Web Service Discovery and Dynamic Invocation 
based on UDDI/OWL-S

JianJun Yu\(^1\) and Gang Zhou\(^2\)

\(^1\) National Laboratory of Software Development Environment, Beihang University, Xueyuan Road No.37, Beijing, China, yujj@nlsde.buaa.edu.cn
\(^2\) gzhou@nlsde.buaa.edu.cn

Abstract. With the development of web technology and e-business, the Web Services paradigm has been widely accepted by industry and academic research. More and more web applications have been wrapped as web services, which triggers the change of research focus from finding and integrating them to maximizing reuse. Usually a service requestor retrieves (web) service advertisements out of a UDDI registry based on keywords, like in search engines, which inevitably brings the difficulty of discovering the most appropriate service from a massive number of existing services. This paper briefly introduces a new mechanism to discover and match services in a more fine-grained manner by taking advantage of UDDI and OWL-S. UDDI is used to discover approximate services syntactically by adding new elements and an API. OWL-S is used to match the exact service semantically using ontologies and inference. Finally this paper draws out a framework for discovering and invoking services dynamically.

1 Introduction

With the development of web technology and e-business, the web service has been widely accepted by industry and academic research as a distributed and loose-coupled component which is able to realize sharing and interchanging resources on web.

On the other hand, these services are universally distributed and of various kinds whilst with quite a diverse description. If there lacks a universal description, discovery and integration mechanism, certainly it’s difficult to find and invoke the exact service. Therefore, how to find and integrate the exact service from massive services has become the current focus of research on service matching.

The service registry is an indispensable component in Service-Oriented Architecture. It provides description, discovery and integration of services through uniform API interface on the basis of UDDI protocol. Nevertheless, based on

\(^*\) This research was supported by NSFC Grant 60403003 and FANEDD Grant 200241 of China
syntactical structure, UDDI aims to make searches on keywords and classification. Thus, wide application of Web Services may be negatively affected for failing to well match the requirements from customers.

UDDI can hardly meet the requirement of automatic service discovery due to lack of semantic description of web service. While in OWL-S, ServiceProfile gives the semantic description of web service which helps to match the service precisely via ontology and inference mechanism. But the disadvantage is obvious with little supporting software.

By combining these two service discovery mechanisms mentioned above, this paper aims to use UDDI to make extensive inquiry and OWL-S to match service accurately in order to retrieve the exact service. This new mechanism provides a method to aggregate and classify massive loose-coupled and heterogenous services on Web. It enables the service provider to publish services more easily and the service requestor to discover, integrate and invoke services more quickly and succinctly.

This paper is organized as follows:

The second part introduces the mechanism of UDDI discovery as well as its advantages and disadvantages and then adds new elements and API to enhance service description and discovery. The third part introduces how ServiceProfile matches and infers the exact service. The fourth part brings in the framework of service discovery and dynamic invocation as well as its stereotype. The fifth part gives the related work. The final part is conclusion.

2 UDDI Discovery

UDDI [6] registry can be divided into two types: public UDDI registry and private UDDI registry. Corporations like IBM, Microsoft, SUN, SAP, etc. have established public UDDI registry for commercial purpose. Detailed information can be found in [7].

Private UDDI registry is generally applied to service integration within corporations which always built on open source package that enables service provider to have more control over service description, security and collaboration. UDDI registry discussed in this paper will be based on private UDDI registry.

UDDI defines four core data structures as service metadata, which are illustrated in Fig.1:

As is shown in Figure 1, BusinessEntity describes specific information about the service provider, including the name of corporation, contact information and taxonomy. One BusinessEntity can contain one or more BusinessService which advertises service capability with one or more BindingTemplate. BindingTemplate contains the service access point and refers to a series of tModel. tModel depicts the criteria and classification of a service. Services with the same function refer to the same tModel. Service requestor uses inquiry API to search tModel or BusinessEntity and obtain relevant registry information. There are three patterns for discovering web services: the browse pattern, the drill-down pattern
The core data structures and APIs give a universal method for service publishing and inquiry. We can use IBM uddi4j to interact with different UDDI registries.

