International Journal of u- and e- Service, Science and Technology
Vol.9, No. 9 (2016), pp.187-200
http://dx.doi.org/10.14257/ijunesst.2016.9.9.19

A Semantic Web Service Discovery System Based on UDDI and OWL-S

Li Peng

School of Information Science and Engineering, Hunan First Normal University, Changsha, China
goodbetter@163.com

Abstract

As a popular registry for Web services, UDDI provides a mature mechanism for service registration and search, but it does not support semantic operations on Web services, so its service discovery capability is not satisfactory. OWL-S overcomes UDDI's shortcomings. Using OWL-S, service advertisements and requests based on capability description could be built in a uniform way, and the semantic matching between them could also be achieved. By means of these two technologies, a system was developed which is compatible with UDDI and supports the publication and discovery of semantic Web services. The matching performed by the system is divided into two parts: pre-matching and post-matching. The pre-matching is generally performed only in the course of service publication, so the time for service discovery is reduced. Experimental results show that the performance of the system is reliable. Furthermore, the system can produce query results ordered by priority according to the requester's expectations.

Keywords*: Web service, UDDI, OWL-S, publication and discovery, semantic matching*

1. Introduction

In a distributed heterogeneous environment, especially on the Web, it is very hard to achieve the interoperability of applications. The Web service has become a standard way to solve this problem because of its advantages of simplicity, loose coupling, ease of integration, *etc.*, provided by its components such as XML, SOAP [1], and WSDL [2]. At present, Web service technology is widely adopted by B2B and B2C applications.

With the increase of the number of Web services, it is more and more difficult to find the services that satisfy our requirements. As an industrial standard recommended by OASIS, Universal Description, Discovery and Integration (UDDI) [3] was developed to solve the problem of service publication and discovery. UDDI is a registry that allows service providers to publish their services and allows service requesters to discover the services that meet their needs. Although UDDI has some outstanding features that make it an appealing registry for Web services, it has two shortcomings in terms of search capability. Firstly, it only supports the keyword search because it uses XML to describe its data models, which supports syntax operations only. Secondly, its search mechanism is not powerful enough because it only provides limited support for capability-based service search. Because of the two shortcomings, the search results produced by UDDI usually are not accurate or complete enough, which means its service discovery capability is unsatisfactory.

In fact, UDDI's two shortcomings reflect a common limitation with traditional Web service technologies, that is, they are only used to process Web services on the syntax layer. The emergence of the Semantic Web has brought hope for overcoming the limitation. The goal of the Semantic Web is to introduce machine-understandable semantics into Web information such that it can be automatically processed by machines. Web Ontology Language (OWL) [4] is a technical standard for the Semantic Web. As a

ISSN: 2005-4246 IJUNESST

knowledge representation language, OWL has not only sufficient expressivity which supports the representation of complicated knowledge, but also well-defined semantics which makes knowledge-based inference possible. To bring semantics to Web services, an OWL ontology framework OWL-S [5] was developed. With the help of OWL-S, service providers can advertise what their services do, and service requesters can specify what capabilities they expect from the services they need to use [6]. As a result, capability-based semantic matching between service requests and advertisements can be achieved. OWL-S overcomes UDDI's shortcomings, so it is believed that OWL-S can improve the effect of service discovery.

In the paper, a system is proposed which takes advantage of OWL-S and UDDI to achieve the publication and discovery of semantic Web services. For the client, the system acts like a UDDI registry except that it can also accept the advertisements and requests containing OWL-S profile information. To reduce the time of service discovery, the matching performed by the system is divided into two parts, pre-matching and post-matching, and the pre-matching is generally performed only in the course of service publication. Experimental results show that the system's performance is reliable. Furthermore, the system can produce query results ordered by priority according to the requester's expectations.

The rest of the paper is organized as follows. Section 2 is the background introduction, including UDDI, OWL-S and related work. Section 3 outlines the system's architecture. Section 4 illustrates the requirements for OWL-S advertisements and requests. Section 5 describes the method of converting OWL-S profiles into UDDI data structures. Section 6 describes the processes of service publication and discovery. Section 7 describes the matching performed by the system. In section 8, the performance of the system is analyzed. Section 9 is the conclusion. It is assumed that the readers of this paper are familiar to the technical details of UDDI and OWL-S.

2. Background

2.1. UDDI

As a universal service registry, UDDI supports various ways of searching. Published services can be searched by name, description, business, or tModel. For example, we can search by tModel for the services that adhere to a specific technical specification. However, UDDI's search mechanism is based on keyword matching and therefore does not support any inference about search criteria. For example, a service may be categorized as "Passenger Car Rental", but a search for "Automotive Equipment Rental and Leasing" services cannot find it despite the fact that "Passenger Car Rental" is a subtype of "Automotive Equipment Rental and Leasing" in the NAICS classification scheme. In general, a service requester does not know how published services are described and may use some words in the request which are not present in the descriptions of published services. As a result, some services which are useful to the requester may be discarded in the search. In a word, UDDI works well only when the requester knows some information about published services.

UDDI supports classifying services and searching by category. Searching by category can actually narrow down search results, but they may still include some services which are no of interest to us. For example, when searching for "Passenger Car Rental" services, we may not be interested in the ones that don't provide a certain car type. To get more precise results, the search mechanism should consider more functional properties of services such as input and output, not just category. Unfortunately, UDDI does not support other functional properties than category, which means its search mechanism is not powerful enough.

