Advances in Computational Research
ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 4, Issue 1, 2012, pp.-78-82
Available online at http://www.bioinfo.in/contents.php?id=33

DYNAMIC INVOCATION OF WEB SERVICES

TERE G.M.[1]*, JADHAV B.T.[2] AND MUDHOLKAR R.R.[3]

[1]Department of Computer Science, Shivaji University, Kolhapur, Maharashtra - 416004, India
[2]Department of Electronics & Computer Science, Y.C. Institute of Science, Satara, Maharashtra - 4, India
[3]Department of Electronics, Shivaji University, Kolhapur, Maharashtra - 416004, India
*Corresponding Author: Email - girish.tere@gmail.com

Received: February 21, 2012; Accepted: March 06, 2012

Abstract- When we use web service, we should add it in the web reference and then call its methods statically. This way of calling web services has lots of limitations. In order to take maximum advantage of the flexibility and power of Web services, the user must be able to dynamically discover and invoke a Web service. We need to dynamically discover and invoke the service because the information returned from web services can be used by heterogeneous applications which are executed on different machines. As our business world is dynamic and heterogeneous, a client often needs to invoke an unfamiliar web service at run time. However, current web services technology pays little attention to this issue. In this paper, we propose a framework for a client to dynamically invoke web services. The framework can increase the use and reliability of web services invocation in a dynamic, heterogeneous environment. Web Service has been widely accepted by industry. How to find and integrate existing Web Service is a crucial work. Client finds Web Service from UDDI Registry and invokes it directly as described in a contract, web service description language, WSDL. It is difficult for an enterprise user to dynamically invoke the most appropriate Web Service. This paper briefly introduces Service-Oriented Architecture and discusses advantages and disadvantages of UDDI, then puts forward a dynamic Web Service framework that extends the SOA .
Keywords- SOA, Web services, UDDI, WSDL

Citation: Tere G.M., Jadhav B.T. and Mudholkar R.R. (2012) Dynamic Invocation of Web Services. Advances in Computational Research, ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 4, Issue 1, pp.-78-82.

Introduction

With the rapid growth in Internet functionality, distributed computing systems have attracted more attention in the Information Technology world. This has resulted in recent standardization effort of distributed computing architecture, which is known as Service Oriented Architecture (SOA). The Web Service is the main component of this architecture. Some of the challenges in implementing the SOA [1] architecture are maintainability, reliability, and security.
Fig. (1) shows the basic Service Oriented Architecture (SOA) using web services. It contains three main components viz., Service broker, Service provider, Service consumer. The communication between them is achieved by exchanging messages in SOAP form. SOA is the exposure of software resources in the form of services, which can be accessed over a network [6]. When SOA is used for EAI (Enterprise application integration) where diverse applications in an enterprise communicate and collaborate to achieve a business objective, binding between the web services is pre-configured and the interaction is static. For discovering a web service client concerns a service broker. A common service broker is Universal Description, Discovery and Integration, UDDI. UDDI is a platform-independent, extensible markup language, XML, based registry for businesses worldwide to list themselves on the Internet and a mechanism to register and locate web service applications. The UDDI used is private and is accessible to organization, its business partners only

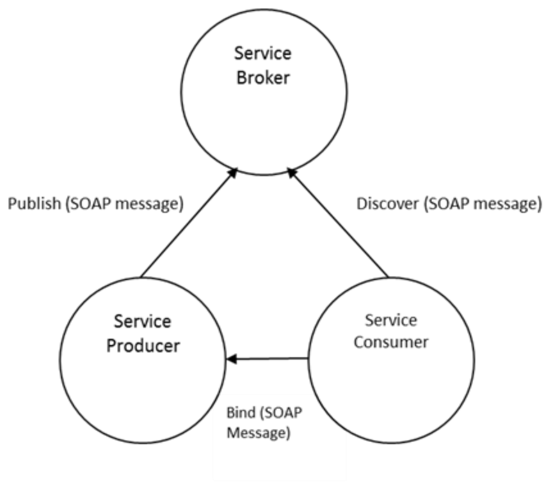


Fig. 1- Basic SOA

Fig. (2) shows implementation of SOA architecture for EAI where static binding between web services is required

