Parameterized Attribute and Service Levels Semantic Matchmaking Framework for Service Composition

Salem Chakhar
CRAD, ESAD, University Laval, Québec, QC, Canada
Email: salem.chakhar@crad.ulaval.ca

***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: (i) functional attribute-level matching, (ii) functional service-level matching, and (iii) non-functional matching. This paper focuses only on functional matching. A series of algorithms is advised for both attribute-level and service-level matching. These algorithms are designed to support a customizable matching process that permits the user to control the attributes matched, the order in which attributes are compared, and the way the sufficiency is computed for both attribute and service levels. Experimental results show that both the proposed algorithms and the original one perform pretty much in the same way.**

***Keywords-web service*; *service composition*; *matchmaking*; *attribute-level matching*; *service-level matching*.**

I. 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 more complex ones, called *composite* or *aggregated* web services [1]. One important issue within web service composition is related to the selection of the most appropriate one among the different candidate composite web services. The selected web service(s) should *mach* at best the specifications provided by the user. Most of existing matchmaking frameworks such as [2] [3] [4] utilize a strict capability-based matchmaking, which is proven [5] to be inadequate in practice. Some recent proposals including [6] [7] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only. To avoid the shortcomings of strict capability-based matchmaking, Doshi et al. [5] present a parameterized semantic matchmaking framework that exhibits a customizable matchmaking behavior. One important shortcoming of [5] is that the sufficiency condition defined by the authors is very strict since it requires that all the specified conditions hold at the same time. This seems to be very restrictive in practice, especially for attributes related to the Quality of Service (QoS).

The objective of this paper is to propose a semantic matchmaking framework for web service composition. In this framework, we distinguish three types of matching: (i) functional attribute-level matching, (ii) functional service-level matching, and (iii) non-functional matching. The functional attribute-level matching concerns capability and property attributes. 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 deals with the QoS attributes. This paper focalizes on functional matching only.

The rest of the paper is as follows. Section II sets the background. Section III addresses functional attribute-level matching. Section IV deals with functional service-level matching. Section V presents experimental results. Section VI discusses related work. Section VII concludes the paper.

II. BACKGROUND

A. Example Scenario

For the purpose of illustration, we consider a web service use case concerning travel reservation. This example is freely inspired from a use case scenario described in the WSC Web Services Architecture Usage Scenarios [8].

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) are providing web services to query their offerings and perform reservations.

The user gets the location of a travel agent service via an unspecified way (search engine, service directory, etc).

The user provides a destination and some dates to the travel agent service. The travel agent service inquires airlines about deals and presents them to the user.

B. Basic Definitions

We introduce some basic definitions of a service and other service-specific concepts. Some ones are due to [5].

*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 $S.N$ denote the cardinality of this set.$\Diamond$

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.$\Diamond$

*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.$\Diamond$

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

 ISBN: 978-1-61208-247-9

*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 Service 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$\}$.◊

C. Service Matching Types

The input for web service composition is a set of specifications describing the capabilities of the desired service. These specifications can be decomposed into two groups:

- Functional requirements that deal with the desired functionality of the composite service,
- Non-functional requirements that relate to issues like cost, performance and availability.

The non-functional requirements are often called QoS attributes. The QoS attributes can be subdivided into two groups [1]:

- *Hard attributes*: These correspond to QoS attributes for which some obligatory constraints are imposed by the client. Instances that fail to meet these constraints are automatically eliminated,
- *Soft attributes*: These correspond to QoS attributes that should be optimized, i.e., maximized or minimized, according to the user desires.

Hard QoS attributes are very useful in practice since they permit to reduce the computing time. They should be used in a *preliminary analysis step* [1] to reduce the number of candidate compositions. Soft QoS attributes, in the contrary, are used to compute the overall QoS of the remaining candidate compositions.

Furthermore, we may distinguish three types of service matching [1]:

- **Functional attribute-level matching**: This type of matching concerns capability and property attributes and consider each matching attribute independently of the others.
- **Functional service-level matching**: This type of matching concerns capability and property attributes but the matching operation implies the service as a whole.
- **Non-functional matching**: This type of matching concerns QoS attributes. It uses soft QoS attributes.

In the rest of this paper, we will focalize only on functional attribute-level and service-level matching. The non-functional QoS-oriented matching is addressed in [9].