2.2. OWL-S

OWL-S is an ontology framework based on OWL. Its goal is to realize automatic discovery, invocation, and composition of Web services. OWL-S mainly includes three sub-ontologies: service profile, process model, and grounding. The service profile is used to describe what the service does. The process model is used to describe how the service is used. The grounding is used to describe how to interact with the service. Among them, the service profile is normally used for service discovery.

OWL-S provides two ways to represent the capabilities of Web services [6]. The first way is to build a class hierarchy with OWL ontologies, with each class representing a set of similar services. Using such a hierarchy, a service can be defined as an instance of the classes that represent its capabilities. For example, a toy store can be defined as an instance of the OWL class for toy selling services. The second way is to describe the functionality of a service in a generic manner according to its state transition. For example, a toy selling service can be specified to take as inputs a toy name, an address, and a credit card number, and produce a state transition such that the toy is delivered to that address, and the credit card is charged the toy price. In one of the above two ways, service requests and advertisements based on capability description can be created in a uniform manner. In addition, OWL-S supports semantic inference, so capability-based semantic matching between service requests and advertisements can be achieved.

2.3. Related Work

The study of using OWL-S for service discovery has been carried out for some years. A lot of capability matching algorithms based on OWL-S service profile have been presented. To represent the capabilities of services, they generally adopt one of the two ways mentioned in the previous section. In some algorithms [7-8], functionality ontologies are used to describe the capabilities of services, and the matching between a request and an advertisement is reduced to a subsumption test between two ontology classes. In the other algorithms [9-11], a service's capabilities are regarded as the state transition produced by it, and the matching is performed by comparing the state transition described in the request with that described in the advertisement. These algorithms generally employ a ranking mechanism to evaluate matching results. Researchers have also studied how to integrate OWL-S into the existing Web service discovery architecture. The OWL-S/UDDI matchmaker [9-12] enhances UDDI's service discovery capability by performing capability-based semantic matching. This enhancement is based on the conversion of OWL-S service profiles into UDDI data structures [13]. The semB-UDDI registry [14] provides modified UDDI APIs to support the processing of OWL-S ontologies, thus improving the coupling between UDDI and OWL-S. A similar idea is also adopted in [15].

3. System Architecture

Although UDDI does not support semantic operations on Web services, it provides a mature mechanism for service registration and search, so it is still adopted by many applications. In view of this situation, our system was designed to be compatible with UDDI. As shown in Figure 1, the system is composed of a UDDI registry, a publication component, a query component, and an OWL reasoner with a knowledge base, and maintains a concept hierarchy. The system exposes all the APIs of the UDDI component to the client, so it can accept all UDDI messages. Furthermore, to support the publication and discovery of semantic Web services, it provides two APIs which are used to accept the advertisements and requests containing OWL-S profile information.

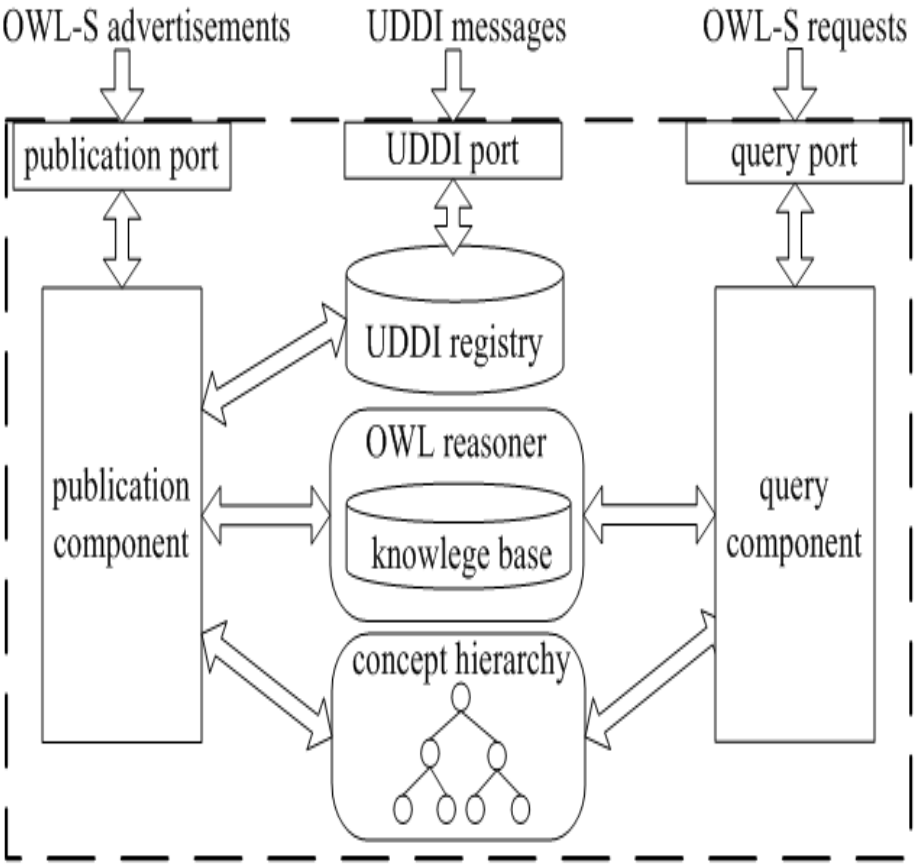

Figure 1. System Architecture

On receiving an OWL-S advertisement, the publication component registers all the services being published with the UDDI component, loads the ontologies used by the advertisement into the reasoner, updates the concept hierarchy, and performs the pre-matching based on the concept hierarchy. On receiving an OWL-S request, the query component performs the post-matching by searching the concept hierarchy, so as to find the services that match the request. In specific cases, it also needs to load ontologies into the reasoner and update the concept hierarchy. Because the client has direct access to the UDDI component, it can retrieve all the information saved in the UDDI component, including the semantic description information of services.

