Functionality-Based Service Matchmaking for Service-Oriented Architecture

Stephen S. Yau and Junwei Liu
Arizona State University
Tempe, AZ 85287-8809, USA
{yau, junwei.liu}@asu.edu

Abstract

Service matchmaking is a basic feature of Service-Oriented Architecture (SOA). In this paper, a semantic-based flexible service matchmaking approach is presented to efficiently identifying functionality-compatible services. This approach utilizes SAW-OWL-S to specify the service advertisements and service discovery requests. The functionality-compatibility of a service to a request is defined on their parameters and conditions. This approach uses functionality filtering to prune out incompatible services, and then select services based on the aggregated similarities of input/output parameters, precondition/result situations and other service attributes. Experimental results are given to illustrate that this approach can efficiently generate precise service matchmaking results.

Keyword: Service-oriented architecture, service matchmaking, context and situation, service functionality, functionality-compatibility.

1. Introduction

Service-Oriented Architecture (SOA) [1] enables rapid composition of distributed applications from services in a flexible and agile manner, and has become the new-generation computing architecture for many large-scale distributed systems in various application areas, such as scientific collaboration, e-business, health care, military, and homeland security. A *service* is a well-defined and self-contained software entity with a discoverable and invocable interface to provide certain functionality over networks using standard protocols. Various services can interoperate with each other, regardless of the programming languages and platforms used. The independency and interoperability of services make SOA a suitable architecture for Autonomous Decentralized Systems. Various service providers, service requestors and service directories collaborate in SOA. Service providers publish service advertisements in the service directories; service directories handle service discovery requests by identifying suitable services matching the requests. Precise and efficient service matchmaking is a basic and important feature of SOA.

A good service matchmaking approach must find the service best matching the service request. There are two important issues involved in this approach: How to specify the functionality of the services semantically; and how to understand such semantic specifications and match services semantically. Traditional service matchmaking approaches are based on syntactical matching on the textual description of service names and properties, and hence they lack semantic considerations for supporting semantic-based service matchmaking. Although there are no available ways of interpreting the full semantics of services, there have been great improvements in semantic service specification languages, such as WSDL [2], OWL-S [3], and SAW-OWL-S [4]. With such improvements, there have been approaches for matching services based on the semantics of input and output parameters of the services. However, considering only the semantics of the input/output parameters is not sufficient to generate precise matching results due to the following reasons: First, for many services, the input and output parameters do not clearly represent the functionality provided by the service. For example, many services will return a Boolean value to indicate whether the service is executed successfully. In such a case, the output parameter cannot differentiate services with different functionalities. The semantics of services' results can help the differentiation in such a case. Secondly, the match of the input and output parameters does not guarantee the service can be successfully used by the requestor. In dynamic SOA-based systems, the execution of a service that meets the service requestor's expectation must also satisfy certain preconditions and provide the expected results.

To solve these two problems, a service matchmaking approach based on service functionalities, called *F-Match,* will be presented in this paper. F-

Match utilizes SAW-OWL-S [4], which extends OWL-S with situation ontology [5], as the semantic specification language for services. SAW-OWL-S extends OWL-S with the semantic-based preconditions and postconditions specification using situations. A *situation* is a set of contexts in the application over a period of time that affects future system behavior. A *context* is any instantaneous, detectable, and relevant property of the environment, the system, or users, such as location, available bandwidth and a user's schedule [6, 7]. Our approach will provide precise and efficient service matchmaking based on service parameters, conditions and attributes. The experimental results on evaluating the efficiency and matching accuracy of our approach will also be presented

2. Current State of the Art

Many Service Discovery Protocols (SDPs) have been developed. Among them, Jini [8], SLP (Service Location Protocol) [9], Salutation [10], UPnP (Universal Plug and Play) [11], and Bluetooth [12] are five main approaches. Various service matchmaking approaches are supported in these SDPs. In Jini, service matchmaking is based on the interface type of services written in Java or the specific value of service attributes. SLP supports service matchmaking on service attributes and service types. Salutation does not support service attributes based matchmaking directly. In UPnP, the XML service descriptions are used. Bluetooth also supports service matchmaking based on service types and attributes. All these approaches are based on syntactical level matching of the text description of service interfaces, types or attributes.

