July 2006, Vol.21, No.4, pp.574–581 J. Comput. Sci. & Technol.

A Semantic Matchmaker for Ranking Web Services

Bin Xu (许　斌), Po Zhang (张　钋), Juan-Zi Li (李涓子), and Wen-Jun Yang (杨文军)

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P.R. China

E-mail: xubin@tsinghua.edu.cn

Received February 20, 2006; revised May 8, 2006.

Abstract This paper is concerned with the matchmaker for ranking web services by using semantics. So far several methods of semantic matchmaker have been proposed. Most of them, however, focus on classifying the services into predefined categories rather than providing a ranking result. In this paper, a new method of semantic matchmaker is proposed for ranking web services. It is proposed to use the semantic distance for estimating the matching degree between a service and a user request. Four types of semantic distances are defined and four algorithms are implemented respectively to calculate them. Experimental results show that the proposed semantic matchmaker significantly outperforms the keyword-based baseline method.

Keywords semantic web services, semantic distance, services ranking

1 Introduction

Web services[1] are self-contained, self-describing, modular applications that can be published, located and invoked across the web. Supporting standards like UDDI[2], WSDL[3] and SOAP[4] make web services widely adopted by many Internet applications, such as peer-to-peer[5−7]. So far, service discovery is becoming one of the challenges in web services. The goal of service discovery is to find feasible services accurately and efficiently according to the user request.

By using UDDI and WSDL, web service discovery is conducted by keyword matching on the properties of the services (e.g., name, key, and category, etc.). This method cannot achieve high accuracy.

Semantic web service technology is aimed at improving the accuracy of service discovery by providing semantics for web services. The discovery of semantic web services is usually done by using semantic matchmaker. The matchmaker evaluates the matching degree between a request and the services.

Several approaches for semantic matchmaker have been proposed. For example, Paolucci *et al.*[8] proposed a basic matchmaker to match the capability of services. Based on the hierarchical relation of concepts in input and output, matched web services are classified into several categories: exact, subsume, plugin and fail. This approach, however, is not sufficient to distinguish two web services which are classified into the same category. For instance, it is difficult to say that service A is more related to the request than service B when they are all in *subsume* category. The category classification is rough for a requestor to choose an optimal service.

This paper proposes a semantic matchmaker to rank web services according to the semantic information in service description. Two challenges arise in the matchmaker: 1) how to propose a principled approach for measuring the semantic distance between concepts in the ontology; 2) how to estimate the matching degree of the service to the request by making use of the semantic distances. A baseline matchmaker based on keyword match is firstly proposed to quantify the matching degree of service. Semantic distance is a quantitative value in this paper to measure the distance between concepts in the ontology. Four types of semantic distances are defined and four algorithms are implemented respectively to calculate them. We propose the semantic matchmaker algorithm based on semantic distance, similarity function and output weight. We performed experiments for evaluating the proposed matchmaker. Web services descriptions of WSDL are collected and transformed into semantic descriptions of OWL-S[9] as test data. Experimental results show that the proposed semantic matchmaker algorithm has better performance than the baseline matchmaker algorithm.

The rest of this paper is organized as follows. Section 2 presents the related work. In Section 3, we propose a baseline matchmaker. In Section 4, we present the definition of semantic distance. In Section 5, we propose a semantic matchmaker algorithm using the idea of semantic distance. In Section 6, we provide the experimental results. Finally Section 7 gives the concluding remarks.

2 Related Work

Recently, many research efforts have been made on providing semantics for web services. Sivashanmugan *et al.*[10] provided a semantic extension approach for WSDL by mapping elements in WSDL to concepts of ontology. Ogbuji[11] substituted the XML in WSDL as RDF to add semantics. Peer[12] used MDL (Meaning Definition Language)[13] as a bridge to link WSDL and semantics together. Sriharee *et al.*[14] added the ontology-based behavior information to WSDL. Sirin *et al.*[15] proposed a manual tool to translate WSDL into

Regular Paper

This work is supported by the National Natural Science Foundation of China under Grant No. 90604025.

OWL-S, in which the user must map ontology class to WSDL schema type manually. Patil *et al.*[16] mapped the XML Schema in WSDL into concepts in ontology, and transformed WSDL to WSDL-S semi-automatically. Unfortunately, none of them are automatic, and the efficiency of getting semantic description is not high.