4. Requirements for OWL-S Advertisements and Requests

The system can accept the advertisements and requests containing OWL-S profile information, but supports OWL-S 1.2 only. Multiple profiles are allowed to be contained in an OWL-S advertisement in support of the simultaneous publication of multiple services, while only one profile is allowed to be contained in an OWL-S request. The types of the inputs and outputs in OWL-S advertisements and requests must be OWL concepts, not XML Schema types, because inference cannot be applied to the latter. In the paper, the concepts used as the types of inputs and outputs are called type concepts. The inputs in an OWL-S request are required to be arranged in order of how much the requester expects them to be achieved, and the same requirement applies to the outputs, because the system will determine the priority of the services that match the request according to the two orders. Each profile in an OWL-S advertisement must contain a “contactInformation” element, and its value must be a businessKey that identifies the

service provider already registered in the system. The reason for that will be explained in the next section.

5. Converting OWL-S profiles into UDDI Data Structures

On receiving an OWL-S advertisement, the publication component converts each profile in it into a UDDI data structure businessService. To achieve this, a mapping mechanism is adopted, which is a modified version of that described in [16]. As shown in Figure 2, the mapping is divided into two types: direct mapping and tModel-based mapping. The direct mapping is applied to the OWL-S profile elements which have corresponding UDDI elements, such as serviceName, and it converts an OWL-S profile element into its UDDI counterpart by assignment. Note that contactInformation is directly mapped to businessKey, although there is no strict correspondence between them. For those OWL-S profile elements with no corresponding UDDI elements, such as hasInput and hasOutput, the tModel-based mapping is adopted, which is similar to the WSDL-to-UDDI mapping proposed in [17]. To perform the tModel-based mapping, eight tModels need to be created, each of which corresponds to an OWL-S profile element with no UDDI counterpart. For example t_Input corresponds to hasInput. These tModels are used in a way similar to how the NAICS tModel is used to describe the categories of services. Through the tModel-based mapping, an OWL-S profile element is converted into a keyedReference in the categoryBag. The keyedReference contains a reference to the tModel corresponding to the element and some key information the element describes. For example, a "hasInput" element which describes an input is converted into a keyReference, in which the keyName and keyValue attributes are set to the input's URI and the URI of the type concept of the input, respectively, and the tModelKey attribute is set to the tModelKey of t_Input. Figure 3 shows the keyedReferences converted from a "hasInput" element and a "hasOutput" element.

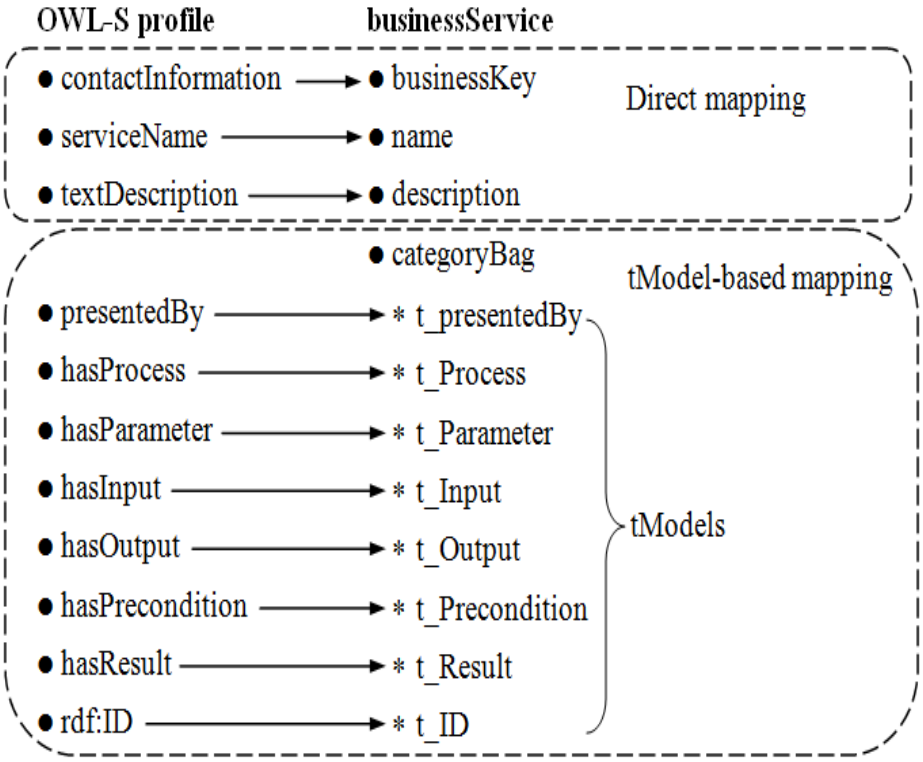

Figure 2. OWL-S-to-UDDI Mapping

After the conversion, all the produced businessServices will be saved into the UDDI component by the invocation of the UDDI API save_service. If a service provider wants to publish services, it should first register itself in the system to get a businessKey, and when publishing services, put the businessKey into the OWL-S advertisement to identify itself, as required at the end of the previous section. If not doing so, the businessServices produced through the conversion will have no businessKey or have an incorrect businessKey, therefore cannot be saved into the UDDI component.