The Universal Description, Discovery and Integration (UDDI) project [13] is an industry initiative of service discovery, and supports syntactical keyword matching of the name of the businesses, Web services and TModels. Paolucci et al. [14] extended the UDDI with the semantic matching of Web services based on the subsumption relations between their capabilities specified by ontology concepts of input/output parameters using DAML-S. Sirin et al. [15] presented the input/output parameter-based service filtering and selection based on OWL-S specification. The overall structure of OWL-S [3] (formally DAML-S) includes three main parts: the service profile for advertising and discovering services; the process model, which gives a detailed description of a service's operation, including the IOPE (Input, Output, Precondition, and Effect) parameters of the process; and the grounding, which provides the details on how to interoperate with a service, via messages. These two semantic matching approaches [14, 15] are solely based on the concept taxonomy of input/output parameters of services, and do not differentiate the matching services with different precondition/postcondition specifications as mentioned before. In our approach, we include the consideration of these service conditions during matchmaking by incorporating service functionality filtering and matching based on both the input/output parameters and precondition/result situations.

3. Requirements for Service Matchmaking

SOA involves a large number of services with various functionalities, which can be discovered and used to compose distributed applications dynamically. In order to select services that can be successfully used to achieve the requestor's goal; the underlying service matchmaking mechanism must satisfy the following requirements:

- Semantic matchmaking

Traditional syntactical service matchmaking approaches encounter serious difficulties in service adequate environment, especially when multiple parties are involved. This is due to the absence of a unified understanding of the semantics carried by the syntactical service specifications among multiple parties. For example, although the words "car" and "vehicle" have similar semantics, they are totally different at syntactical-level. On the other hand, the word "paper" may be used to describe both a service to browse scientific research papers and a service to sell printing paper. With the development of semantic service specification languages, it is possible for multiple parties in SOA to share and exchange semantics of services with a unified common ground. The service matchmaking mechanism should try to maximally explore the semantics carried in service specifications and select services based on the unified semantics.

- Flexible matchmaking

Basically, the service matchmaking mechanism answers the question, "Whether this service advertisement matches this service discovery request?" In a service adequate environment, the binary "match" and "not match" results will not work because there will be a very good chance that multiple services will provide similar functionalities that all these services "match" the request. Instead of returning all these services as the generated results and burdening service requestors to select the most suitable service, the service matchmaking mechanism should further distinguish similar services on how well the services match the request.

- Functionality matchmaking

Although it is important for the service matchmaking engine to provide a flexible matching schema, there should still be some "hard" guidelines for selecting services during the matchmaking process. The top priority guideline should be “The discovered service must be usable to the service requestor”, which means its functionality should be compatible with the requests. The service matchmaking mechanism must be able to categorize the services into “compatible” and “incompatible” classes for each service discovery request based on their functionalities and only return the compatible services as the results.

- Efficient matching

Efficiency is always an important requirement in SOA. The service matchmaking mechanism must not cause too much computation and communication burden on either the service directories or the service requestors. The service matchmaking mechanism should be computationally efficient running on the service directories and avoid adding complicated communication interactions.

4. Service Specification for F-Match

The capability of a service matchmaking approach is limited by the underlying service specification languages. Since there are no good ways to interpret the full semantics of services, what type of the information of the service is specified and how these types of information are specified should be carefully considered. In this paper, we will use SAW-OWL-S as the service specification language for F-Match. By integrating the situation ontology with OWL-S, the SAW-OWL-S provides specification of the related contextual and the precondition/result situations of a service. Although the precondition/result-based matching of software component has been presented [16], few service matchmaking approaches have considered such conditions of services. The use of situations to specify precondition/result of services provides a feasible approach to incorporating conditions in service matchmaking.

For the sake of completeness, we will give a brief introduction of situation ontology and SAW-OWL-S before presenting the semantic specification of service advertisements and service discovery requests in F-Match. For detailed information of situation ontology and SAW-OWL-S, the reader is referred to [4, 5].