Although UDDI gives a universal API to publish and inquire service information, its disadvantages are obvious:

1) Similar to a search engine, UDDI uses syntactic structures that lacks relevancy and semantic classification of services. However, in many cases, services advertised at UDDI are not exactly the same as their actual functionality or they are identified by abbreviation and synonymous words. Therefore, searching based on syntax cannot discover the right service easily.

2) In UDDI, there lacks sufficient metadata for service description to support automatic service discovery. As for a program, it will be easier to understand the capability and performance of the services by retrieving both the functionality and non-functionality information from UDDI registry. Apparently, current structures of UDDI need improved and added more description information to support further service discovery.

3) More APIs are needed in UDDI to get the specified information. Now, service requestor needs to call find_tModel, get_tModel, find_bindingTemplate and get_bindingTemplate APIs to get the service access point. This process is not necessarily and reduces the search efficiency. So, new APIs should be defined to get the specified information directly. For example, service requestor can get the access point from tModel through find_wdsl_with_tModel API.

4) UDDI registry accepts service information passively, which means that if a service changes and so does access point without updating service description...
in the registry, service requestor will probably use false information and fail to invoke the service.

Therefore current UDDI cannot entirely meet the requirement of exact web service discovery and dynamic service invocation. Service requestor hopes that they can retrieve the right service and satisfy reliable and high-efficient invocation by merely describing their needs and requirements. When one service is down, service requestor can choose another service to guarantee uninterrupted service invocation.

This paper improves core data structures and adds one API to enhance the veracity and efficiency of UDDI inquiry:

1) Add ServiceProperty in BusinessService as non-functional description and Add ServiceInterface in tModel as functional description. ServiceProperty mainly includes the name, type and value of the property. The data structure is as follows:

```xml
<ServiceProperty PropertyName="" PropertyType="" PropertyValue="" />
```

BusinessService can have one or more ServiceProperty which are used to describe the non-functionality of the service, such as the response time of the service, the rate of CPU occupancy and the maximal process count of the service.

ServiceInterface is used to describe the functionality of a service. The data structure is as follows:

```xml
<ServiceInterface>
  <ServiceMethod name="">
    <Parameters>
      <Param name="" type="" />
    </Parameters>
    <return name="" type="" />
  </ServiceMethod>
</ServiceInterface>
```

ServiceInterface has one or more child nodes named ServiceMethod which is used to describe the name of method, the name and type of input and output.

2) Add find_wsdl_with_tModel API. Retrieve the access point of the service by inquiring the name of tModel or matching the ServiceInterface structure. The data structure is as follows:

```xml
<find_wsdl_with_tModel>
  [<tModelName/>[<tModelName />]...]
  [ServiceInterface]
</find_wsdl_with_tModel>
```

3) Add a UDDI Monitor used to monitor the service state at real-time in uddi4j. It can route to the most appropriate service according to the service state.
3 OWL-S Matching

Semantic Web Service mainly use OWL-S [8] as its protocol which enables web services to be machine understandable, including service dynamical discovery, invocation, composition and monitoring. OWL defines three upper ontologies, including ServiceProfile, ServiceModel and ServiceGrounding. ServiceProfile defines what the service does; that is, it gives service description needed by a service requestor to determine whether the service meets his requirement. ServiceModel defines how the service works; that is, it describes the workflow and possible execution paths of the service. ServiceGrounding specifies the details of how to access a service. Typically it will specify a communication protocol and service-specific details such as port numbers used in contacting the service.

ServiceProfile describes three types of service information: service provider information, functionality and non-functionality. The service provider information consists of contact information about the service. The functionality of the service is expressed in terms of inputs, outputs, preconditions and effects. These features assist when reasoning about several services with similar capabilities. Non-functionality of the service means service QoS and service state, including service response time, service process ability. [9] gives detailed information about service non-functionality.