III. Functional Attribute-Level Matching

Functional matching may be defined as the process of discovering a service advertisement that *sufficiently* satisfies a service request [5]. Functional matching is based on the concept of *sufficiency*. The latter relies on the *similarity* measure.

A. 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. Following [5], 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,

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

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

A preferential total order may now be established on the above mentioned similarity maps.

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

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

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

This definition of similarity measure is due to [5]. Other matchmaking frameworks (e.g., [10] [5] [3] [4]) utilize an idea similar to μ, but label it differently.

B. Matchmaking Supported Customizations

The functional attribute-level matching algorithms proposed in this paper extend [5]'s matchmaking algorithm. Indeed, to enhance the work of [5], we propose an additional customization by allowing the user to specify the way the similarity measures are logically aggregated. Indeed, in [5]'s matching algorithm, the match must hold conjointly for all attributes. However, it may be useful to allow the user to specify different types of logical expressions. In fact, other logical connectors than "AND" may be used to combine attribute-level similarity measures.

Accordingly, we distinguish three types of functional attribute-level matching: (i) conjunctive matching based on the use of "AND" connector, (ii) disjunctive matching based on the use of "OR" connector, and (iii) complex matching that uses different logical connectors, especially

 ISBN: 978-1-61208-247-9

"AND", "OR" and "NOT" operators. For the purpose of this paper, only conjunctive and disjunctive functional attribute-level matching are considered. Complex matching is addressed in [9].

C. Functional Attribute-Level Conjunctive Matching

As specified above, functional matching concerns only capability and property attributes. 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 (capability and property) attribute in $S^R.A$. In this case, 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. A second customization of the matching process is to allow the user specifying which attributes of the service requested should be utilized during the matching process, and the order in which the attributes must be considered for comparison.

In order to support these customizations, we use the concept of Criteria Table, introduced by [5], 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. $C.\eta$ denotes the total number of tuples in C.$\Diamond$

Example 5: Table I shows a Criteria Table example.$\Diamond$

Table I
AN EXAMPLE CRITERIA TABLE

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

A sufficient functional attribute-level conjunctive match between services can now be 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,

$$\forall_i \exists_{j,k} (C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \Rightarrow \text{SuffFunctionalConjunctiveMatch}(S^R, S^A) \quad 1 \leq i \leq C.\eta.\Diamond \tag{1}$$

The functional attribute-level conjunctive match is formalized in Algorithm 1. This algorithm follows directly from Sentence (1).

D. Functional Attribute-Level Disjunctive Matching

A less restrictive definition of sufficiency consists in using a disjunctive rule on the individual similarity measures. The attribute-level disjunctive matching is defined as follows.

*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,

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

The functional attribute-level disjunctive match is formalized in Algorithm 2. This algorithm follow directly from Sentence (2). Algorithms (1) and (2) differ only at the level of the last *while* loop.

Algorithm 1: FunctionalConjunctiveMatching

```
Input  : S^R, // requested service.
         S^A, // advertised requested.
         C, // criteria table.
Output: Boolean // success: true; fail: false.
while (i ≤ C.η) do
    while (j ≤ S^R.N) do
        if (S^R.A_j = C.A_i) then
            Append S^R.A_j to rAttrSet;
        end
        Assign j ⟵ j + 1;
    end
    while (k ≤ S^A.N) do
        if (S^A.A_k = C.A_i) then
            Append S^A.A_k to aAttrSet;
        end
        Assign k ⟵ k + 1;
    end
    Assign i ⟵ i + 1;
end
while (t < C.η) do
    if (μ(rAttrSet[t], aAttrSet[t]) ≺ C.M_t) then
        return fail;
    end
    Assign t ⟵ t + 1;
end
return success;
```

E. Computational Complexity

Algorithms 1 and 2 have the same complexity. Generally, we have $S^A.N \gg S^R.N$, hence the complexity of the first outer *while* loop is $O(C.\eta \times S^A.N)$. Then, the worst case complexity of our algorithms is $O(C.\eta \times S^A.N) + \alpha$ where α is the complexity of computing μ. The value of α depends on the approach used to infer $\mu(\cdot,\cdot)$. As underlined in [5], inferring $\mu(\cdot,\cdot)$ by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines which use the fast Rete [11] 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 Algorithms 1 and 2 is $O(C.\eta \times S^A.N) + O(|R||F||P|)$. Furthermore, we observe, as in [5], that the process of computing μ is the most "expensive" step of the algorithms. Hence, we obtain: $O(C.\eta \times S^A.N) + O(|R||F||P|) \asymp O(|R||F||P|)$.