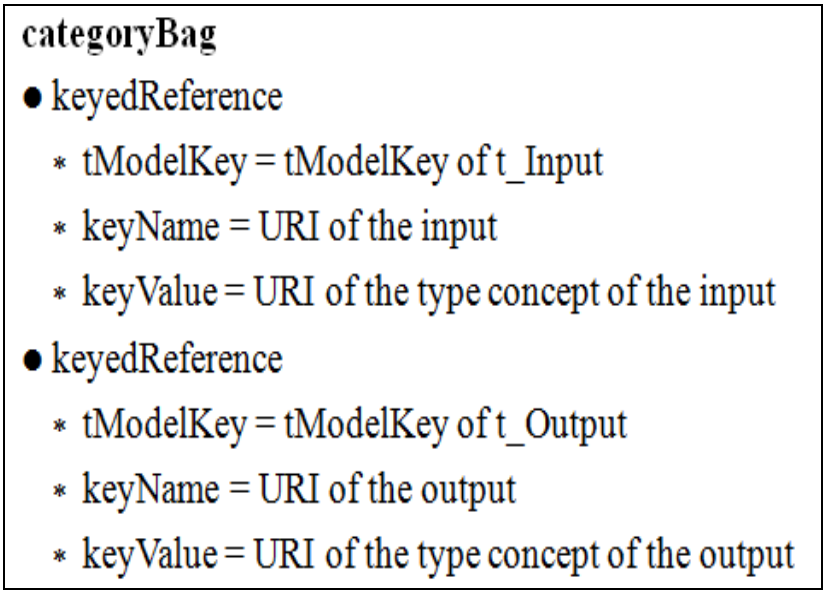

Figure 3. KeyedReferences Converted from HasInput and HasOutput

6. Service Publication and Discovery

6.1. Service Publication

On receiving an OWL-S advertisement, the publication component performs the following steps.

(1) As described in Section 5, the publication component converts all the OWL-S profiles in the advertisement into businessSevices, and saves them into the UDDI component.

(2) The publication component returns a response message to the client, which is similar to the response message of the UDDI API save_service.

(3) For each of the ontologies that are used by the inputs and outputs in the advertisement, if it is not present in the reasoner, the publication component downloads it, validates its consistency with the reasoner, and loads it into the reasoner.

(4) For each concept in the newly-loaded ontologies, if it is new to the system (not present in the concept hierarchy), the publication component fetches the subsumption relation information associated with it out of the reasoner, and inserts it into the concept hierarchy according to the information.

(5) The publication component performs the pre-matching, which will be described in Section 7.2.

Considering that steps 3-5 may consume some time, they are scheduled to execute after step 2. This way, the response time is reduced. If they happen to fail, the related services will be unregistered in the UDDI component. Doing so may confuse the client because some services published successfully are removed afterwards, but this possibility is very low.

6.2. Service Discovery

On receiving an OWL-S request, the query component performs the following steps.

(1) This step is similar to step 3 in Section 6.1, with the difference that the ontologies used by the inputs and outputs in the request are processed.

(2) This step is the same as step 4 in Section 6.1.

(3) The query component performs the second part of operations of the pre-matching.

(4) The query component performs the post-matching, which will be described in Section 7.3.

(5) The query component returns a response message to the client, which contains a list. Each item in the list is a simple description of a service that matches the request, including a serviceKey that identifies the service in the UDDI component, and two matching result lists of the service. All the items are ordered according to the priority of the services they describe.

The first three steps are optional, that is, the requester can choose to perform or not to perform them, and by default they don't need to be performed. We tend not to perform them for the following two reasons. Firstly, downloading, validating, and loading an ontology are all time consuming, and the number of the ontologies allowed to be loaded is theoretically unlimited. Secondly, even if an ontology used by the request is loaded, it is very likely that the new concepts in it have no relation with the concepts already present in the concept hierarchy, so the matching results will not be influenced anyway. However, omitting the first three steps may result in an incomplete matching because the possibility that the new concepts have relations with the concepts already present in the concept hierarchy still exists. So we also allow them to be performed. In fact, even if the requester chooses to perform them, as more and more services are published, the number of loaded ontologies increases rapidly. Then after the system runs for a period of time, the work required to process a request can be roughly considered to just perform the last two steps, because no ontology needs to be loaded.

7. Matching Performed by the System

To discover the services that match a request, a natural idea is to match the request against each published service [10]. However, with the increase of the number of published services, the time taken to process a request increases as well. To solve the problem, the matching performed by the system is divided into two parts: pre-matching and post-matching. The former completes most of matching work, but it is generally performed only in the course of service publication, so the time required for service discovery is reduced.

7.1. Degree of Match Between two Concepts

In the system, the matching between a request and a published service is based on the matching between two concepts. There are different methods of computing the degree of match between two concepts. Some methods [12-18] have high computational efficiency but produce less precise results, while the other methods [19-22] are quite the opposite. To balance between efficiency and precision, we adopt a method which extends the method in [12] with path length information.

