:

Inputs:	*Destination*
Outputs:	*Accommodation*, *Entertainment*, *Cost*
Categories:	*AirplaneModel*

Suppose also that *Query* has the following Inputs, Outputs and Categories:

Inputs:	*Destination*
Outputs:	*HotelType*, *Excursion*
Categories:	*AirplaneModel*

Based on the same reasoning as earlier, the match for *Destination*, *Excursion* and *AirplaneModel* concepts will be flagged Exact; and the match for concept *HotelType* will be flagged Plug-in. Assume that the Criteria Table is as in Table 7.

Table 7: Criteria Table.

C.A	*C.M*
input	Exact
output	Exact
service category	Subsumption

The instantiation of the aggregation rule ζ (see Section 4) will then look like ζ(Exact, min{Plug-in, Exact}, Exact) which leads to ζ(Exact, Plug-in, Exact). Then, the result of aggregation according to different aggregation rules (except Median which does not apply here) is given in Table 8.

Table 8: Result of aggregation.

Aggregation rule(ζ)	Minimum	Maximum	Floor	Ceil
Result	Plugin	Exact	Plugin	Exact

Let now focalize on non-functional matching and consider the list of potential compositions given in Table 9. We assume that these compositions have meet the functional requirements of the user. Table 9 shows the evaluation of the compositions in respect to four QoS attributes (i.e. 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 9. In this table, we can see that both simple majority and majority with veto algorithms (denoted Algo 3 and Algo 3' in Table 9, respectively) 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 9 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 9: Potential compositions and final classification for λ=0.65.

s_i	$A_1(s_i)$	$A_2(s_i)$	$A_3(s_i)$	$A_4(s_i)$	Algo 3	Algo 3'
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

7 EXPERIMENTAL RESULTS

For the purpose of experimental tests, we adopted the same architecture proposed by (Bellur and Kulkarni, 2007). We used the Protege editor Protege (http://protege.stanford.edu/) to browse and edit OWL ontologies. The Mindswap OWL-S API has been used to load the OWL ontologies into the Knowledge Base and parse the OWL-S Queries and Advertisements. We used the Pellet reasoner Pellet (http://pellet.owldl.com/) to classify the loaded ontologies and Jena API JENA (http://jena.sourceforge.net) to query the reasoner for concept relationships. To test the algorithms, we used 14 ontologies from the OWTLS-TC (service retrieval test collection from SemWebCentral) in our Knowledge Base. About 700 advertisements from OWLS-TC were loaded into the advertisement repository. All measurement points shown are average results taken from 30 runs. The data sets for clients and providers were randomly generated.

The result of experimental tests is given in Figures 3 and 4. Figure 3 shows the execution time of Algorithms 1, 2 and 3. Two versions (denoted Algo 3 and Algo 3' in this figure) of Algorithm 3 have been tested: the first version (Algo 3) uses the simple majority and the second version (Algo 3') uses simple majority with veto. Figure 4 shows the evolution of execution time in respect to the number of instances for the two versions of Algorithm 3. As shown in this figure, it is easy to see that the execution time varies

asymptotically for both versions of Algorithm 3. We note, however, that the second version (Algo 3') is more expensive in execution time. This because the second version of Algorithm 3 needs to execute Algorithms 4 and 5 (for computing the concordance and discordance powers) while the first version (Algo 3) of Algorithm 3 needs to execute only Algorithm 4.

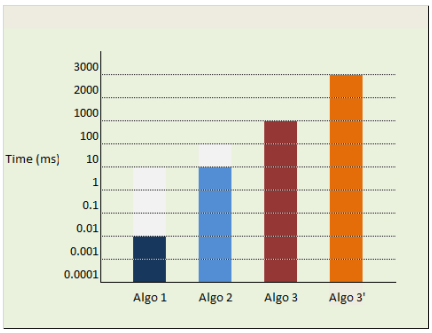

Figure 3: Execution time of algorithms.

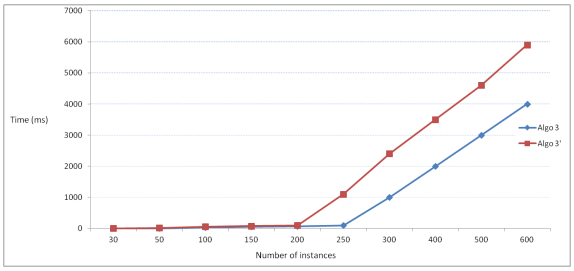

Figure 4: Execution time vs number of instances.

8 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 (Doshi et al., 2004). Most of proposed matching systems ignore this point and only a few ones take into account this aspect. (Doshi et al., 2004), for instance, present a parameterized semantic matchmaking framework that enables the user to specify the matched attributes and the order in which attributes are compared. In (Doshi et al., 2004), the sufficiency condition defined by the authors is very strict. This problem has been addressed in (Chakhar, 2013) 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 (Ben Mokhtar et al., 2006; Guo et al., 2005; Li and Horrocks, 2003) use only service capability as the criteria for the match. The authors in (Doshi et al., 2004) distinguish two types of matching attributes: capability and property. (Ludwig, 2011) proposes two approaches to service selection based on QoS attributes. (Sathya et al., 2011) discuss various techniques of QoS based service selection. (Krithiga, 2012) proposes a QoS-based web service selection based on a stochastic optimization. (Xia et al., 2011) propose a QoS-aware web service selection algorithm based on clustering. The framework presented in this paper identify three types of matching attributes (capability, property, and QoS) 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 (Doshi et al., 2004).