Several matchmaker algorithms have also been proposed. Paolucci *et al.*[8] proposed the basic matchmaker to discover web services in four categories as described in Section 1. Bansal *et al.*[17] separated the Composite Process in Process Model into an atomic process tree, and then matched the leaf atomic process recursively. Klein *et al.*[18] used process ontology in matchmaker to match the Process Model. Jaeger *et al.*[19] proposed a ranked matchmaker with more categories according to the inputs, outputs and the service categories. Verma *et al.*[20] adopted service template to match services of WSDL-S. Bentallah *et al.*[21] used description logic (DL) in service matchmaking. Gonzalez-Castillo *et al.*[22] used DAML+OIL as semantic description of web service and matched services based on DL. Caragea *et al.*[23] used a semantic schema matching of the I/O descriptions of services. Syeda-Mahmood *et al.*[24] used domain-independent and domain-specific ontologies to find matching service descriptions. Although so many matchmaker algorithms are proposed, they cannot give quantitative rank of services according to the user request.

3 Baseline Matchmaker

In this section, we give a baseline matchmaker to quantify the matching degree between a service and a user request. The matchmaker gives each service a quantitatively ranked value to represent the matching degree, and compares those ranked values to find a better service. This makes requestor much easier to identify services.

For instance, if a requestor wants to choose an optimal service from a large set of services that might meet the requirements, the basic matchmaker[8] returns the services of several categories. But if matchmaker provides ranked results instead of the conclusion that all services are *subsume*, the optimal service is given by the ranking.

Here we explain how to rank services using the baseline matchmaker. Suppose that a requestor wants to find a web service about weather forecasting shown in Fig.1.

```
⟨profile xmlns=http://keg.cs.tsinghua.edu.cn/
                    ontology/travel#···⟩
...
⟨profile:hasInput rdf:resource="#Date"/⟩
⟨profile:hasInput rdf:resource="#City"/⟩
⟨profile:hasOutput rdf:resource="#Weather"/⟩
...
```

Fig.1. Request for weather forecasting service.

This request is an OWL-S Profile. The inputs are *Date* and *City*, and the output is *Weather*. *Date*, *City* and *Weather* are classes in the travel ontology[25]. It means that the requestor wants to find a web service which can provide weather information of specific date and city. The baseline matchmaker proposed in this paper uses all the web service descriptions in repository to match the request. The match results are shown in Fig.2.

```
Web service 1 ranked value = 1.0
Web service 2 ranked value = 0.88
Web service 3 ranked value = 0.67
Web service 4 ranked value = 0.33
Web service 5 ranked value = 0
...
```

Fig.2. Match results.

Every ranked value in Fig.2 represents the matching degree between the service and the request. These services are ranked in a descendent order according to the ranked values and the top-ranked services can be chosen as the best services.

```
Input: request_inputs[ ], request_outputs[ ], services[ ];
Output: rank[ ];
{
Step 1: get each service's rank value.
for each service[i] in service [ ]{
  rank[i] = 0;    //initial rank value of service[i]
  for each atomicProcess[j] in services[i] {
    tempRank[j] = 0;
    for each service_inputs[k] in atomicProcess[j]
    {for each request_inputs[m] {
    if equals(service_inputs[k], request_inputs[m])
      {tempRank[j]++; break;}
      }
  }
  for each request_outputs [n]{
    for each service_outputs[k] in atomicProcess[j] {
      if equals(service_outputs[k], request_output[n])
        {tempRank[j]++; break;}
    }
  }
  rank_of_atomicProcess[j] = tempRank[j]/
     (atomicProcess[j].service_inputs.size()
        +request_output.size())
  }
  rank[i]=Max(rank_of_atomicProcess[ ]);
}
Step 2: sort and return.
       sort(rank[ ]);
       return rank[ ];
}
```

Fig.3. Baseline matchmaker algorithm.

The baseline matchmaker algorithm is shown in Fig.3. The algorithm calculates the ranked value for each service according to input/output matching ratio between the request and the service. There are many atomic processes in one service. Each atomic process has its input/output matching ratio to the request. We choose the highest matching ratio as the ranked value of the service.

The advantage of the baseline matchmaker is giving a ranked result with numeric value instead of rough categories. But the matching of input/output is based on string comparison, which cannot achieve high accuracy. We improve the matching of input/output by using semantic information in service description, which is discussed in Section 4.

4 Semantic Distance (SD)

The input/output matching in the baseline matchmaker is a keyword-based matching, which does not utilize the semantic information in service description. In semantic description of services, all inputs and outputs are represented by the concepts of ontology. Hence the input/output matching can be based on the concepts, which is implemented by the distance between these concepts.

In this section, we first define ontology model and then define semantic distance to measure the distance between concepts in ontology. Four types of semantic distances are discussed and four algorithms are implemented respectively to calculate them.

Definition 4.1 (Ontology Model). *Ontology model is a tri-tuple model, which can be represented as $O = \langle T, H, X\rangle$, and briefly called ontology.*

T is the set of terminologies. The term in T is called atomic term, including atomic class C and atomic property P. It is represented as $T = \langle C, P\rangle$. There are two kinds of properties, including ObjectProperty and DatatypeProperty. ObjectProperty represents the relation between classes; while DatatypeProperty represents that the property of the class is a value of certain data type. H is the hierarchical set of classes, including subClassOf and subPropertyOf. X is the rule set, or can be called constraint set. It is represented by FOL (First-Order Logic) or DL (Description Logic).