 ISBN: 978-1-61208-247-9

Algorithm 2: FunctionalDisjunctiveMatching

```
Input  : S^R, // requested service.
         S^A, // advertised requested.
         C, // criteria table.
Output: Boolean // success: true; fail: false.
while (i ≤ C.η) do
    while (j ≤ S^R.N) do
        if (S^R.A_j = C.A_i) then
            Append S^R.A_j to rAttrSet;
        end
        Assign j ⟵ j + 1;
    end
    while (k ≤ S^A.N) do
        if (S^A.A_k = C.A_i) then
            Append S^A.A_k to aAttrSet;
        end
        Assign k ⟵ k + 1;
    end
    Assign i ⟵ i + 1;
end
while (t < C.η) do
    if (μ(rAttrSet[t], aAttrSet[t]) ⪰ C.M_t) then
        return success;
    end
    Assign t ⟵ t + 1;
end
return fail;
```

IV. FUNCTIONAL 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. In order to define the sufficient functional service-level match, we need to introduce the concepts of aggregation rule and sufficient single attribute match.

A. Aggregation Rule

The computing of service-level similarity measure requires the definition of an aggregation rule to combine the similarity measures associated with attributes into a single measure relative to the service as a whole.

*Definition 10 (**Aggregation Rule**):* An aggregation rule ζ is a mean to combine the similarity measures into a single similarly measure. Mathematically,

$$\zeta : \mathcal{F}_1 \times \cdots \times \mathcal{F}_\eta \rightarrow \{\text{Exact, Plug-in, Subsumption, Container, Part-of, Disjoint}\}$$

where $\mathcal{F}_j$={Exact, Plug-in, Subsumption, Container, Part-of, Disjoint} ($j = 1, \cdots, \eta$); and η is the number of attributes included in the Criteria Table.$\Diamond$

The similarity maps given in Section III-A still apply here. Furthermore, the preference amongst similarity measures is governed by the same strict total order given in Definition 6.

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:

- **Minimum**: picks out the minimum similarity measure,
- **Maximum**: picks out the maximum similarity measure,
- **Median**: picks out the similarity measure corresponding to the median (in terms of order).
- **Floor**: picks out the similarity measure corresponding to the floor of the median values.
- **Ceil**: picks out the similarity measure corresponding to the ceil of the median values.

The Floor and Ceil rules apply only when there is an even number of similarity measures (which leads to two median values).

B. Sufficient Single Attribute Match

The sufficient single attribute match is defined as follows.

*Definition 11 (**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,

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

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

Algorithm 3: SuffSingleAttrMatching

```
Input  : S^R, // requested service.
         S^A, // advertised requested.
         C, // criteria table.
         A_i, // service attribute.
Output: Boolean // success: true; fail: false.
while (j ≤ S^R.N) do
    if (S^R.A_j = C.A_i) then
        Append S^R.A_j to rAttrSet;
    end
    Assign j ⟵ j + 1;
end
while (k ≤ S^A.N) do
    if (S^A.A_k = C.A_i) then
        Append S^A.A_k to aAttrSet;
    end
    Assign k ⟵ k + 1;
end
if (μ(rAttrSet[i], aAttrSet[i]) ⪰ C.M_i) then
    return success;
end
return fail;
```

C. Functional Service-Level Matchmaking Algorithm

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,

$$\begin{aligned}&[\forall i \quad (\text{SuffSingleAttrMatch}(S^R, S^A, A_i)) \quad 1 \leq i \leq C.\eta] \quad \wedge \\ &\quad [\exists j_1, \cdots, j_i, \cdots, j_\eta \quad (\zeta(s_{1,j_1}, \cdots, s_{i,j_i}, \cdots, s_{\eta,j_\eta}) \succeq \beta)] \\ &\Rightarrow \text{SuffFunctionalServiceLevelMatch}(S^R, S^A) \end{aligned} \quad (4)$$

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

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

The service-level matching is formalized in Algorithm 4. This algorithm follows directly from Sentence (4).

