Two Extensions of Service Description to Enhance the Scalability of SOA *

Zhifeng Gu
DCST, Tsinghua University
Beijing, 100084, China
gzf@keg.cs.tsinghua.edu.cn

Juanzi Li
DCST, Tsinghua University
Beijing, 100084, China
ljz@keg.cs.tsinghua.edu.cn

Ruobo Huang
CSDL, IBM
Beijing, 100085, China
huangrb@cn.ibm.com

ABSTRACT

Service description is the underlying basis of different service tasks. In this paper, we proposed two service description extensions to enhance the scalability of SOA. First we proposed multi-input operation to increase the flexibility of service interface. Second, we encode service dependency into service description. Service dependency is not only an empirical knowledge for reliable service composition, but it is also a distributed service recommendation and discovery mechanism, which is more scalable than centralized service discovery mechanisms such as UDDI.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Service Description, Extension

Keywords

Service Description, Multi-input Operation, Service Dependency

1. INTRODUCTION

Services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Internet. To enable highly flexible and reliable service composition technology, there are many tasks to be concerned, including service description, service discovery, service execution, transaction, and security, etc, among which, we think that service description is the underlying basis of all the other tasks.

*The paper is supported by the IBM SUR project (Service Science and Technology).

INFOSCALE 2007, June 6-8, Suzhou, China

DOI 10.4108/infoscale.2007.884

In the research community, many works have tried to enrich the expressiveness of service description, for example, the semantic extensions [1] [5] and the behavioral extensions [3] [2]. In this paper, we proposed two extensions of service description to enhance the scalability of SOA. The first extension is multi-input operation. We allow one operation to support different types of messages. This concept is quite similar to the concept of method overload in programming languages. The second extension is adding service dependencies to service description. The concept of dependency is similar to that of service dependency graph (SDG) [4]. However, in SDG, dependencies are defined on the abstract level and are implied by service descriptions, while in our work, dependency is defined among service instances and needs explicit declaration. We think that our extensions will benefit the scalability of SOA through three aspects:

1. Enhancing the adaptability of service operations. When composing services, if the type of the required message does not match the type of the provided message, the traditional solution is to create a message adapter as a mediator. However, without a deep understanding of the schema of the required and the provided messages, which might be very complicated and domain-specific, it is not easy to generate a message adapter correctly. Although there are many tools aiming at XML message mapping, it is still a time-consuming and error-prone task to create message adapters. So we think the service provider should make a service supporting the mainstream schemata in its domain, in order to avoid message type mismatch in the composing process.

2. Applying empirical knowledge to service composition. Web services are independent and autonomous systems. It is hard to test and debug service-based applications in a cross-enterprise and cross-organization environment. It is hard to make sure two services that have never been tested together can be composed and work properly. Even if all the message types are exactly matched, it is still very likely that two services will fail to work together due to different implementations. We define service dependency as the relation among the service instances. Dependencies can either be provided by the service provider, or be collected by the service composer. Service dependency is a kind of empirical knowledge. When composing services, service composer can use service dependency as guidance for service discovery and selection. Also, service dependency can be used to

evaluate the reliability of existing compositions.

3. Providing a distributed service recommendation and discovery mechanism. As well as to increase the reliability of service composition, service dependency can also be considered as a distributed service recommendation and discovery mechanism. Service dependency is very similar to "Related Links", which is a very common concept in web pages. It contains related services that can work with this service, and produce more interesting and value-added results. Comparing with centralized service discovery mechanisms such as UDDI, service dependency is more scalable owing to its distributed natures.

2. MULTI-INPUT OPERATION

In this section, we introduce our first extension, multi-input operation.

2.1 Motivation

Service designers have to face difficult situations: there are several well-known schema in his domain; new business partner uses another data schema; customers always require support for various data schemata. Simply speaking, there is always a requirement that a service need to deal with heterogeneous schemata with similar data semantics. It is a design philosophy for service designers whether,

1. A service interface should directly support different types of messages
2. Or, a service interface should be kept simple and clean, while leaving the message transforming job to message adapters.