4.1 Situation Ontology and SAW-OWL-S [4, 5]

Situation ontology models context and situation in a hierarchical way such that the definitions for context and situation can be easily shared and reused. Situation ontology is extensible to user-defined domain specific situation specifications. In situation ontology, an Entity in the system may satisfy or notSatisfy a situation; the satisfaction of a situation may imply satisfaction of another situation, which can be inferred based on their semantic specifications.

SAW-OWL-S models four main relations between context/situation and service by integrating situation ontology into OWL-S: the service contextual data, the situation precondition, the situation result, and the situation-service-triggering. By defining service preconditions and results using situations, SAW-OWL-S extends the expressiveness of OWL-S to support contextual/situational preconditions and results. Figure 1 shows the key classes and relations in SAW-OWL-S.

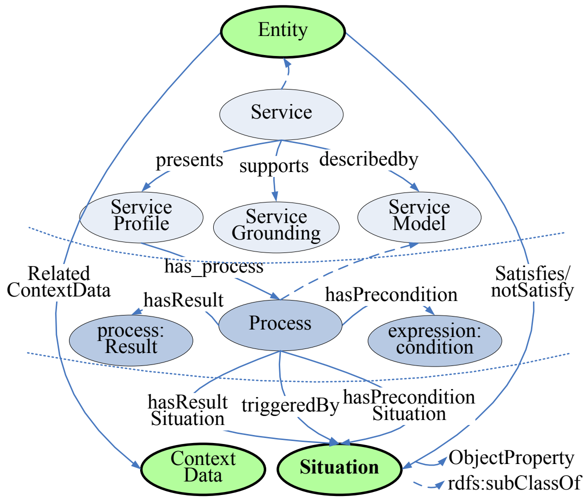


Figure 1. SAW-OWL-S (only related important classes and relations are shown)

4.2 Service Advertisement and Request Specification for F-Match

The SAW-OWL-S is utilized in F-Match to specify the provided and required service functionality for both service advertisements and requests. The semantics carried in service specifications is reorganized into three categories in F-Match: the input/output parameters; the precondition/result conditions and other service attributes. In F-Match, the input/output parameters are represented by ontology concepts; the precondition/result conditions are specified using situations and the service descriptions include all data type attributes of the services. The following is the reorganized service specification in F-Match.

$$S_{service} = \{S_{parameter}, S_{condition}, S_{description}\}$$
$$S_{parameter} = \{P_{input}, P_{output}\},$$
$$where\ P_{input} = \{in_1, in_2, ...\}, P_{output} = \{out_1, out_2, ...\}$$
$$S_{condition} = \{C_{pre}, C_{post}\},$$
$$where\ C_{pre} = \{pre_1, pre_2, ...\}, C_{post} = \{post_1, post_2, ...\}$$
$$S_{description} = \{D_{AnyURL}, D_{String}, D_{Double}, D_{float}, ...\}$$

5. Matchmaking Algorithm of F-Match

The basic strategy behind F-Match is to prune incompatible services and sort compatible services, based on the following two criteria:

C1. Guarantee that the functionality of the discovered service is compatible with the request.

C2. Rank all the compatible services according to the similarity to the request and return the most similar ones as matchmaking result.

Figure 2 shows the matchmaking algorithm of F-Match. To process a new service discovery request, F-Match will first filter out any service advertisements which are not functionality-compatible to the request. Then, F-Match will calculate and aggregate the similarities of parameters, conditions and attributes between each functionality-compatible service advertisement and the request; sort all the remaining service advertisements based on the final similarities and return the top ranked service advertisement as the matchmaking result.