Algorithm 4: FunctionalServiceLevelMatching

```
Input  : S^R, // requested service.
         S^A, // advertised requested.
         C, // criteria table.
         ζ, // aggregation rule.
         β, // overall similarity threshold.
Output: Boolean // success: true; fail: false.
while (i ≤ C.η) do
    if (NOT(SuffSingleAttrMatch(S^R, S^A, A_i))) then
        return fail;
    end
end
while (i ≤ C.η) do
    while (j ≤ S^R.N) do
        if (S^R.A_j = C.A_i) then
            Append S^R.A_j to rAttrSet;
        end
        Assign j ⟵ j + 1;
    end
    while (k ≤ S^A.N) do
        if (S^A.A_k = C.A_i) then
            Append S^A.A_k to aAttrSet;
        end
        Assign k ⟵ k + 1;
    end
    Assign i ⟵ i + 1;
end
if
(ζ(μ(rAttrSet[1], aAttrSet[1]), ··· , μ(rAttrSet[C.η], aAttrSet[C.η])) ⪰ β)
then
    return success;
end
return fail;
```

D. Computational Complexity

The complexity of the two first *while* loops in Algorithm 3 is equal to $O(S^R.N) + O(S^A.N)$. Generally we have $S^A.N \gg S^R.N$, hence the complexity of the two first *while* loops in Algorithm 3 is equal to $O(S^A.N)$. Then, based on the discussion given in Section III-E, we may conclude that the worst case complexity of Algorithm 3 is $O(|R||F||P|)$, where R, F and P have the same definition as in Section III-E.

Based on the discussion in Section III-E, we can set that the complexity of the first *while* loop in Algorithm 4 is equal to $O(C.\eta \times (|R||F||P|))$ and the complexity of the second *while* loop is equal to $O(C.\eta \times S^A.N)$. Finding the minimum, maximum, median, floor and ceil of a list of n values are $O(n)$ worst-case linear time selection algorithms. Then, the overall complexity of Algorithm 4 is equal to $O(C.\eta \times (|R||F||P|)) + O(C.\eta \times S^A.N) + O(n)$. Then based on the discussion of Section III-E, the worst case complexity of Algorithm 4 is $O(|R||F||P|) + O(n)$.

V. Illustration and Experimental Results

A. Illustration

Let consider the travel agent scenario example given in Section II. Fig. 1 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 Fig. 1, 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*

Let now focalize on the process of matching *Query* outputs. The same reasoning applies to the process of matching the inputs. The matching algorithm 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 algorithm first looks to see if there is a match for *Hotel*. Based on Fig. 1, the match between *Hotel* and *Accommodation* will be flagged Exact. Then, the algorithm looks a match for *Excursion*. Based on Fig. 1, the match for *Excursion* with respect to *Accommodation*, *Sport* and *Cost* will fail (since there is no relationship between these concepts and *Excursion* concept). Based on the same reasoning, we obtain an Exact match for the input concept *Destination*.

Assume now that the Criteria Table is as in Table II. The desired similarity specified by the Criteria 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 II
Criteria Table

$C.A$	$C.M$
input	Exact
output	Exact

Suppose now that the *Advertisement* has the following Inputs, Outputs and Categories:

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

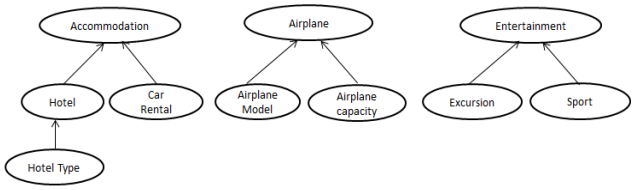

Figure 1. Travel Ontology

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 follows:

Table III
CRITERIA TABLE

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

The instantiation of the last instruction in Algorithm (4) will then look like ζ(Exact, Plug-in, Exact, Exact). Then, the result of Algorithm (4) according to different aggregation rules (except Median which does not apply here) is as follows:

Table IV
RESULT OF AGGREGATION

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

B. Implementation and Experimental Results

For the purpose of implementation, we used the same architecture proposed by [10]. We used the Protege editor [12] to browse and edit OWL [13] ontologies. The Mindswap OWL-S API [14] has been used to load the OWL ontologies into the Knowledge Base and parse the OWL-S [13] Queries and Advertisements. We used the Pellet reasoner [15] to classify the loaded ontologies and Jena API [16] to query the reasoner for concept relationships. To test the algorithms, we used 10 ontologies from the OWTLS-TC (service retrieval test collection from SemWebCentral) in our Knowledge Base. About 500 advertisements from OWLS-TC were loaded into the advertisement repository. Experiments were designed to measure execution time of the algorithms. All measurement points shown are average results taken from 50 runs. The data sets for clients and providers were randomly generated. Fig. 2 shows the execution time of all algorithms.