Definition 4.2 (Semantic Distance). *It is defined as the minimum length of relation path between two classes in ontology. The relation path is composed of relations defined in ontology, such as subClassOf or ObjectProperty. When constructing a relation path, ObjectProperty is a one-way relation, and subClassOf is a two-way relation, because superClassOf is also a relation between two classes.*

We can consider the basic categories of exact, subsume, plugin and fail as a rough representation of the distance between two classes in ontology. It only takes subClassOf relation into account to compute the distance of two classes. If two classes have no hierarchical relation, they will have an infinite distance, and will be classified in *fail* category. But other relations between classes are useful since they lead to a deeper semantic understanding.

In short, semantic distance is actually the extension to the relations of the four categories of exact, subsume, plugin and fail.

There are several algorithms which can compute semantic distance using the relations defined in ontology. We introduce four algorithms in the following subsections.

4.1 Unweighted Relation Distance

As illustrated in Fig.4, C1 to C7 are classes in ontology. Now we want to get the SD between C6 and C7. There are two relation paths between C6 and C7. One is C6→C3→C1→C2→C5→C7, which is formed by the relation subClassOf. The other path is C6→C7, which is formed by the relation ObjectProperty.

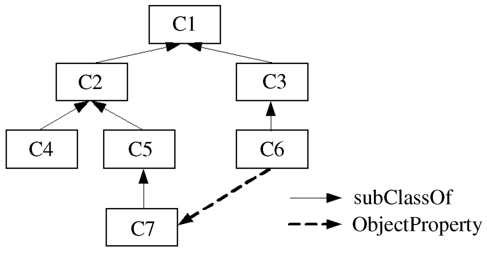

Fig.4. Two kinds of relations in ontology.

The assumption of Unweighted Relation Distance (URD) is that all relations linking two classes have the same weights, no matter what kinds of relations they are.

In this case, obviously, the length of path C6→C3→C1→C2→C5→C7 is 5, while the length of path C6→C7 is 1. The shortest relation path is C6→C7. So the SD between C6 and C7 is the length of C6→C7. That is, URD(C6, C7) = 1.

```
Input: uri1, uri2;
Output: URD;
{
  depth1=getDepth(uri1, uri2); //one way depth
  depth2=getDepth(uri2, uri1); //other way depth
  URD=Min(depth1, depth2);
  return URD;
}
  getDepth(uri1, uri2) {
    depth=0;
    Set S=null; List A=null; Set R1=null;
    S.add (uri1);
    while (true) do {
       for each uri in (S-A) {
          R1.add (uri.superClass());
          R1.add (uri.subClasses()); //all sub classes
          R1.add (uri.objectPropertyClasses());
          /*add classes linked by class uri through
            ObjectProperty relation*/
       }
       if (uri2 ∈ R1) //link to uri2
          return(depth+1); //get depth
       depth=depth+1, A = A ∪ S, S = R1, R1 =null;
    }
}
```

Fig.5. URD algorithm.

The URD algorithm is illustrated in Fig.5. Since the ObjectProperty is a one-way relation, it is necessary to use method getDepth() to get two depths and choose the smaller one as URD.

4.2 Weighted Relation Distance

Weighted Relation Distance (WRD) is the extension of Unweighted Relation Distance, which gives different weights to different relation paths. This idea is inspired by [26], which gives a computation of semantic distance between classes. We extend this work by taking ObjectProperty relation into account. The weight of path is illustrated in Table 1.

Table 1. Weight of Path in WRD

	g	s	o
g	2	1	$3\ (= 1 + 2)$
s	1	2	$3\ (= 1 + 2)$
o	$3\ (= 2 + 1)$	$3\ (= 2 + 1)$	$4\ (= 2 + 2)$

In this table, g stands for generalization, s stands for specialization (inverse of generalization), o stands for ObjectProperty. As a matter of fact, g is like superClassOf, s is like subClassOf.

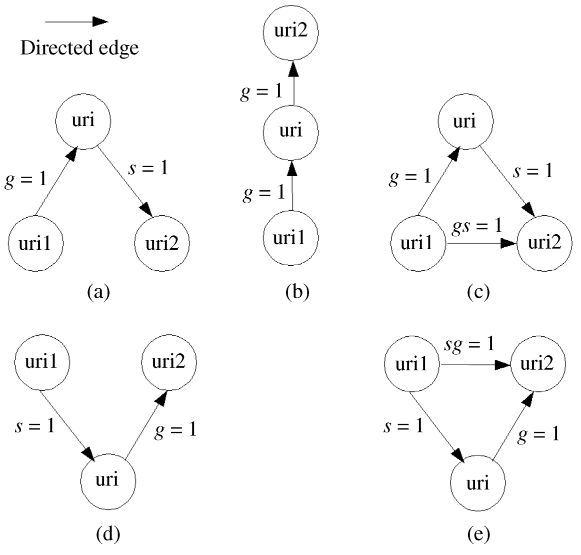

Fig.6. Weight of the paths gs and sg.