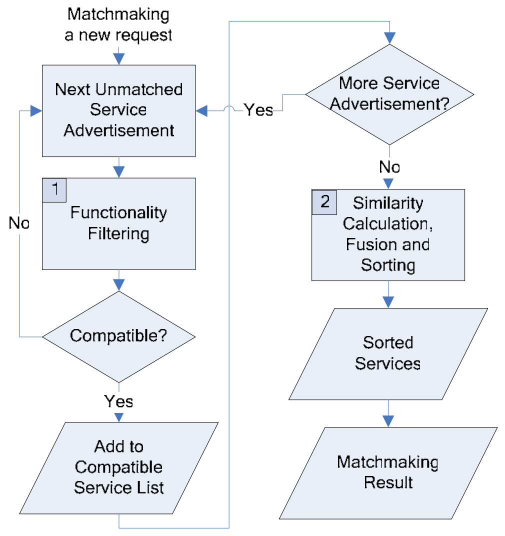
5.1 Functionality Filtering

The objective of service discovery is to find a service that can be executed by the requestor. In the best scenario, there is a service which is exactly the same as the request. However, in SOA, with numerous combinations of service attributes, input/output parameters and precondition/result conditions, the chance of having such a perfect match is very small. Instead of trying to find a perfect match, a more realistic objective is to find a service which is "similar enough" to the request. The functionality of the discovered service should be at least compatible to the request. For example, a user wants to find a service which lists available used cars within a certain price range. A service that lists used SUVs within a price range is a positive discovery result, whereas a service that lists used boats within a price range should be filtered out.

In SOA, service discovery involves communication iterations between service requesters and service directories. In case the discovered service is not usable to the requester, there will be large computation and communication overhead to redo the service discovery process, and it should be avoided. To achieve this goal, the first step in F-Match is to filter out service advertisements which are not functionality-compatible to the request by performing functionality filtering. The *functionality-compatibility* of a service to a service discovery request is defined as follows:

Definition 1. Given a service discovery request *R* and a service *S*, *S* is *functionality-compatible* to *R* if *S* is both *parameter-compatible* and *condition-compatible* to *R*.

We will define *parameter-compatibility* and *condition-compatibility* in the following subsections.



Matchmaking a new request
Next Unmatched Service Advertisement
1 Functionality Filtering
Compatible?
No
Yes
Add to Compatible Service List
More Service Advertisement?
Yes
No
2 Similarity Calculation, Fusion and Sorting
Sorted Services
Matchmaking Result

Figure 2. The matchmaking algorithm of F-Match

5.1.1 Parameter-compatibility

For an output parameter of service advertisement *OutS* and an output parameter of service request *OutR*, there are the following four relations [17]:

-Exact Match. Either *OutS* is same as *OutR* or *OutS* is a subclass of *OutR*.

-Plug-in. *OutS* subsumes *OutR*

-Subsume. *OutS* is indirectly subclass of *OutR*

-Fail. No subsumption relation between *OutS* and *OutR*

For the first three relations, there exist a subsumption relation between *OutS* and *OutR,* and we consider *OutS* and *OutR* are compatible with each other in F-Match. Similarly, if there exists a subsumption relation between two input parameters, we consider they are compatible.

In most cases, a service discovery request may specify multiple input and output parameters. In F-Match, we define parameter-compatibility of a service to a service discovery request as follows:

Definition 2. Given a service discovery request *R* and a service *S*, *S* is *parameter-compatible* to *R* if for any output of *R*, *S* has a corresponding compatible output; and for any input of *S*, *R* has a corresponding compatible input. Two parameters p_1 and p_2 *are compatible* to each other if there exists a subsumption relation between p_1 and p_2.

F-Match checks the parameter-compatibility of a service advertisement against a service discovery request to filter out all the service advertisements which do not have compatible parameters.

5.1.2 Condition-compatibility

For precondition/result conditions, we adopt the software component conditional matching level categorized in [16] to define condition-compatibility:

Definition 3. Given a service discovery request *R,* a service *S*, and that *preS, postS, preR, postR* represent the precondition of *S*, the result of *S*, the precondition of *R* and the result of *R,* respectively, *S* is *condition-compatible* to *R* if there exists a combination of *preS, postS, preR, postR* such that one of the following relations holds.

-Equal: $(pre_R \Leftrightarrow pre_S) \wedge (post_S \Leftrightarrow post_R)$

-Plug-in: $(pre_R \Rightarrow pre_S) \wedge (post_S \Rightarrow post_R)$

-Plug-in post: $post_S \Rightarrow post_R$

F-Match analyzes the relations among precondition/result situations based on the value of "implies" properties of situations derived from their specifications and logical compositions to prune out service advertisements without compatible conditions.