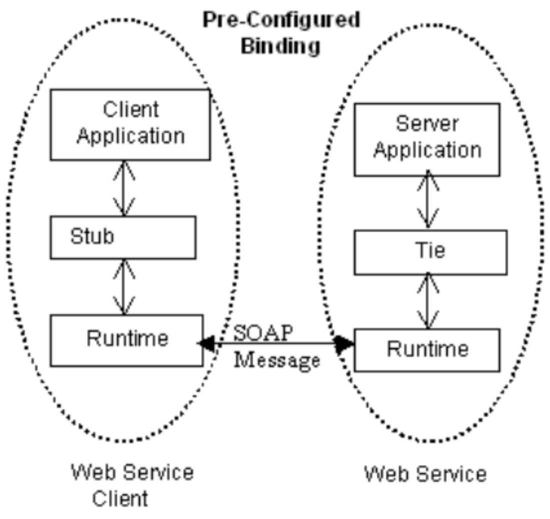


Fig. 2- Static web service interaction

Dynamic Web Service Invocation

We propose to use dynamic 'Web Service Invocation' method to address maintainability and reliability issues without sacrificing the overall system performance. In order to consume a web service, we need to create client stubs from the WSDL description. This can be done through static client or dynamic client [3]. Static Client is a stand-alone program that calls the operations of a web service through a stub, a local object which acts as a proxy for the remote service. Because this stub is created before runtime it is called a static stub. Dynamic Client calls a remote procedure through a dynamic proxy, and object created at runtime that represents the Web service [2]. As businesses become global there is a need for these applications to become available globally [9]. Thus the pre configured binding between web services becomes obsolete. Suppose an application for an online shopping chain accesses a service broker that specializes in shipping. The broker locates services from public UDDI registry that meet certain criteria such as fast delivery time and invokes them at run time. Thus the binding between broker and web services is dynamic. Fig. (3) shows the dynamic binding between client and web services in SOA.

As shown in Fig. (4) UDDI Proxy is added in SOA and acts as a proxy for Web Service [7], which is placed at client side. UDDI Proxy obtains a list of currently usable Web Services from private service registry and accomplishes the work of finding, testing, verification and management of the Web Services through Monitor Service and Dispatch Service inside UDDI Proxy component. Service provider registers Web Service information in public service registry as before