In Table 1, the value in the second row and the third column is 1, which indicates that the weight of the path gs is 1. As Fig.6(a) illustrated, uri1 is subClass of uri, and uri is superClass of uri2. So the path from uri1 to uri2 is gs, and its weight is 1, which is less than the weight of g plus s, and less than the weight of the path gg in Fig.6(b). This is because one class's two subClasses are more similar. Then we can add a directed edge gs from uri1 to uri2 like Fig.6(c). We can also add another directed edge gs from uri2 to uri1. Fig.6(d) represents the path sg, then a directed edge sg is added as shown in Fig.6(e).

Fig.7(a) illustrates an ontology, which is converted to a directed graph shown in Fig.7(b). The subClassOf relation in ontology is represented as a two-way directed edge, while the ObjectProperty relation is represented as one-way directed edge directly. If any two classes have the relation like Fig.6(a) or 6(d), a two-way directed edge is added between them, such as the edges between A and B, and between B and C in Fig.7(b).

The WRD from uri1 to uri2 is the shortest path between uri1 and uri2. We use Dijkstra algorithm[27] to find the shortest path and get WRD.

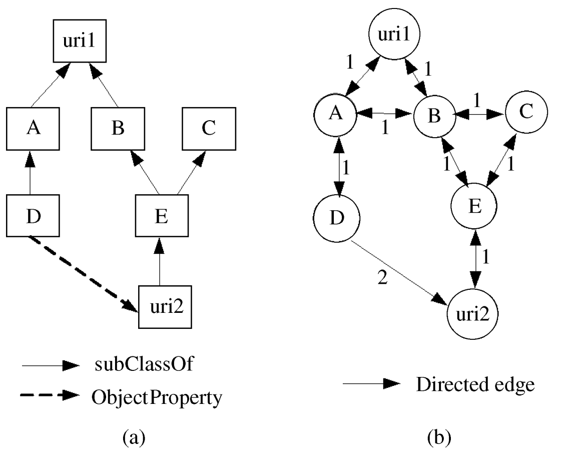

Fig.7. From ontology to directed graph.

4.3 Depth First Distance

WRD does not take the depth of hierarchy into account, which is a problem to compute SD. Let us see Fig.8 bellow.

If we use WRD and ignore the ObjectProperty relation, we can obtain WRD(People, Building) = 1, WRD(Male, Female) = 1, and WRD(Girl, Lady) = 1. Thus, these semantic distances are equal. The result may not be consistent with our intuition. The fact is that, the deeper the relation between two entities is in one hierarchy, the higher the similarity between them is. That is to say, Girl and Lady should be more similar than Male and Female. As a result, the SD between Male and Female should be smaller than the one between People and Building.

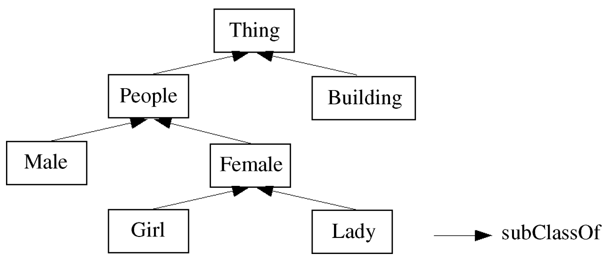

Fig.8. Hierarchical structure in ontology.

To solve this problem, we propose Depth First Distance (DFD) by inversing the formula of GCSM (Generalized Cosine-Similarity Measure)[28]. DFD formula is as follows:

$$\mathrm{DFD}(\mathrm{uri1}, \mathrm{uri2}) = \frac{\mathrm{depth}(\mathrm{uri1}) + \mathrm{depth}(\mathrm{uri2})}{2 \times \mathrm{depth}(\mathrm{LCA}(\mathrm{uri1}, \mathrm{uri2}))}.$$

LCA (Lowest Common Ancestor)[28] is the node of greatest depth that is an ancestor of both uri1 and uri2. Note that the depth is accumulated from the beginning. For example in Fig.8, depth(People) = 1, depth(Lady) = 3. So DFD(Girl, Lady) = $(3+3)/(2\times2) = 6/4 = 1.5$, DFD(Male, Female) = $(2+2)/(2\times1) = 4/2 = 2$.

This implies that the distance between Male and Female is greater than the distance between Girl and Lady, which is consistent with our intuition.