5.1.3 Functionality-compatible filtering

It is noted that the definition of functionality-compatibility is based on parameter-compatibility and condition-compatibility. There are other factors affecting the successful execution of the returned services, such as the internal process and the required Quality of Service. In this paper, we do not address the service matchmaking based on such factors.

5.2 Similarity Calculation and Fusion

The measurement of the similarity between a service advertisement and a service discovery request involves various types of service specifications as categorized in Section 4.2. F-Match utilizes different algorithms to calculate the similarities of parameters, conditions and attributes; and aggregate these similarities to an overall similarity. In the following subsections, we will present the calculation of these similarities.

5.2.1 Similarity of input/output parameters

The similarity of parameters (represented as ontology concepts) is usually measured based on their relative positions in a concept tree. The similarity has discrete values [14] based on the categorization as summarized in Section 5.1. However, their approach has two limitations. First, the discrete similarity calculation algorithm in their approach does not reflect the levels of direct subsumption relations between two parameters. For example, for a service discovery request with output of "SUV", two services with outputs "Thing" and "Car" have the same similarities as "Plug In", and hence these two totally different services are not differentiated. Second, the number of subclasses of concepts is not considered during their similarity calculation. In a particular concept tree, if a concept has more children concepts, the similarity between this concept and each of its children concepts should be smaller because each additional child concept inherits an additional part of the semantics of the parent concept. To overcome these two limitations, with the similarity calculation in F-Match, if two concepts are the same, then they have the maximum similarity 1. The more intermediate concepts between the two concepts, the smaller the similarity will be. And the more subclasses of the intermediate concepts between two concepts, the smaller the similarity will be. If there exists no subsumption relations between two concepts, then they have the minimum similarity 0. The similarity calculation is given as follows:

Given a concept subsumption tree *T,* in which a parent concept directly subsumes its children concepts, with root *R* and two particular concepts C_1 and C_2. Without losing generalization, assume that C_1 has at least the same depth-level as C_2, the similarity between C_1 and C_2 can be calculated as follows:

$$S(C_1,C_2)=\begin{cases} 1, & \text{if } C_1=C_2; \\ \prod\limits_{i=1,\dots n}\dfrac{1}{S_{a_i}}, & \text{if there exists a path } C_1,C_{a_1},\dots,C_{a_n},C_2,\dots,R, \\ & \text{where } S_{a_i} \text{=number of the subclasses of } C_{a_i}; \\ 0, & \text{if such a path does not exist and } C_1 \neq C_2. \end{cases}$$

Unlike the approach in [14], the input parameters and output parameters are not differentiated in F-Match during similarity calculation. In case multiple input/output parameters are specified, the similarity of the input/output parameters between a service advertisement and a service discovery request is aggregated by finding the one-to-one mapping between the parameters of the service advertisement and request which has the largest average similarities. This problem can be formulated as follows:

Given a request R (input $inR_1,\dots,inR_m$, output $outR_1,\dots,outR_n$), and a service S (input $inS_1,\dots,inS_p$, output $outS_1,\dots,outS_q$):

$$PS(S,R)=(PS_{input}(S,R)+PS_{output}(S,R))/2,$$

where $PS_{input}(S,R)=Max(\underset{i=1,\dots p}{Sum}(Similarity(inS_i,inR_{j_i}))/p)$,

$1\le j_i\le m, and\ j_{i_1}\neq j_{i_2}$ for $i_1\neq i_2$;

and $PS_{output}(S,R)=Max(\underset{i=1,\dots n}{Sum}(Similarity(outR_i,outS_{j_i}))/n)$,

$1\le j_i\le q, and\ j_{i_1}\neq j_{i_2}$ for $i_1\neq i_2$.

It is noted that a perfect matched service will have the largest parameter similarity 1, and any parameter incompatible service will have the smallest parameter similarity 0. The aggregation method will ensure that a service with the parameter similarity 1 will have its parameters in one-to-one equivalent to those of the service discovery request although the order of the parameters may be different.

5.2.2 Similarity of precondition/result conditions

In F-Match, the similarities of precondition/result conditions between service advertisements and service discovery requests has four different discrete levels given in Table 1 based on the categorization of relations in Section 5.1. The range of the similarity values of these categories is [0, 1], with 1 for exact match and 0 for no match or incompatible.