In OWL-S, service matching is based on ServiceProfile using ontologies. Service provider and service requestor refer to OWL ontologies, and then the matching engine can perform inferences to recognize semantic token despite different syntax and model between service provider and service requestor. [1] gives matching algorithm mainly based on input and output of the service. Just like UDDI, ServiceProfile also provides the metadata for services. However, since OWL-S language is not mature and ontology definition is not authoritative, there is inevitably few supporting software and practical application.

This paper uses UDDI2SWSPro(based on OWL-S/UDDI MatchMaker [11]) to create ServiceProfile for the subsequent service matching based on current UDDI. If ServiceInterface is published in UDDI, this structure will be used to create functionality in ServiceProfile. If not registered, WSDL is used to create functionality in ServiceProfile. The contact information in ServiceProfile is created by Contacts in UDDI. The non-functionality in ServiceProfile is created by ServiceProperty. To be more specific, corresponding relation between ServiceProfile and UDDI is as Table 1:

4 Framework and Prototype

The framework mainly uses improved mechanism mentioned above to realize service discovery and matching, and then puts forward a prototype to support service dynamic invocation based on the result services founded using UDDI and OWL-S.

User’s requirements are classified into three types: 1) The name of the service; 2) The functionality of the service; 3) The non-functionality and QoS of
<table>
<thead>
<tr>
<th>ServiceProfile Element</th>
<th>UDDI Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor:name</td>
<td>Contacts:PersonName</td>
</tr>
<tr>
<td>Actor:phone</td>
<td>Contacts:phone</td>
</tr>
<tr>
<td>Actor:fax</td>
<td>/</td>
</tr>
<tr>
<td>Actor:email</td>
<td>Contacts:email</td>
</tr>
<tr>
<td>Actor:physicalAddress</td>
<td>Contacts:address</td>
</tr>
<tr>
<td>Actor:WebURL</td>
<td>discoveryURLs</td>
</tr>
<tr>
<td>serviceName</td>
<td>BusinessService:name</td>
</tr>
<tr>
<td>intendedPurpose</td>
<td>ServiceProperty</td>
</tr>
<tr>
<td>textDescription</td>
<td>BusinessService:descripion</td>
</tr>
<tr>
<td>requestedBy</td>
<td>Contacts</td>
</tr>
<tr>
<td>providedBy</td>
<td>Contacts</td>
</tr>
<tr>
<td>serviceType</td>
<td>ServiceProperty</td>
</tr>
<tr>
<td>serviceCategory</td>
<td>CategoryBag</td>
</tr>
<tr>
<td>communicationThru</td>
<td>ServiceProperty</td>
</tr>
<tr>
<td>qualityGuarantee</td>
<td>ServiceProperty</td>
</tr>
<tr>
<td>qualityRating</td>
<td>ServiceProperty</td>
</tr>
<tr>
<td>Input</td>
<td>ServiceInterface or WSDL</td>
</tr>
<tr>
<td>Output</td>
<td>ServiceInterface or WSDL</td>
</tr>
<tr>
<td>Precondition</td>
<td>/</td>
</tr>
<tr>
<td>Effect</td>
<td>/</td>
</tr>
</tbody>
</table>

Table 1. Translation between UDDI and ServiceProfile

the service. This paper uses service name to achieve service discovery, input and output class in ServiceProfile to achieve service functional matching, and ServiceProperty to achieve service non-functional matching.

To get the exact service, the following steps should be adhered to:

1) Use improved UDDI to inquire about services extensively, get their UDDI entries and store them in UDDIEntry DB, retrieve their WSDL documents and store them in WSDL DB. Private UDDI receives the registration of services and retrieves entries from public UDDI registries at the same time using subscription API. The functionality of the service is described by ServiceInterface in tModel, and non-functionality of the service is described by ServiceProperty in BusinessService. The service name is used to inquire tModel and BusinessService. Therefore the inquiry will be mapped to two API inquiries: find_service and find_wSDL_with_tModel.