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. (Doshi et al., 2004) present a semantic matchmaking framework that avoids the limitations of strict capability-based matchmaking. (Fu et al., 2009) transform the problem of matching web services to the computation of semantic similarity between concepts in domain ontology using a semantic distance measure. (Bellur and Kulkarni, 2007) improve (Paolucci et al., 2002)'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 (Doshi et al., 2004).

The fourth characteristic concerns the *support of the multicriteria evaluation*. There are a few proposals that explicitly support multicriteria evaluation, e.g. (Cui et al., 2011; Jeong et al., 2007; Menascé, 2004; Menascé and Dubey, 2007; Zeng et al., 2003). Most of them use weighted-sum like aggregation techniques. (Zeng et al., 2003) use linear programming techniques to compute the optimal execution plans for web service. (Menascé, 2004) 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. (Menascé and Dubey, 2007) propose a service selection QoS broker by maximizing a utility function. 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.

9 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 classification (Greco et al., 2001) for assigning instances to QoS classes. Third, the use of multicriteria ranking/choice approaches instead of the classification approach used in this paper.

REFERENCES

Agarwal, V., Chafle, G., Dasgupta, K., Karnik, N., Kumar, A., Mittal, S., and Srivastava, B. (2005). Synthy: A system for end to end composition of web services. *Journal of Web Semantics*, 3:311–339.

Bellur, U. and Kulkarni, R. (2007). 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.

Ben Mokhtar, S., Kaul, A., Georgantas, N., and Issarny, V. (2006). Efficient semantic service discovery in pervasive computing environments. In *ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 240–259, Melbourne, Australia.

Chakhar, S. (2012). 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.

Chakhar, S. (2013). 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.

Chakhar, S., Youcef, S., Mousseau, V., Mokdad, L., and Haddad, S. (2011). Multicriteria evaluation-based conceptual framework for composite web service selection. Working Paper, Lamsade, University Paris-Dauphine, France. http://basepub.dauphine.fr/ xmlui/bitstream/handle/123456789/5283/multicriteria mokdad.PDF?sequence=2.

Cui, L., Kumara, S., and Lee, D. (2011). Scenario analysis of web service composition based on multi-criteria mathematical goal programming. *Service Science*, 3(4):280–303.

Doshi, P., Goodwin, R., Akkiraju, R., and Roeder, S. (2004). Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division.

Forgy, C. (1982). Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37.

Fu, P., Liu, S., Yang, H., and Gu, L. (2009). Matching algorithm of web services based on semantic distance. In *International Workshop on Information Security and Application (IWISA 2009)*, pages 465–468, Qingdao, China.

Greco, S., Matarazzo, B., Slowinski, R., and Stefanowski, J. (2001). An algorithm for induction of decision rules consistent with the dominance principle. In Ziarko, W. and Yao, Y., editors, *Rough Sets and Current Trends in Computing*, volume 2005 of *LNCS*, pages 304–313. Springer Berlin Heidelberg.

Guo, R., Le, J., and Xiao, X. (2005). Capability matching of web services based on OWL-S. In *Sixteenth International Workshop on Database and Expert Systems Applications*, pages 653–657.

Hao, H., Haas, H., and Orchard, D. (2004). Web services architecture usage scenarios. W3C Working Group Note 11, IBM Research Division.

Jeong, B., Cho, H., Kulvatunyou, B., and Jones, A. (2007). A multi-criteria web services composition problem. In *IEEE International Conference on Information Reuse and Integration(IRI 2007)*, pages 379–384.

Krithiga, R. (2012). QoS-aware web service selection using SOMA. *Global Journal of Computer Science and Technology*, 12(10):46–51.

Li, L. and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *12th International World Wide Web Conference*, pages 331–339, Budapest, Hungary.

Ludwig, A. (2011). 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. ACM.

Menascé, D. (2004). Composing web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90.

Menascé, D. and Dubey, V. (2007). Utility-based QoS brokering in service oriented architectures. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 422–430.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002). Semantic matching of web services capabilities. In *First International Semantic Web Conference on The Semantic Web*, pages 333–347, Sardinia, Italy.

Sathya, M., Swarnamugi, M., Dhavachelvan, P., and Sureshkumar, G. (2011). Evaluation of QoS based web-service selection techniques for service composition. *International Journal of Software Engineering*, 1(5):73–90.

Xia, Y., Chen, P., Bao, L., Wang, M., and Yang, J. (2011). A QoS-aware web service selection algorithm based on clustering. In *IEEE International Conference on Web Services (ICWS)*, pages 428–435.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. (2003). Quality driven web services composition. In *12th International Conference on World Wide Web*, pages 411–421, New York, NY, USA. ACM.