Table 1. Condition similarity values

Relation	*CS(S,R)*
Exact match	1.0
Plug-in	2/3
Plug-in post	1/3
Fail	0

5.2.3 Similarity of data type service attributes

In this section, we will briefly discuss the similarity calculation for three major types of data type service attributes.

-*String attribute*: The traditional string similarity algorithm, such as the longest substring algorithm or the Soundex algorithm will be used;

-*Integer/Float attribute*: The similarity is calculated by the ratio between two values.

-*AnyURI attribute*: Each URI refers to a concept and utilizes the concept similarity measurement in [17] to calculate the similarity.

After calculating the similarity of each data type service attributes, we aggregate and normalize the attribute similarity into the range [0, 1].

5.2.4 Similarity fusion and sorting

The parameter similarity (*PS*), the condition similarity (*CS*) and the attribute similarity (*AS*) are fused into a unified and normalized overall similarity with different weights by the following formula:

$$Sim(S,R) = W_{PS} * PS(S,R) + W_{CS} * CS(S,R) + W_{AS} * AS(S,R),$$
$$\text{where } W_{PS}, W_{CS}, W_{AS} \geq 0 \text{ and } W_{PS} + W_{CS} + W_{AS} = 1$$

Users can assign these weights in service discovery request to reflect different user preferences. F-Match then sorts all the functionality-compatible service advertisements based on their overall similarities from high to low. The top ranked service is returned as result. According to our similarities calculation, the overall similarity will be the highest value 1 if and only if the parameters, conditions, and attributes of the service advertisement all perfectly match the service discovery request.

6. Experiments and Results

In this section, we will discuss our experiments and results for evaluating the performance of F-Match in terms of average request processing time and accuracy of matchmaking results. We have implemented F-Match in Java, and used the Jena2 toolkit [18] to generate, parse and reason about OWL-based specifications. The hardware used in our experiments is a PC with 3GHZ CPU and 2GB memory.

In our experiments, we first defined a hierarchical concept tree as parameter set and define three sets of situations as condition sets. Then, we generated five experiment sets with 20, 40, 60, 80 and 100 service advertisements and service discovery requests. Each SAW-OWL-S service specification in service advertisement or service discovery requests includes input/output parameters selected from the parameter set, precondition/result situations selected from one of the condition set and a set of service attributes. A segment of OWL specification of input/output parameters and precondition/result situations of one generated service advertisement is shown in Figure 3.

```
<rdf:RDF
  ……
   <j.6:DataService rdf:ID="dataService17">
     <j.11:describedBy>
       <j.5:Process rdf:ID="process17">
         <j.5:hasOutput rdf:resource="PS.owl#para-1(2)-1(2)"/>
         <j.5:hasInput rdf:resource="PS.owl#para-2(2)-1(2)-1(2)"/>
         <j.8:hasResultSituation>
           <j.9:AtomicSituation rdf:about="CS.owl#aS-3-3"/>
         </j.8:hasResultSituation>
         <j.8:hasPreconditionSituation>
           <j.9:ConjunctionSituation rdf:about="CS.owl#cS-2-2-2"/>
         </j.8:hasPreconditionSituation>
       </j.5:Process>
     </j.11:describedBy>
      ……
   </j.6:DataService>
</rdf:RDF>
```

Figure 3. A Sample SAW-OWL-S specification

To evaluate the effectiveness of our functionality filter, especially the effectiveness of condition-compatibility filtering, three different condition sets shown in Table 2 were used in our experiments. For each condition set, besides the number of different situations defined in the set, we also used the *Implication Ratio* to measure the density of the implication relations among the situations in the set. The *Implication Ratio* is defined as follows:

Definition 4. Given a condition set *CS* with *n* different conditions, the *implication ratio IR* of *CS* is defined as:

$$IR(CS) = \sum_{s_1 \in CS} \sum_{s_2 \in CS} implies(s_1, s_2) / n_2,$$

$$where\ implies(s_1, s_2) = \begin{cases} 1, if\ s_1\ implies\ s_2 \\ 0,\ otherwise \end{cases}$$