2) Use UDDI2SWSPro to create contact information, non-functionality and functionality in ServiceProfile from UDDIEntry DB and WSDL DB.

3) Use matching engine to match functionality and non-functionality of the services, and then rank the founded services according to the matching degree.

The system architecture is described as Fig.2:

The resulting web services are stored in UDDI Monitor. UDDI Monitor is used as a service subscriber and service dispatching center. ServiceSubscription receives all updated web services information from UDDI registry immediately
which may have already disabled or cannot satisfy actual needs in function. All services received from ServiceSubscription will be put into a service pool, which is classified into active service pool and inactive service pool. Active service pool caches all the services in operation; inactive service pool stores disabled services or services that should be activated. The ServiceListener manages the service pools by polling service state information periodically. If the service is disabled, it will be put into inactive service pool. If the current state of the service cannot satisfy the user’s requirement, for example, the load is too heavy, then another available service in active pool will be released to respond the client request. The priority of the current service will decline and the service will be put to the end of the active queue. If a disabled service becomes activated, it will be released from inactive queue and put to the end of the active queue.

When a service is invoked, it will be changed the previous access point by ServiceProxy and routed to the ServiceSelector. ServiceSelector decides whether the first service in the current active service pool satisfies the requirement. If it satisfies the requirement, it will be taken out and invoked; if not, the second service will be taken out instead till the requirement of the user is satisfied.

For example, service provider transfers the service to another server, and changes its service access point in UDDI registry. ServiceSubscription receives the notification immediately, and then changes the access point of the exact service in service pool. Then the next request will be routed to the new service address dynamically. During the process of invocation, if the current web service
breaks down, the ServiceListener will mark this service as an inactive service, and then put it to the end of the inactive queue. It will not be marked as an active service until it recovers to normal state. If the capability of current service is poor, the ServiceSelector will choose another suitable service to invoke.

The framework described as Fig. 3:

![Fig. 3. UDDI Monitor framework](image)

5 Related Work

Currently most of work distributed to service finding uses UDDI or OWL-S. But the literature on Web Services and Semantic Web is not abundant for a specified area, that is, UDDI basically uses syntactic service discovery and OWL-S uses semantic service discovery. They should be merged together to meet the user’s requirement using enhanced descriptions.

[4] gives an enhanced UDDI to record user’s defined properties. [5] gives an active UDDI allowing the extension of UDDI API to enable fault-tolerant and dynamic service invocation. They all should define more metadata and APIs to support UDDI translated to ServiceProfile and make service discovery more accurate.

[1] gives semantic matching of web services capability which shows how service capabilities are presented in the ServiceProfile and how a semantic matching between advertisements and requests is performed. [2] presents basic requirements for service discovery and evaluates current web service technology (WSDL, UDDI, DAML-S) that is used to address these requirements. [3] gives semantic API matching for automatic service composition. In this paper, these works are analyzed and summarized to give a universal solution for service matching.
6 Conclusion

Web service discovery and matching is an important aspect in Web Services oriented technology. Current available industry standards such as UDDI and WSDL can only support syntactic service discovery which will not be sufficient for exact service matching. OWL-S gives a semantic service matching which can find the service exactly. However, OWL-S is still in its infancy and a lot of work has to be done in order to overcome its limitations and problems.

This paper combines UDDI with OWL-S to give a solution for service discovery and matching, which uses UDDI to discovery extensively and OWL-S to match service accurately. When the result services are invoked, this paper gives a framework and prototype extended from previous work in [10] to assure the services uninterrupted invocation at runtime.

References

8. OWL-S 1.1. http://www.daml.org/services/owl-s/1.1/.