4.4 Synthetized Relation Distance

Since DFD does not concern the ObjectProperty relations between two classes, we propose Synthetized Relation Distance (SRD) to take both WRD and DFD into account.

The original implementation of SRD is to get the minimum distance between WRD and DFD. The idea is that, although two classes have a long hierarchical distance (computed by DFD), they may also have some ObjectProperty relations linking them directly, so the distance between them should be shorter. WRD is sometimes smaller than DFD, so to get the minimum between them may realize this idea.

We find that in WRD, the hierarchical relation's weight is 1 and ObjectProperty relation's weight is 2, while in DFD the hierarchical relation's weight is usually greater than 1. These two algorithms are not consistent in the hierarchical relation's weight.

The solution to this problem is to take the middle layer in the hierarchy tree as standard value 1; and the rest layers will change the distance proportionally.

For example, we can take the Male and Female layer as standard 1. Let SD(Male, Female) = 1 =DFD (Male, Female) × coefficient. Since DFD(Male, Female) = 2, coefficient is 0.5. This coefficient is applied to other distances, such as SD(Girl, Lady) will be $1.5 \times 0.5 = 0.75$.

This solution makes WRD and DFD have the same hierarchical relation's weight at some layers. Then the WRD can take the advantage of hierarchical relation's weight from DFD, and DFD can take the advantage of ObjectProperty relation's weight from WRD.

According to the above discussion, the final form of SRD is Min(WRD, DFD×coefficient). The general form of coefficient would be $(n-1)/n$, if the nodes in depth n have an ancestor in depth $n-1$. The experiments will take the coefficient as a constant 0.5.

5 Semantic Matchmaker

In baseline matchmaker, similarity is calculated by simply using keywords matching between the request and the services. If the words are matched, the similarity is 1, otherwise, the similarity is 0. In this paper, we use the four semantic distance algorithms proposed above to get a more flexible similarity other than 0 or 1. The similarity is a value between 0 and 1. The semantic matchmaker algorithm is shown in Fig.9.

The bold italic characters in Fig.9 are two extensions to the baseline matchmaker algorithm. One extension is using computeRank() method to compute the similarity between service_input/output and request_input/output. The other extension is output weight.

```
Input: request_inputs[], request_outputs[],
          services[];
Output: rank[];
{
Step 1: get each service's rank value.
for each service[i] in service[] {
    rank[i] = 0;  //initial rank value of service[i]
    for each atomicProcess[j] in services[i] {
      tempRank[j] = 0;
      for each service_inputs[k] in atomicProcess[j] {
           for each request_inputs[m] {
         sRank[m] =computeRank(service_inputs[k],
                  request_inputs[m]);
         }
      tempRank[j]+ =Max(sRank[]);
         }
    for each request_outputs[n] {
      for each service_output[k] in atomicProcess[j] {
        sRank[k] =computeRank(service_outputs[k],
                  request_outputs[n]);
         }
        tempRank[j]+ =Max(sRank[])*outputweight;
    }
     rank_of_atomicProcess[j] =tempRak[j]/
            (atomicProcess[j].service_inputs.size()
               +request_output.size()*outputweight);
    }
    rank[i] =Max(rank_of_atomicProcess[]);
}

Step 2: sort and return.
         sort(rank[]);
         return rank[];
}
```

Fig.9. Semantic matchmaker algorithm.

The computeRank() method is shown in Fig.10.

```
Input: service_message, request_message;
Output: similarity;
{
//get message's type uri
uri1 = service_message.getTypeURI(); //get uri1
uri2 = request_message.getTypeURI(); //get uri2
if uri1.typIsString() //uri is string XSD type
{ //get string name of service_message
messageName = service_message.getStringName();
  uri1 = getClosestClass(messageName);
  }
  distance = //try to get distance from cache
     SemanticDistanceCache.getValue(uri1, uri2);
if (distance < 0)  //no value in cache
  distance = computeSD(uri1, uri2); //compute online
  similarity = 1/(e^distance);
  return similarity;
}
```

Fig.10. ComputeRank method.

The method is to compute the similarity between service_input/output and request_input/output. Service_message stands for service_input/output and request_message stands for request_input/output. The main idea of this method is to get the cached value from semantic distance cache. If it does not exist, then compute the SD online. The computeSD() method can choose one of the SD algorithms within URD, WRD, DFD or SRD. Semantic distance cache can improve the performance of semantic matchmaker algorithm, because SD between any two classes of ontology can be pre-computed and pre-cached. Additionally, there are some special cases we should handle.

First, to convert the distance to similarity, we use a similarity function (SF) of $1/(e^{\text{distance}})$. Intuitively, the longer the distance, the lower the similarity. We will discuss SF in Subsection 6.3.