Table 2. The condition sets used in experiments

	Number of Situations	Implication Ratio
Small	16	14.84%
Medium	60	7.73%
Large	115	2.38%

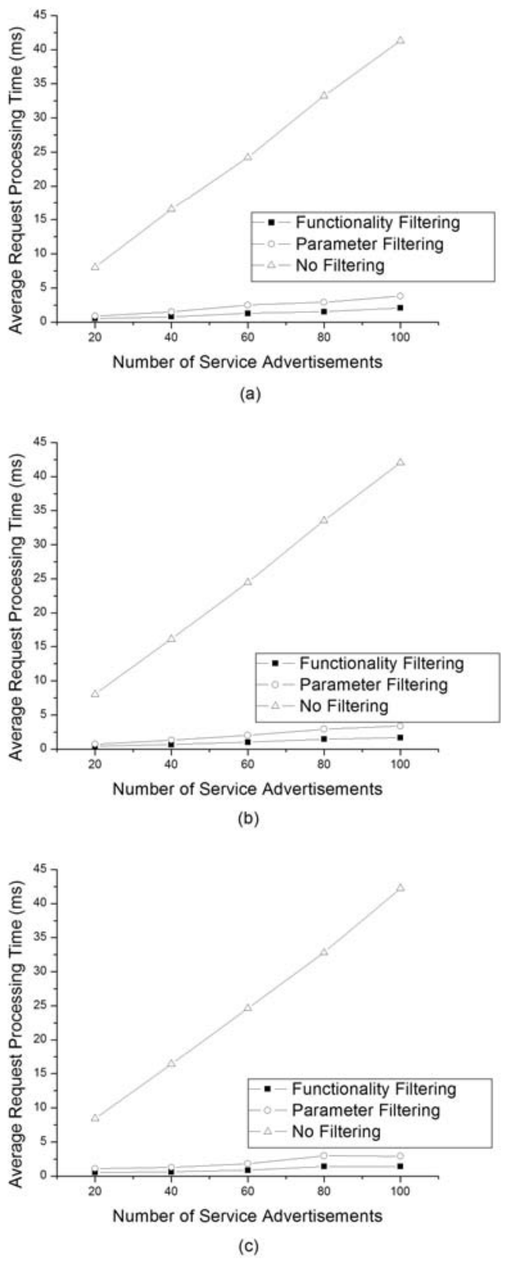


Figure 4. Average request processing time comparison for different filtering strategies. (a) Small condition set; (b) medium condition set; and (c) large condition set.

We first evaluated the effectiveness of functionality filter by comparing the average request processing time of three types of filtering strategies: functionality filtering as utilized in F-Match, parameter filtering similar to that in [14] and no filtering. Figure 4 shows the experimental results. From the experimental results, we observe that for all the three condition sets, F-Match requires the least average request processing time. The parameter filtering also performs quite well, whereas the matchmaking not considering compatibility filtering has poor performance. The experimental results validate that the functionality filtering in F-Match improves the efficiency of matchmaking.

Second, we evaluated the effectiveness of F-Match in term of the accuracy of the matchmaking results. Using functionality filter, F-Match guarantees that the matchmaking results are always functionality-compatible to the service discovery requests. We compare the hit rate and miss rate for the three types of filtering: functionality filtering, parameter filtering, and no filtering. The hit rate is defined as the ratio of the number of returning functionality-compatible services and the number of requests processed in cases there exists a functionality-compatible service. The miss rate is defined as the ratio of the number of functionality-incompatible services and the number of all the returned services. The results are shown in Tables 3 and 4.

Table 3. Hit rate comparison

Condition Set	Functionality Filtering	Parameter Filtering	No Filtering
Small	1.0	0.5873	0.4206
Medium	1.0	0.4394	0.4242
Large	1.0	0.5900	0.4090

Table 4. Miss rate comparison

Condition Set	Functionality Filtering	Parameter Filtering	No Filtering
Small	0.0	0.7233	0.8233
Medium	0.0	0.8700	0.9067
Large	0.0	0.9400	0.9700

According to the experimental results, we observe that the parameter filtering and no filtering are largely error prone, especially for the large condition set with a small implication ratio. On the other hand, F-Match always returns functionality-compatible services and never returns functionality-incompatible services as results.