We argue that, from the perspective of service composer, the first method is better. Regarding the second method, that who will design and implement the adapter remains a problem. Normally there are three candidates to design and implement the adapter:

1. The service composer. It is not an ideal choice, since it will take a lot of time to understand the data semantics in a complicated data schema so as to design the message adaptor. And unfortunately, the adapter created by service composer is usually error-prone.
2. The third party. It is better than the service composer. But there will be some problems when service composer is required to deal with third party adapters. For example, how to find a good adapter? If the third party adapter is an online service and the data is confidential, how can we make sure the third party is trustworthy?
3. The service provider. We think this is the most reasonable choice. A service provider is usually an expert in his domain, so it will be easy for the service provider to develop his own message adapters. Also, a service provider knows well where to purchase high quality message adapters from third parties. Furthermore, a service provider can choose not to use message adapter at all, and directly map messages to his internal objects.

To summarize, we think a service should be adaptive to different message types. A service can be considered as the mother board of a PC. If there are only PCI slots on this mother board, it will be very inconvenient and unreliable to assemble a computer. For example, the first step, you need to install an IDE hard disk. It is obviously impossible to understand the PCI specification and the IDE specification so as to make an adapter. The only feasible solution is to buy an IDE/PCI adapter. However, if unfortunately the bought adaptor is incompatible with the mother board or the hard disk, unexpected data-loss or blue screen may be incurred. A mother board with IDE interface integrated will partially solve this problem, since the integrated IDE interface should have been tested to be fully compatible with the mother board. Then, the rest of the problem becomes: how can we make the hard disk and the integrated IDE interface compatible with each other? This is really a problem, and will be discussed in section 3.

2.2 The Problem and Our Extension

We give an example to show what the problem is. Suppose we are designing a service that processes news materials. We are facing two schemata in this domain, NewsML [1] and XinhuaML [2]. In order to support both of these two schemas, there are two approaches.

The first approach is to define two operations to deal with NewsML and XinhuaML respectively. As WSDL allow operation overload, the names of the two operations may be the same. However, we think this is not a good way, since these two operations have the same functional semantics. It is better to aggregate these two operations into one. The second approach is to defined a new complex type, which can contain an element being either NewsML or XinhuaML. Using this new defined type, we can define a unified operation handling both NewsML and XinhuaML. However, this approach does not explicitly address that this service is adaptive to both NewsML and XinhuaML. Although it is possible to exploit the adaptability of this service through analyzing the schema, it is better to address the adaptability explicitly.

To solve this problem, we think one operation should support different inputs. Our approach is illustrated in list 1. we allow the *input* element to have several *option* subelements. Each *option* is an alternative input of the operation. Although this extension seems redundant to operation overload, we think it makes the semantics of the operations more clear.

3. SERVICE DEPENDENCY

3.1 Motivation

We have mentioned earlier that even if the message types are matched when composing services, it is still unsafe to say that the composition will work without sufficient testing. It is due to two reasons:

1. There is no well-developed method to formally describe data semantics. Currently, data semantics is usually specified with natural languages. As services

[1] `http://www.newsml.org`

[2] `http://news3.xinhuanet.com/it/2005-01/21/content_2491428.htm`

```
<import namespace="http://newsml.org/"
  location="NewsML.xsd"/>
<import namespace="http://xinhuaml.gov/"
  location="XinhuaML.xsd"/>
<message name="NewsMLResquest">
  <part name="news" element="nl:NewsML">
</message>
<message name="XinhuaMLResquest">
  <part name="news" element="xl:XinhuaML">
</message>
<portType>
  <operation name="ProcessNews">
    <input>
      <option message="NewsMLResquest">
      <option message="XinhuaMLResquest">
    </input>
    ...
  </operation>
</portType>
```

Listing 1: Our approach: Multi-input operation

are developed and deployed across companies and organizations, different service providers may have different understandings to the data semantics carried by the XML data.