Second, getTypeURI() method should return an ontology class uri, such as http://keg.cs.tsinghua.edu.cn/ontology/travel#City, which can be used to compute SD in SD algorithms. But the method may return a simple XSD type, such as http://www.w3.org/2001/XMLSchema#string, which cannot be used in SD algorithms. In this case, we can regard the name of service_message as messageName, and use getClosestClass method to get a class uri which is the closest to this messageName. The getClosestClass method is to get the class which contains the messageName in its string pattern with the shortest string length. Thus the returned class can be used in SD algorithms.

We use output weight to enhance the semantic matchmaker algorithm. We observe that when a requestor needs a web service, he may concern more about outputs of the service rather than inputs. For example, if all inputs of a request and a web service are matched perfectly, and output of request is not satisfied by the service, then the service is useless for the requestor. Consequently, the ranked value between this request and this service should be very low. The results in the following experiment are proved to be significantly enhanced by output weight.

6 Performance Evaluation

6.1 Experimental Data

We collected WSDL files from the Internet, and transform them into OWL-S as web services repository. There are four sources to obtain WSDL files.

1) Web services collecting site, such as xMethods (http://www.xmethods.com/).
2) Web services search engine, such as SalCentral (http://www.salcentral.com/).
3) Web search engine, such as Google (http://www.google.com).
4) UDDI server, such as http://www.uddi.ibm.com.

We obtain more than one thousand WSDL from these sources, which are all real web services and can be invoked through the web. In this paper, we choose 290 WSDL files in travel domain as experimental data, and transform them into OWL-S by using rules[29].

6.2 Evaluation Criterion

There are three factors which affect the semantic matchmaker algorithm. They are similarity function, output weight and SD algorithms (URD, WRD, DFD and SRD).

In order to evaluate the proposed matchmaker algorithms, we have to obtain the right answer for a specified request. The request is shown in Fig.1. There are 290 OWL-S files in the web services repository, so we need to get 290 ranked values as answers.

First, the answer has to be given manually. We require a person to give a value between 0 and 1 for each service, indicating the matching degree between the service and the request.

Second, only one person is not adequate to determine the answer, because everyone can make mistake, such as missing information, looking across to another web service and so on. We gathered 10 persons to give answers independently.

Third, we propose an algorithm which can determine the final answer from the ten answers. This algorithm assumes that only 70% people will make a right decision for each web service. For each web service, we only choose 7 answers and eliminate other 3 answers.

In order to quantify the evaluation, we define an evaluation function (EF) to compute the comparative distance to the answer. The formula is shown as follows:

$$EF = \frac{1}{\sqrt{\sum_{i=1}^{290}(values[i] - answer[i])^2}}.$$

There are 290 web services in the repository. We give each web service a service ID. Answer[i] is the ranked value given by person for service i. Values[i] is the ranked value computed by semantic matchmaker algorithm for service i. The larger of EF the better.

6.3 Comparison Between Similarity Functions

There are many choices of SFs, we define three SFs as follows:

$$\text{SF}_1 = 1/(\text{distance} + 1),$$
$$\text{SF}_2 = 1/(\text{distance}^2 + 1),$$
$$\text{SF}_3 = 1/(e^{\text{distance}}).$$

These three SFs have the same features as follows:

- when distance = 0, SF = 1;
- SF is decreased when the distance is increased;
- SF is always a real number between 0 and 1.

The difference between them is that the decreasing rates with the increasing of distance are different. The sequence of the decreasing rate from slow to fast is $\text{SF}_1 < \text{SF}_2 < \text{SF}_3$.

We use URD to compute SD, and show comparison between different SFs' effect in the semantic matchmaker algorithm. The reason to choose URD is that other more sophisticated SD algorithms may affect the comparison between SFs. We set output weight to 1 in the algorithm.

To make a concrete comparison, we use evaluation function and get the results as follows:

$$\begin{aligned}&\mathrm{EF(baseline)} = 0.751473133,\\&\mathrm{EF}(\mathrm{SF}_1) = 0.673793516,\\&\mathrm{EF}(\mathrm{SF}_2) = 0.796402303,\\&\mathrm{EF}(\mathrm{SF}_3) = 0.902797892.\end{aligned}$$

We can see that SF_3 is the best one among all similarity functions. The following experiments will use SF_3 as similarity function.

6.4 Comparison Between SDs

The four semantic distance algorithms are compared to evaluate their effects in the semantic matchmaker algorithm. They are URD, WRD, DFD and SRD. We used SF_3 as similarity function. And we set output weight to 1.

By applying the EF defined above, we can get the following results:

$$\begin{aligned}&\mathrm{EF(URD)} = 0.902797892,\\&\mathrm{EF(WRD)} = 0.961250775,\\&\mathrm{EF(DFD)} = 0.898888934,\\&\mathrm{EF(SRD)} = 1.106136137,\\&\mathrm{EF(DFD} \times 0.5) = 0.947407142.\end{aligned}$$

The EF sequence from high to low is: SRD>WRD>DFD×0.5 >URD>DFD.