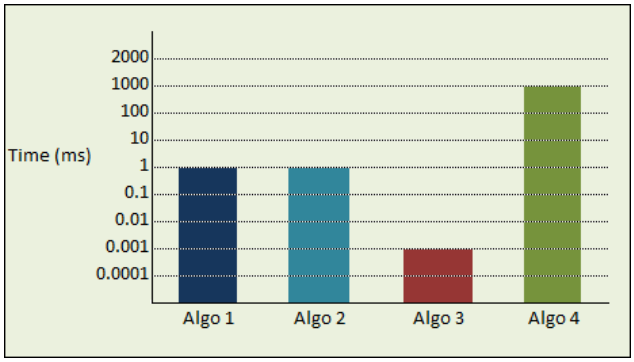

Figure 2. Execution time of algorithms

VI. RELATED WORK

Existing matchmaking frameworks such as [2] [3] [4] utilize a strict capability-based matchmaking, which is proven [5] to be inadequate in practice. Some recent proposals [6] [7] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only. Doshi et al. [5] present a parameterized semantic matchmaking framework that exhibits a customizable matchmaking behavior. This framework has three main merits. First, it uses semantic information in the matching process which has been proven in information retrieval [17], fuzzy database [18] and semantic web services [19] [20] [21] to be more useful than simple syntactic search. Second, it uses both functional and non-functional matchmaking. Third, by enabling the user to specify matchmaking criteria and conditions. One important shortcoming of [5] is that the sufficiency condition defined by the authors is very strict since it requires that all the specified conditions hold at the same time.

In [22], Fu et al. transform the problem of matching web services to the computation of semantic similarity between concepts in domain ontology. They developed a semantic distance measure to provide a quantitative similarity measures to support matching in semantic web services. Bellur and Kulkarni [10] improve [3]'s matchmaking algorithm and propose a greedy-based algorithm that relies on the concept of matching bipartite graphs. Their idea permits to avoid problems faced with the original [3]'s matchmaking algorithm.

In [9], we extended the work given in this paper by adding generic attribute-level and non-functional QoS matching. The non-functional matching permits to categorize web services into different ordered QoS classes. The user should then select one web service from the highest QoS class for implementation.

We think that the matchmaking framework presented in this paper extends and improves the works of [10] [5] [3]. Specifically, the proposed framework adds more customization, relaxes matchmaking conditions and supports three types of matching. An important addition of the proposed framework is the service-level matching which, to the best knowledge of the author, previous studies have not dealt with it.

 ISBN: 978-1-61208-247-9

VII. Conclusion and Future Work

In this paper, we presented a parameterized and highly customizable semantic matchmaking framework. A series of highly customizable algorithms is advised for both attribute-level and service-level matching. These algorithms are designed to support a customizable matching process that permits the user to control the attributes matched, the order in which attributes are compared, as well as the way the sufficiency is computed for both attribute and service levels. In this paper, we addressed functional attribute-level conjunctive matching, functional attribute-level disjunctive matching and functional service-level matching. We extended the work given in this paper in [9] by adding generic attribute-level and non-functional QoS matching.

The matchmaking framework proposed in this paper and its extended version given in [9] constitutes a basic component of a layered system, called QoSeBroker, for web service composition that we have proposed in [1]. In the future, we plan to conduce additional experimental tests in order to analyze the performance of the matching algorithms. We also intend to implement the subsequent components of QoSeBroker system.

References

[1] S. Chakhar, "QoS-enhanced broker for composite web service selection," in *Proc. of the 8th International Conference on Signal Image Technology & Internet Based Systems*, Sorrento - Naples, Italy, Nov. 2012, pp. 533–540.

[2] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic web technology," in *Proc. of the 12th International World Wide Web Conference*, Budapest, Hungary, May 1999, pp. 331–339.