2. Even if the service providers have the same understanding, there may still be some differences introduced in the development process.

So it is hard to guarantee that two services can work together when they have not been tested together. Besides developing more powerful formal methods to eliminate misunderstanding on abstract description and differences on concrete implementation, we can make use of empirical knowledge to improve the reliability of service composition. In our work, service dependency is a relation between two service instances. If service A can accept and fully understand the output of service B, then, we say there is a dependency from B to A. We think instance level dependency is necessary because:

1. As discussed above, due to the misunderstanding on abstract description and the differences on concrete implementation, it cannot be guaranteed that two service instances can work together without testing.
2. The back-end data source maybe incompatible across different service instances. For example, an ID generated by a service of company A is very likely to be invalid in the scope of the services of company B.

Of course, real world applications are very complicated. A service dependency working in some environments may be broken in other environments. Therefore, we think that service dependency can only partially solve the problem, although it is really a useful empirical knowledge that can be made use of in service discovery and composition.

Service dependency can be specified in two ways. One is specifying service dependencies together with service description; the other is specifying service dependencies separately in a user-manageable knowledge base. The first way can be used by service providers to encode dependency info into service description. The second way is service composer-oriented. Service composer can extract dependencies from a successful composition, and store these dependencies into his own knowledge base.

Thus, service providers can publish a set of service dependencies together with the service. Service composers can gather and maintain service dependencies in his own knowledge base. Furthermore, the knowledge base can also be published as a service for public and commercial use. In section 2, we leave a question regarding the incompatibility between the integrated IDE interface and the hard disk Our answer is to ask the vendor of the mother board for a list of compatible hard disks, which is quite similar to the concept of service dependency.

3.2 A Side Product

We noticed that, as a side product, service dependency can be considered as a distributed service recommendation and discovery mechanism. Service dependency is very similar to "Related Links", which is a very common concept on web pages, and is a very useful feature to browse the Internet. Service dependency will bring great convenience to service provider and service composer.

From the perspective of service composers, given one service, they will be able to find the services producing interesting and value-added results, and the services producing the necessary messages to invoke this service. From the perspective of service providers, they will be able to advertise their own or their partner's services through cross dependencies.

Compared with centralized service discovery mechanisms such as UDDI, service dependency is more scalable owing to its distributed natures. As services are maintained by service providers independently, the maintenance cost is distributed to every service provider. However, as the maintenance cost of centralized service discovery mechanisms is usually centralized into one or several organizations, it will become a bottle neck when the number of services gets large.

4. IMPLEMENTATION

In this section, we introduce how we extend WSDL to apply our extensions.

4.1 WSDL Extension

We extend the schema of WSDL to apply our two extensions. Generally speaking, a WSDL document can be logically divided into three parts: XML schemata, abstract descriptions and concrete descriptions [6], as shown in figure 1. In order to fully re-use XML data schemata and abstract descriptions. XML data schemata should be defined first. In abstract descriptions, XML data schemata should be imported instead of being re-defined or being copied. Similarly, concrete descriptions should also refer to abstract descriptions by importing them. In another word, XML data schemata and abstract descriptions should be fully shared, and relatively stable, while concrete descriptions may vary from one service provider to another.

Multi-input operation is an extension of abstract descriptions. We modify the schema of WSDL to enable multi-input operation. We add an element named *option* under

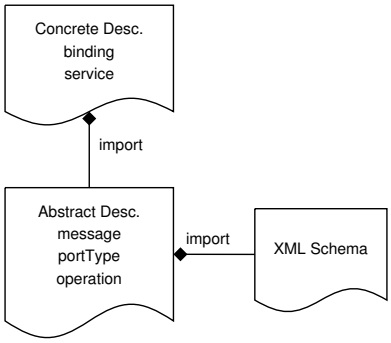

Figure 1: Logical structure of a WSDL document

input. Each *option* is associated with a message by the *message* attribute. An example can be found in listing 1.