Given two concepts C and C′ in an OWL concept hierarchy, let's use D to represent C's degree of match against C′. D is defined as follows.

(1) If C and C′ are equivalent, D=1. This means C exactly matches C′.

(2) If C subsumes C′, $D=0.5\times(1+1/e^{M/8})$, where M is the length of the longest directed path from C to C′.

(3) If C is subsumed by C′, $D=0.5/e^{M/8}$, where M is the length of the longest directed path from C′ to C.

(4) If there is no subsumption relation between C and C′, D=0. This means C fails to match C′.

According to the definition, in Figure 4, C2's degrees of match against C3, C5, C1, and C4 are 1, 0.94, 0.44, and 0, respectively. The reason that path length information is introduced is as follows. For any three concepts C, C′ and C″ in an OWL concept hierarchy, if C subsumes both C′ and C″, and C′ subsumes C″, obviously C's degree of match against C′ should be higher than that against C″ because C is semantically closer to C′ than C″; similarly, if both C′ and C″ subsume C, and C″ subsumes C′, C's degree of match against C′ should be higher than that against C″. It can be seen that according to the definition, C2's degree of match against C7 does be higher than that against C8 in Figure 4. Except for introducing path length information, our method is essentially the same as that in [12].

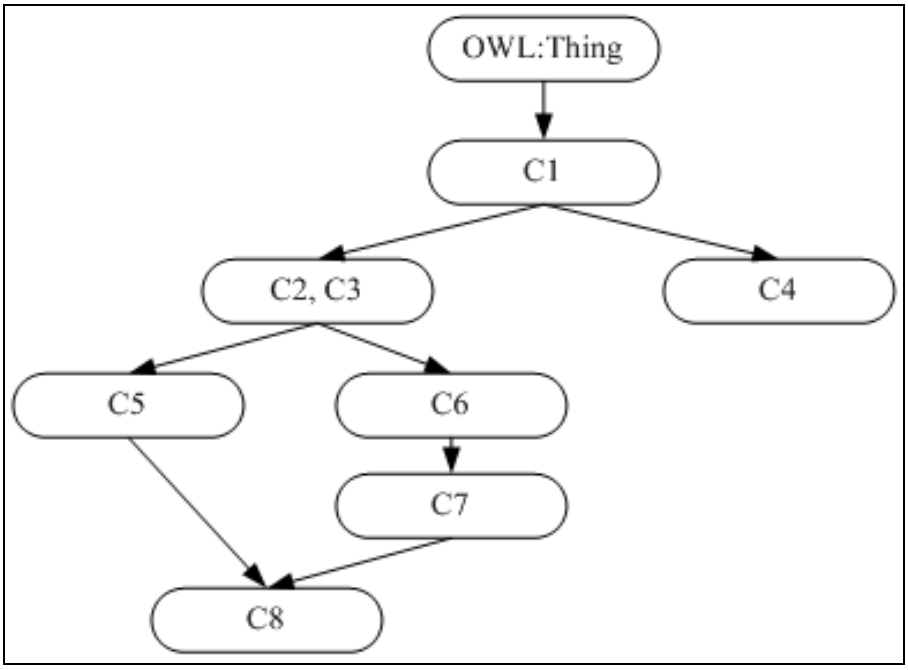

Figure 4. An OWL Concept Hierarchy

7.2. Pre-Matching

The idea behind the pre-matching is to take all the concepts in the system as the type concepts used by potential requests, and take the time of service publication to pre-match them against the type concepts used by the services being published.

The pre-matching is based on a concept hierarchy maintained by the system. As services are published, new concepts are inserted into the hierarchy, as described in section 6.1. In the hierarchy, each node is a group of equivalent concepts, and each edge represents the direct subsumption relation between two groups of equivalent concepts. Each concept is provided with two pre-matching result sets, ISet and Oset, both of which are a set of triples. For any concept, an element (S,C,D) in its ISet indicates its degree of match against the type concept C of an input of the service S is D; an element (S,C,D) in its OSet has the similar meaning with the difference that C is the type concept of an

output of S. Although there is a similar hierarchy inside the reasoner, it cannot be directly accessed. Thus we set up such a structure to improve the efficiency of the pre-matching.

The operations of the pre-matching are divided into two parts. To find and record the match between the concepts in the hierarchy and the type concepts used by the services being published, the first part of operations is firstly performed.

(1) The first part of operations

```
for each service S being published {
    for each input of S {
        locate its type concept C in the concept hierarchy.
        add (S,C,1) to C's ISet.
        for each equivalent concept, ancestor and descendant C1 of C {
            add (S,C,D) to C1's ISet, where D is C1's degree of match against C. }}
    for each output of S {
        the similar operations are performed such that some concepts' OSets are
        updated. }}
```

Before the pre-matching starts, some new concepts may be inserted into the concept hierarchy, so there is a need to find and record the match between them and the type concepts used by published services. Furthermore, the degrees of match of some concepts in the hierarchy against the type concepts used by published services may vary because of the inserting of the new concepts, so there is a need to update their pre-matching result sets. To fulfill these tasks, the second part of operations is then performed.