7. Conclusion and Future Work

In this paper, a service matchmaking approach based on service functionality, called F-Match, is presented. F-Match utilizes SAW-OWL-S to specify service advertisements and service discovery requests, in which situations are used to specify the precondition/result conditions of services. F-Match begins with functionality filtering to prune out functionality-incompatible services, and then ranks functionality-compatible services based on the fusion of similarities against the input/output parameters, precondition/result situations and other data type service attributes of the request. F-Match provides a semantic-based flexible matchmaking approach to efficiently identifying functionality-compatible services. Our experimental results validate this conclusion.

Further research in this area is needed to improve the matchmaking for SOA. For example, a more general algorithm needs to be developed to inference about the implication relation among various situations. We also need to investigate the possible condition-compatibility analysis based on SWRL condition specification supported in OWL-S. We also need to improve F-Match to support matchmaking of service related properties other than functionalities and data type attributes, such as the process models, the related contextual data and the Quality of Service of services.

Acknowledgment

The work reported here was supported by the National Science Foundation under grant number ITR-CYBERTRUST 0430565.

References:

[1] W3C, "Web Services Architecture," *http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.*

[2] WSDL, "Web Services Description Language", available at: http://www.w3.org/TR/wsdl.

[3] "OWL-S: Semantic Markup for Web Services", available at: http://www.w3.org/Submission/OWL-S/.

[4] S.S. Yau and J. Liu, "Incorporating Situation Awareness in Service Specifications", *Proc. 9th IEEE Int'l Symp. on Object and Component-Oriented Real-Time Distributed Computing (ISORC 06),* April 2006, pp.287-294.

[5] S. S. Yau and J. Liu, "Hierarchical Situation Modeling and Reasoning for Pervasive Computing", *Proc. 4th Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 06)*, April, 2006, pp.5-10.

[6] S. S. Yau, Y. Wang, and F. Karim, "Development of Situation-Aware Application Software for Ubiquitous Computing Environments", *Proc. 26th Ann. Int'l Computer Software and Applications Conf. (COMPSAC 2002)*, 2002, pp. 233-238.

[7] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing", IEEE Pervasive Computing, 1(3), July-September 2002, pp.33-40.

[8] Sun Microsystems, "Jini Architecture Specification", V1.2, December 2001, available at: *http://www.sun.com/software/jini/specs/jini1_2.pdf.*

[9] E. Guttman, C. Perkins, J. Veizades and M. Day, "Service Location Protocol", V2, June 1999, available at *http://www.faqs.org/ftp/rfc/pdf/rfc2608.txt.pdf.*

[10] The Salutation Consortium, "Salutation Architecture Specification (Part 1)", V2.0c, June 1999, available at: *http://www.salutation.org/.*

[11] Microsoft Corporation, "Understanding Universal Plug and Play White Paper", June 2000, available at *http://www.upnp.org/download/UPNP_Understanding UPNP.doc.*

[12] Bluetooth, "Specification of the Bluetooth system", V1.0B, December 1999, available at *http://grouper.ieee.org/groups/802/15/Bluetooth/core_10_b.pdf.*

[13] Universal Description Discovery and Integration Platform", September 2000, available at *http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.*

[14] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities", *Proc. 1st Int'l Semantic Web Conf. (ISWC2002),* 2002, pp. 333-347.

[15] E. Sirin, B. Parsia, and J. Hendler, "Filtering and selecting semantic Web services with interactive composition techniques", *Intelligent Systems*, *IEEE*, Volume 19, Issue 4, Jul-Aug 2004 Page(s):42 - 49

[16] A. Moormann Zaremski, and J. M. Wing, "Specification matching of Software Component", *ACM Trans. on Software Engineering and Methodology*, 1997, pp. 333-369.

[17] J. Zhong, H. Zhu, J. Li, and Y. Yu, "Conceptual graph matching for semantic search," *Proc. 2002 Int'l Conf. on Computational Science (ICCS2002),* April, 2002, pp.92-106.

[18] Jena2 Semantic Web Toolkit. Available at *http://www.hpl.hp.com/semweb/jena2.htm.*