DFD has the worst performance. The major reason is that DFD is usually greater than 1; and URD has smaller distance than DFD. So the ranked value computed by DFD is usually smaller than that by URD. Consequently, when the similarity between request and service is high, the ranked value computed by DFD will be lower and make larger error.

We introduce the coefficient 0.5 to reduce the DFD distance, and compare the DFD×0.5 with other algorithms. The result of DFD×0.5 is better than URD, but not as good as WRD. And when WRD and DFD×0.5 are mixed together to make the SRD, the result is the best of all.

6.5 Comparison Between SDs Plus OW

To evaluate the effect of output weight in the semantic matchmaker algorithm, we set output weight to 5. URD_OW represents the semantic matchmaker algorithm using URD as semantic distance, and using SF_3 as similarity function. WRD_OW, DFD_OW, SRD_OW and DFD×0.5_OW are the same way like URD_OW.

By applying the EF, we can get the following results:

$$\begin{aligned}&\mathrm{EF(URD\_OW)} = 1.553116947,\\&\mathrm{EF(WRD\_OW)} = 3.340367447,\\&\mathrm{EF(DFD\_OW)} = 3.195993154,\\&\mathrm{EF(SRD\_OW)} = 3.656443513,\\&\mathrm{EF(DFD} \times 0.5\mathrm{\_OW}) = 2.542732149.\end{aligned}$$

All the performances are better than the results in Subsection 6.4. It proves that the output weight can enhance the performance of the semantic matchmaker algorithm.

The evaluation sequence from high to low is: SRD_OW>WRD_OW>DFD_OW>DFD×0.5_OW> URD_OW.

We see that SRD_OW has the best performance. If we contract the evaluations in Subsections 6.3–6.5, the whole evaluation sequence is as follows:

SRD_OW>WRD_OW>DFD_OW>DFD×0.5_OW> OW>URD_OW>SRD>WRD>DFD×0.5>URD(SF_3) >DFD>URD(SF_2) >baseline>URD(SF_1).

Most of the algorithms in the semantic matchmaker are better than the baseline matchmaker algorithm. And SRD_OW still has the best performance.

In summary, when the semantic matchmaker algorithm uses SRD, SF_3 and output weight together, the performance is the best. And the proposed semantic matchmaker has much better performance than the baseline matchmaker.

7 Conclusions and Future Work

The contribution of this paper is proposing a new method of semantic matchmaker to give ranking result instead of classifying for service matching, which can help the service requestor to choose the best web service during service discovery process. To estimate the matching degree of input/output in semantic description, we define semantic distance and give four algorithms to calculate it. We propose the semantic matchmaker for ranking web services by using semantic distance, similarity function and output weight. Experimental results show that the proposed semantic matchmaker algorithm significantly outperforms the baseline matchmaker.

As future work, we plan to use precondition and effect (PE) of OWL-S in the matchmaker. The difficulty to process PE is that the logic expression of PE is premature and does not have a sufficiently-standardized specification. Other future work may be the expression of personal interest in the matchmaker, which is also related to the logic expression.

References

[1] Haas H. Web services activity statement. W3C, http://www.w3.org/2002/ws/Activity, 2001.

[2] Clement L, Hately A, Riegen C V *et al.* UDDI version 3.0.2. UDDI Spec. Technical Committee Draft, October 19 2004. http://uddi.org/pubs/uddi_v3.htm.

[3] Christensen E, Curbera F, Meredith G *et al.* Web services description language (WSDL). http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[4] Gudgin M, Hadley M, Mendelsohn N *et al.* SOAP version 1.2 part 1: Messaging framework, W3C recommendation. http://www.w3.org/TR/soap12-part1/, June 24, 2003.

[5] Qiu D, Srikant R. Modeling and performance analysis of bitTorrent-like peer-to-peer networks. In *Proc. ACM SIGCOMM*, Portland, Dregon, USA, 2004, pp.367–378.

[6] Liu Y, Xiao L, Liu X *et al.* Location awareness in unstructured peer-to-peer systems. *IEEE Trans. Parallel and Distributed Systems* (*TPDS*), 2005, 16(2): 163–174.

[7] Liu Y, Zhang Z, Xiao L, Ni L. A distributed approach to solving overlay mismatching problem. In *Proc. 24th Int. Conf. Distributed Computing Systems*, Tokyo, Japan, March 2004, pp.132–139.

[8] Paolucci M, Kawamura T, Payne T R *et al.* Semantic matching of web services capabilities. In *Proc. 1st Int. Semantic Web Conf.* (*ISWC2002*), Sardinia, Italy, June 2002, pp.333–347.

[9] Martin D, Burstein M, Hobbs J *et al.* OWL-S: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1/overview/, 2004.