(2) The second part of operations

```
for each new concept Cn {
    locate Cn in the concept hierarchy.
    for each equivalent concept, ancestor and descendant C1 of Cn {
        if C1's ISet includes an element (S,C,D), such that S is a published service,
        D=1,
        and C=C1{
            add (S,C1,D1) to Cn's ISet, where D1 is Cn's degree of match against C1. }
        if C1's OSet includes an element (S,C,D), such that S is a published service,
        D=1,
        and C=C1{
            add (S,C1,D1) to Cn's OSet, where D1 is Cn's degree of match against
C1. }}
    for each ancestor C1 of Cn {
        if C1 is not a new concept {
            for each element (S,C,D) in C1's pre-matching result sets {
                if both Cn and C1 are the ancestors of C {
                    update D to reflect C1's current degree of match against C. }}}}
    for each descendant C1 of Cn {
        if C1 is not a new concept {
            for each element (S,C,D) in C1's pre-matching result sets {
                if both Cn and C1 are the descendants of C {
                    update D to reflect C1's current degree of match against C. }}}}}
```

Figure 5, gives an example of the pre-matching. Let's assume that (a) is the concept hierarchy of when the pre-matching just starts; S2 is the only service being published, and it has an input and an output, whose type concepts are both C2; only C5 is a new concept. When the pre-matching ends, the concept hierarchy will change to (b). It can be seen that the pre-matching result sets of C1-C5 are all updated.

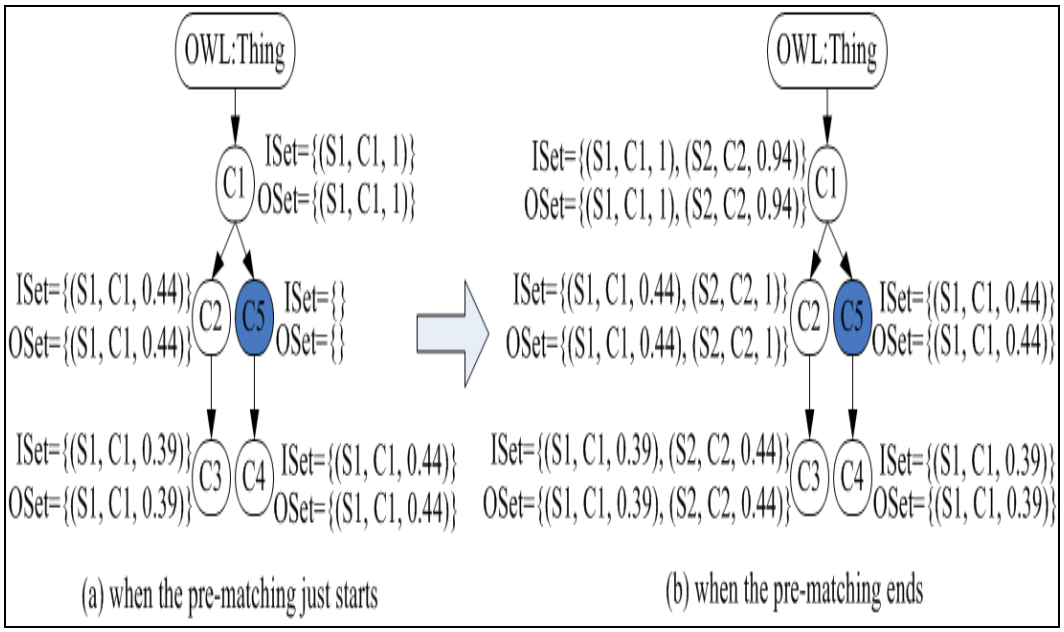

Figure 5. An Example of the Pre-Matching

7.3. Post-Matching

After receiving an OWL-S request, the query component performs the post-matching to find the services that match the request and generate two matching result lists for each of them. Because most of matching work is completed by the pre-matching, the work left to the post-matching is just some simple searches in the concept hierarchy and set operations.

The operations of the post-matching are divided into two parts. The first part of operations is used to find the services that match the request and put them into a set Servs. The condition that a service matches the request is that the type concept of each input in the request is matched by the type concept of an input of the service, and the type concept of each output in the request is matched by the type concept of an output of the service.

(1) The first part of operations

```
create two sets Servs and Temp.
for each input in the request {
    locate its type concept CI in the concept hierarchy.
    if the previous operation fails { the post-matching fails. }
    empty Temp.
    for each element (S,C,D) in CI's ISet { add S to Temp. }
    if CI is the type concept of the first input in the request { let Servs=Temp. }
    let Servs=Servs∩Temp.
    if Servs is empty { the post-matching fails. }}
for each output in the request {
    locate its type concept CO in the concept hierarchy.
    if the previous operation fails { the post-matching fails. }
    empty Temp.
    for each element (S,C,D) in CO's OSet { add S to Temp. }
    let Servs=Servs∩Temp.
    if Servs is empty { the post-matching fails. }}
```

The second part of operations is used to generate two matching result lists IList and OList for each of the matched services. For a matched service, the ith element (C_i,D_i) in

its IList indicates that of the type concepts of its inputs, C_i is the best match of the type concept of the ith input in the request, and the degree of match is D_i; the ith element (C_i,D_i) in its OList indicates that of the type concepts of its outputs, C_i is the best match of the type concept of the ith output in the request, and the degree of match is D_i.