Service dependency is an extension of concrete descriptions. The WSDL specification supports extensibility elements under certain elements. As shown in list 2, we put dependency declaration under the *port* element, which usually contains a URI that indicates the endpoint of a service instance. Note that service dependency is not required to be specified with the service. Each service dependency is self-contained and can be put anywhere. Also there are some other places in WSDL reserved for extensibility, for example, the end of the *definitions* element. Putting the related dependencies under the *port* element can reflect the logical relation between the service instance and the dependencies.

```
<service name="Service1">
  <port name="port1" binding="tns:b1">
    <soap:address location=
        "http://company1.com/Service1"/>
    <dependency type="succeeded">
      <src type="webservice">
        <port>http://company1.com/
        services/Service1</port>
        <operation>Operation1</operation>
        <message>Response1</message>
        <part>Part1</part>
      </src>
      <des type="webservice">
        <port>http://company2.com/
        services/Service1</port>
        <operation>Operation1</operation>
        <message>Response1</message>
        <part>Part2</part>
      </des>
    </dependency>
  </port>
</service>
```

Listing 2: Specifying service dependency in WSDL

Now let's turn to how to specify service dependency. The example is given in list 2. Simply speaking, a service dependency is a connection between two data elements, which share the same data type, but belong to different endpoints. According to web services, the data element will be a *part* in *message* definition. The definition of service dependency mainly consists of two parts: the *src* element and the *des* element. The *src* element specifies the source data element of the dependency. Correspondingly, the *des* element specifies the destination data element of the dependency. The *type* attribute of *dependency* element has three possible values: *succeeded*, *mandatory* and *failed*. The meaning of these two values is self explained by the literal meaning. When the value of *type* is *succeeded*, the dependency is much more like a service recommendation than a "dependency". When the value of *type* is *failed*, it is a negative dependency, and should never be triggered in service composition. If there are more than one *mandatory* dependencies, then any of them will be acceptable.

The *dependency* element in list 2 is a very basic example of service dependency. In practice, we need more features to ease the specification of service dependency and to enhance expressiveness of service dependency. In our work, we have tried to add some extra features into the definition of service dependency. Due to the limit of space, we do not expand these features here.

4.2 A Parser Recognizing our Extensions

We have implemented a simple parser to recognize our extended version of WSDL. The parser is based on WSDL4J. However, it is still a very basic prototype. Many features need to be implemented.

5. CONCLUSION

In this paper, we proposed two service description extensions to enhance the scalability of SOA. Multi-input operation is proposed to improve the adaptability of service. However, the underlying idea we want address is that, for service composers, what is important is that they should avoid designing and implementing message adapters. Service composers should always try to find the services that share the same schemata, and compose these services according to the design goals. Service dependency is similar to "Related Links" in web pages. Although service dependency can not fundamentally solve the interoperability problem among services, it is really a useful empirical knowledge that can be made use of in service discovery and composition.

6. REFERENCES

[1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web service semantics - WSDL-S, April 2005.

[2] D. Berardi. *Automatic Composition Services: Models, Techniques and Tools*. Ph.d, University of Rome, 2005.

[3] D. Beyer, A. Chakrabarti, and T. A. Henzinger. Web service interfaces. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 148–159, New York, NY, USA, 2005. ACM Press.

[4] Q. A. Liang and S. Y. W. Su. AND/OR graph and search algorithm for discovering composite web services. *International Journal of Web Services Research*, 2(4):48 – 67, 2005.

[5] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. METEOR-S Web service annotation framework. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 553–562, New York, NY, USA, 2004. ACM Press.

[6] S. Tyagi. Patterns and strategies for building document-based web services, Sep 2004.