[10] Sivashanmugan K, Verma K, Sheth A *et al.* Adding semantics to web services standards. In *Proc. Int. Conf. Web Services*, Las Vegas, USA, June 23–26, 2003, pp.395–401.

[11] Ogbuji U. Supercharging WSDL with RDF—Managing structured web service metadata. IBM developerWorks article, 2000.

[12] Peer J. Bringing together semantic web and web services. In *Proc. 1st Int. Semantic Web Conf.* (*ISWC2002*), Sardinia, Italy, June 2002, pp.279–291.

[13] Worden R. A meaning definition language. White paper. http://www.charteris.com/mdl/MDLWhitePaper.pdf, 2001.

[14] Sriharee N, Senivongse T. Discovering web services using behavioral constraints and ontology. In *Proc. 4th IFIP Int. Conf. Distributed Applications and Interoperable Systems*, Paris, France, November 19–21, 2003, pp.248–259.

[15] Sirin E, Bijan P, Hendler J. Composition-driven filtering and selection of semantic web services. In *Proc. 19th Conf. Artificial Intelligence* (*AAAI04*), San Jose, USA, July 25–29, 2004, pp.129–136.

[16] Patil A, Oundhakar S, Sheth A *et al.* METEOR-S web service annotation framework. In *Proc. 13th Int. WWW Conf.*, New York, USA: ACM Press, 2004, pp.553–562.

[17] Bansal S, Vidal J M. Matchmaking of web services based on the DAML-S service model. In *Proc. 2nd Int. Joint Conf. Autonomous Agents and Multi-Agent Systems* (*AAMAS'03*), Melbourne, Australia, July 14–18, 2003, pp.926–927.

[18] Klein M, Bernstein A. Searching services on the semantic web using process ontologies. In *Proc. Int. Semantic Web Working Symposium* (*SWWS*), Amsterdam: IOS Press, 2001, pp.159–172.

[19] Jaeger M C, Rojec-Goldmann G, Liebetruth C *et al.* Ranked matching for service descriptions using OWL-S. In *Proc. Communication in Distributed System* (*KiVS05*), Kaiserslautern, Germany, 2005, pp.91–102.

[20] Verma K, Sivashanmugam K, Sheth A *et al.* METEOR-S WSDI: A scalable infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management*, 2005, 6(1): 17–39.

[21] Bentallah B, Hacid M, Alain L *et al.* On automating web services discovery. *VLDB Journal*, 2005, 14(1): 84–96.

[22] Gonzalez-Castillo J, Trastour D, Bartolini C. Description logics for matchmaking of services. In *Proc. The Workshop on Applications of Description Logics* (*KI'01*), Vienna, Austria, 2001, pp.12–24.

[23] Caragea D, Syeda-Mahmood T. Semantic API matching for automatic service composition. In *Proc. 13th Int. WWW Conf.*, New York, USA, 2004, pp.436–437.

[24] Syeda-Mahmood T, Shah G, Akkiraju R *et al.* Searching service repositories by combining semantic and ontological matching. In *Proc. Int. Conf. Web Services* (*ICWS'05*), Orlando, Florida, USA, 2005, pp.13–20.

[25] Zhang P. Travel Ontology. http://www.schemaweb.info/schema/SchemaDetails.aspx?id=236, 2005.

[26] Sycara K, Widoff S, Klusch M *et al.* Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. In *Proc. Conf. Autonomous Agents and Multi-Agent Systems* (*AAMAS'02*), Bologna, Italy, July 2002, pp.173–203.

[27] Dijkstra algorithm. http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml.

[28] Ganesan P, Garcia-Molina H, Widom J. Exploiting hierarchical domain structure to compute similarity. *ACM Trans. Information Systems*, 2003, 21(1): 64–93.

[29] Zhang D, Li J, Xu B. Web service annotation using ontology mapping. In *Proc. 1st Int. Workshop on Service Oriented System Engineering*, Beijing, China, 2005, pp.235–242.

Bin Xu received his B.S. and M.S degrees from the Dept. Computer Science and Technology, Tsinghua University in 1996 and 1998, respectively. He is now a lecturer. His current research interests focus on web service and semantic web.

Po Zhang received his B.S. and M.S. degrees from the Dept. Computer Science and Technology, Tsinghua University in 2003 and 2006, respectively. His research interests focus on semantic web service.

Juan-Zi Li received her Ph.D. degree from the Dept. Computer Science and Technology, Tsinghua University in 2000. She is now an associate professor of Tsinghua University. Her current research interests focus on XML information processing, semantic web and web service.

Wen-Jun Yang received his B.S. and Ph.D. degrees from the Dept. Computer Science and Technology, Tsinghua University in 2000 and 2006, respectively. His research interests focus on semantic web service.