(2) The second part of operations

```
create a tuple (Ct,Dt).
for each matched service Sm {
    create two lists IList and OList for Sm.
    for each input in the request {
        locate its type concept CI in the concept hierarchy.
        let Dt=0.
        for each element (S,C,D) in CI's ISet {
            if S=Sm and D>Dt { let (Ct,Dt)=(C,D). }}
        append (Ct,Dt) to Sm's IList. }
    for each output of the request {
        locate its type concept CO in the concept hierarchy.
        let Dt=0.
        for each element (S,C,D) in CO's OSet {
            if S=Sm and D>Dt { let (Ct,Dt)=(C,D). }}
        append (Ct,Dt) to Sm's OList. }}
```

Once the matching result lists of all the matched services are produced, their priority could be determined according to the following rules. Given two matched service S and S′, let's assume the IList and OSet of S are $((CI_1,DI_1),\ldots,(DI_m,DI_m))$ and $((CO_1,DO_1),\ldots,(CO_n,DO_n))$ respectively, and those of S′ are $((CI'_1,DI'_1),\ldots,(CI'_m,DI'_m))$ and $((CO'_1,DO'_1),\ldots,(CO'_n,DO'_n))$ respectively, where m and n are the numbers of the inputs and outputs in the request, respectively.

(1) S and S′ have the same priority, if $DI_i=DI'_i$ for any positive integer $i \leq m$, and $DO_i=DO'_i$ for any positive integer $i \leq n$.

(2) S has the higher priority than S′ has, if there is an integer i such that $DI_i>DI'_i$ and $DI_j=DI'_j$ for any positive integer $j<i$, or there is an integer i such that $DO_i>DO'_i$, $DO_j=DO'_j$ for any positive integer $j<i$, and $DI_k=DI'_k$ for any positive integer $k \leq m$.

(3) Otherwise S has the lower priority than S′ has.

In general, the requester expects the inputs in the request are achieved firstly, secondly the outputs. Furthermore, as required by the system, the order of how much the requester expects the inputs in the request are achieved should be reflected on their arrangement order in the request, and the same goes for the outputs. It can be seen that the rules conform to the requester's expectations.

8. Performance Evaluation

In this section, we analyze the publication response time and query response time of the system, and verify the analysis with experimental results. The system uses a Racerpro server [23] as the OWL reasoner, and a jUDDI registry [24] as the UDDI component. In the experiments, each OWL-S advertisement contains a profile which has three inputs and two outputs; each OWL-S request contains two inputs and an output; the measurement of response time excludes network latency.

8.1. Publication Response Time

As described in section 6.1, to process an OWL-S advertisement, five steps need to be performed, but the publication response time only includes the time taken to perform the first two steps. Clearly, the time required for the two steps does not depend on the number of published services. In the first experiment, we published 200 advertisements and

measured the response time of each publication operation. The first line of Table 1, shows the average response time of these operations. The small standard deviation indicates that the publication response time is almost constant. This result accords with our analysis.

Table 1. Publication Response Time and Query Response Time

	Average response time in ms	Standard deviation
Publication	220.7	45.6
Query	143.2	28.5

8.2. Query Response Time

As described in section 6.2, in most cases, it's unnecessary to perform the first three steps to process an OWL-S request, so we analyze the query response time only for the condition that only step 4 and step 5 are performed. The operations performed in step 4 are divided into two parts (see section 7.3). The time required for the first part can be expressed as $(N_{in}+N_{out})\times(T_{locating}+T_{set\text{-}operation})$, where N_{in} and N_{out} are respectively the numbers of the inputs and outputs in the request, $T_{locating}$ is the time required to locate in the concept hierarchy a type concept used by the request, and $T_{set\text{-}operation}$ is the time required for those set operations associated with the concept. In the implementation of the system, the dynamic hashing technology is used to manage the concepts in the concept hierarchy, so $T_{locating}$ can be considered constant. Moreover, we can see $T_{set\text{-}operation}$ does not depend on the number of published services. This means that the time required for the first part does not depend on the number of published services. As for the second part, we can draw the same conclusion. Step 5 mainly involves the sorting of the matched services and the construction of the response message, and the time required for it does not depend on the number of published services, either.

In the second experiment, we performed 200 queries and measured the response time of each query. Between each two queries, we published two services as well. We chose not to perform the first three steps when a request was processed. The second line of table 1 shows the average response time of these queries. The small standard deviation indicates that the query response time is almost constant. This result is consistent with the above analysis.

9. Conclusion

To overcome UDDI's shortcomings in search capability, we developed a software system which takes advantage of UDDI, OWL-S and some related technologies to achieve the publication and discovery of semantic Web services. We believe that the system will bring the client a good experience, because it can produce query results ordered by priority according to the requester's expectations, and its performance is reliable. Currently, the matching performed by the system involves two functional properties of services, input and output. In the future work, we will extend the matching to consider more functional properties of services.

Acknowledgments

The author wishes to thank Professor Yongfan Li for his help and guidance. This work is supported by a science and technology plan project of Hunan province (No. 2014GK3018), a scientific research project of the education department of Hunan province (No. 12C0593), and a project of the key laboratory of basic education informatization technology in Hunan province (No. 2015TP1017).