[3] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of the First International Semantic Web Conference on The Semantic Web*, Sardinia, Italy, Jun. 1999, pp. 333–347.

[4] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa, "The retsina mas infrastructure," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1-2, pp. 29–48, 2003.

[5] P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder, "Parameterized semantic matchmaking for workflow composition," IBM Research Division, Tech. Rep. RC23133, Mar. 2004.

[6] S. Ben Mokhtar, A. Kaul, N. Georgantas, and I. Valérie, "Efficient semantic service discovery in pervasive computing environments," in *Proc. of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Melbourne, Australia, Nov. 2006, pp. 240–259.

[7] R. Guo, J. Le, and X. L. Xiao, "Capability matching of web services based on OWL-S," in *Proc. of the 16th International Workshop on Database and Expert Systems Applications*, Copenhagen, Denmark, Aug. 2005, pp. 653–657.

[8] H. Hao, H. Haas, and D. Orchard. (2004) Web services architecture usage scenarios. [Online]. Available: http://www.w3.org/TR/ws-arch-scenarios/ [retrieved: Nov., 2012]

[9] S. Chakhar, "QoS-aware parameterized semantic matchmaking for web service composition," in *Proc. of the 9th International Conference on Web Information Systems and Technologies*, Aachen, Germany, May 2013, unpublished.

[10] U. Bellur and R. Kulkarni, "Improved matchmaking algorithm for semantic web services based on bipartite graph matching," in *Proc. of the IEEE International Conference on Web Services*, Salt Lake City, Utah, USA, Jul. 2007, pp. 86–93.

[11] C. Forgy, "Rete: A fast algorithm for the many patterns/many objects match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, 1982.

[12] Protege: Ontology editor and knowledge-base framework. [Online]. Available: http://protege.stanford.edu/ [retrieved: Nov., 2012]

[13] G. Antoniou and F. van Harmelen, "Web ontology language: OWL," in *Handbook on Ontologies*, 2nd ed., S. Staab and R. Studer, Eds. Berlin / Heidelberg, Germany: Springer-Verlag, 2009, pp. 91–110.

[14] MINDSWAP: Maryland information and network dynamics lab semantic web agents project, OWL-S API. [Online]. Available: http://www.mindswap.org/2004/owl-s/api/ [retrieved: Nov., 2012]

[15] Pellet: An OWL DL reasoner. [Online]. Available: http://pellet.owldl.com/ [retrieved: Nov., 2012]

[16] Jena: Java framework for building semantic web. [Online]. Available: http://jena.sourceforge.net/ [retrieved: Nov., 2012]

[17] A. K. Kirykov and K. S. Iv, "Ontologically supported semantic matching," in *Proc. of NoDaLiDa'99: Nordic Conference on Computational Linguistics*, Trondheim, Norway, Dec. 1999, pp. 9–10.

[18] R. Bouaziz, S. Chakhar, V. Mousseau, S. Ram, and A. Telmoudi, "Database design and querying within the fuzzy semantic model," *Information Sciences*, vol. 177, no. 21, pp. 4598–4620, 2007.

[19] J. Cardoso, J. Miller, and S. Emani, "Web services discovery utilizing semantically annotated WSDL," in *Reasoning Web*, C. Baroglio, P. Bonatti, J. Maluszynski, M. Marchiori, A. Polleres, and S. Schaffert, Eds. Berlin / Heidelberg, Germany: Springer-Verlag, 2008, pp. 240–268.

[20] C. Mateos, M. Crasso, A. Zunino, and M. Campo, "Supporting ontology-based semantic matching of web services in movilog," in *Advances in Artificial Intelligence*, S. R. J. Sichman, H. Coelho, Ed. Berlin / Heidelberg, Germany: Springer-Verlag, 2006, pp. 390–399.

[21] P. V. C. E. Wu, C., "Latent semantic analysis - the dynamics of semantics web services discovery," in *Advances in Web Semantics I*, R. M. K. S. T. Dillon, E. Chang, Ed. Berlin / Heidelberg, Germany: Springer-Verlag, 2009, pp. 346–373.

[22] P. Fu, S. Liu, H. Yang, and L. Gu, "Matching algorithm of web services based on semantic distance," in *Proc. of the 2009 International Workshop on Information Security and Application*, Qingdao, China, Nov. 2009, pp. 465–468.

 ISBN: 978-1-61208-247-9