References

[1] N. Mitra and Y. Lafon, “Soap version 1.2 part 0: Primer (second edition)”, **(2013)**, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.
[2] D. Booth and C. Y. Liu, "Web services description language (WSDL) version 2.0 part 0: Primer", **(2013)**, http://www.w3pdf.com/W3cSpec/WDSL/1/wsdl20-primer.pdf.
[3] L. Clement, A. Hately and C. von Riegen, "UDDI version 3.0.2", http://uddi.org/pubs/uddi-v3.0.2-20041019.htm., **(2013)**.
[4] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider and S. Rudolph, "OWL 2 web ontology language primer", **(2014)**, http://www.w3.org/TR/2009/REC-owl2-primer-20091022/.
[5] "OWL-S 1.2 release", http://www.ai.sri.com/daml/services/owl-s/1.2, **(2014)**.
[6] D. Martin, M. Burstein, D. McDermott, S. McIlraithet, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin and N. Srinivasan, "Bringing semantics to web services with OWL-S", World Wide Web, vol. 10, no. 10, **(2007)**, pp. 243-277.
[7] Y. L. Yao, S. Su and F. C. Yang, "Service matching based on semantic descriptions", in Proceedings of 2006 Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, **(2006)**, pp. 126.
[8] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic web technology", in Proceedings of the 12th International World Wide Web Conference, **(2004)**, pp. 331-339.
[9] N. Srinivasan, M. Paolucci and K. Sycara, “An efficient algorithm for OWL-S based semantic search in UDDI”, in Semantic Web Services and Web Process Composition, Edited J. Cardoso and A. Sheth, Springer Berlin Heidelberg, vol. 3387, **(2005)**, pp. 96-110.
[10] M. Paolucci, T. Kawamura, T. R. Payne and K. Sycara, "Semantic matching of web services capabilities", in The Semantic Web—ISWC 2002, Edited I. Horrocks and J. Hendler, Springer Berlin Heidelberg, vol. 2342, **(2002)**, pp. 333-347.
[11] T. Di Noia, E. Di Sciascio, F. M. Donini and M. Mongiello, "Semantic matchmaking in a P-2-P electronic marketplace", in Proceedings of the 18th Annual ACM Symposium on Applied Computing, **(2003)**, pp. 582-586.
[12] N. Srinivasan, M. Paolucci and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput", in Proceedings of the first international workshop on Semantic Web Services and Web Process Composition, **(2004)**, pp. 75-86.
[13] M. Paolucci, T. Kawamura, T. R. Payne and K. Sycara, "Importing the semantic web in UDDI, in Web Services, E-Business, and the Semantic Web”, Edited C. Bussler, R. Hull and S. McIlraith, Springer Berlin Heidelberg, vol. 2512, **(2002)**, pp. 225-236.
[14] U. Aguilera, J. Abaitua, J. Díaz, D. Buján and D. L. deIpiña, "A semantic matching algorithm for discovery in UDDI", in Proceedings of 2007 International Conference on Semantic Computing, **(2007)**, pp. 751-758.
[15] R. Akkiraju, R. Goodwin, P. Doshi and S. Roeder, "A method for semantically enhancing the service discovery capabilities of UDDI", http://www.isi.edu/info-agents/workshops/ijcai03/papers/Akkiraju-SemanticUDDI-IJCA%202003.pdf. **(2013)**.
[16] T. Qiu, L. Li and P. Li, "Web service discovery with UDDI based on semantic similarity of service properties", in Proceedings of the Third International Conference on Semantics, Knowledge and Grid, **(2007)**, pp. 454-457.
[17] J. Colgrave and K. Januszewski, "Using WSDL in a UDDI registry, version 2.0", http://xml.coverpages.org/UDDI-WSDL20030319.pdf., **(2013)**.
[18] U. Bellur and R. Kulkarni, "Improved matchmaking algorithm for semantic web services based on bipartite graph matching", in Proceedings of 2007 IEEE International Conference on Web Services, **(2007)**, pp. 86-93.
[19] P. B. Fu, S. S. Liu, H. R. Yang and L. H. Gu, "Matching algorithm of web services based on semantic distance", in Proceedings of the 2009 International Workshop on Information Security and Application, **(2009)**, pp. 465-68.
[20] H. R. Yang, S. S. Liu, P. B. Fu, H. H. Qin and L. H. Gu, "A semantic distance measure for matching web services", in Proceedings of 2009 International Conference on Computational Intelligence and Software Engineering, **(2009)**, pp. 1-3.
[21] G. Z. Wang, D. H. Xu, Y. Qi and D. Hou, "A semantic match algorithm for web services based on improved semantic distance", in Proceedings of 4th International Conference on Next Generation Web Services Practices, **(2008)**, pp. 101-106.
[22] T. A. Farrag, A. I. Saleh and H. A. Ali, "Semantic web services matchmaking: Semantic distance-based approach", Computers and Electrical Engineering, vol. 39, no. 2, **(2013)**, pp. 497-511.
[23] V. Haarslev, R. Möller and M. Wessel, "Racerpro user's guide version 2.0", **(2013)**, http://racer.sts.tuhh.de/Racer-2.0/users-guide-2-0.pdf.
[24] "An open source implementation of OASIS's UDDI v3 specification", http://juddi.apache.org/, **(2013)**.

Author

Li Peng, received the B.E. degree from Hunan University in 1996, and the M.S. degree from Hunan University in 2013. He is currently a lecturer in School of Information Science and Engineering, Hunan First Normal University, China. His research interest involves Web service, the semantic Web, computer network.