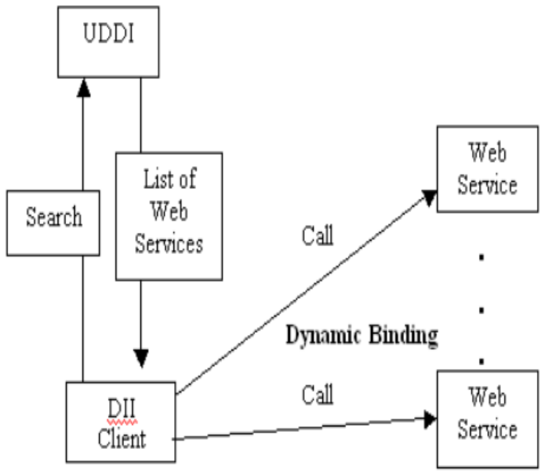


Fig. 3- Dynamic Invocation of multiple web services

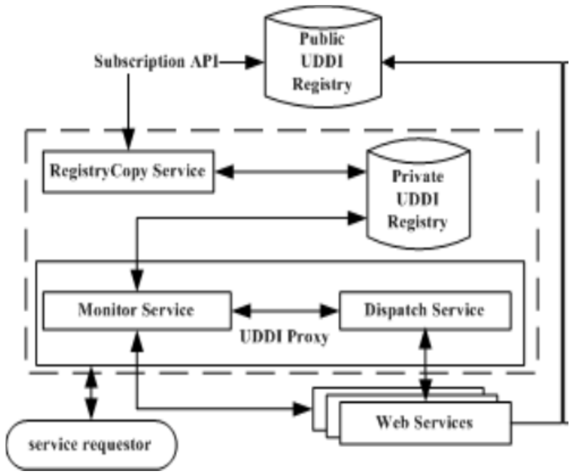


Fig. 4- Web Service dynamic invocation framework

The difference is that service requestor no longer needs to inquire service information in public service registry but only bind service access point provided by UDDI Proxy. There are three components in the UDDI Proxy which are RegistryCopy Service, Monitor Service and Dispatch Service. RegistryCopy Service is closely related to private

service registry. It subscribes Web Service registry information with specific function from public service registry through Subscription API. When public service registry adds, updates or deletes relevant Web Service information, private service registry will receive all of these changes through notification and make changes to corresponding registry information inside it.

Use of dynamically discover and invoke the web service

We will need to discover information about the service from UDDI and read the WSDL implementation file from the service provider and parse it for various information. Every provider creating service need to describe the service in WSDL and every client, after searching the web service need to use the web services as per explained in WSDL [3]. Thus, WSDL is a contract between a server and client. We will have enough information to marshal a SOAP request to the Web service. In both the case to create a web service client we must know the web services and operations to be called beforehand. With the 'Web Service Invocation', a client can call a web service even if the signature of the methods or the name of the service is unknown until runtime. In contrast to a static stub or dynamic proxy client, a 'Web Service Invocation' client does not require runtime classes.

Advantages and disadvantages of static invocation of a Web service versus dynamic invocation

There are actually three different ways to invoke a Web service:

a. static binding
b. dynamic binding
c. dynamic invocation

With static binding compile and bind client proxy at development time. This binding is tightly bound to one and only one service implementation. It provides the fastest performance of the three options, but gives us the least flexibility[4].

With dynamic binding the only thing to compile at development time is the interface to a service type (i.e., the WSDL <portType> definition). At runtime client can bind to any service implementation that supports that <portType>. It generates a dynamic proxy from the service's WSDL <binding> at runtime and casts it to the interface. From a developer's point of view, this approach is as easy to use as a static proxy. We need to retrieve the WSDL and do some runtime compilation of the WSDL, so the initial connection takes a bit longer (a few hundred milliseconds), but once the binding is complete, performance is equivalent. Advantage of this process is we can enjoy a lot of flexibility. Using this technique, application can connect to any number of different service implementations without modification. It can automatically handle changes to the underlying protocols. It doesn't automatically handle changes to the service signature, though. With dynamic invocation, don't compile anything at development time. Instead do everything at runtime. The application retrieves and interprets the WSDL at runtime and dynamically constructs calls. It gives us the most flexibility, but also requires a much more sophisticated client. Obviously, there's a bigger hit in terms of performance, which occurs on each invocation. The advantage of using 'Web Service Invocation' is that we need not to have generated the stubs before runtime. This allows us to generically invoke services that may not know about at run time.

The server implementation

We developed a Web service using a simple Java class with a method that takes two integers as parameters and returns an integer which is the multiplication of the parameters. This class will, of course, be exposed as a Web service, and that service will need to be published to UDDI

The client

Let us discuss how to dynamically discover and invoke the service. First we need to discover information about the service from UDDI. Second, we need to read the WSDL implementation file from the service provider and parse it for various information. In this way we will have enough information to marshal (encode) a SOAP request to the Web service implementation using Axis.

In the client code this is accomplished using the open source packages uddi4j, wsdl4j, and Apache Axis[5]. We have used uddi4j to browse the UDDI registry as it provides an API which allows the user to make inquiries and publish to any UDDI 2.0 registry. The wsdl4j package will be used to programmatically represent the elements of a WSDL file. Then we will use Axis to actually make the SOAP request to the server and wait for the response.

Finding the Web service in UDDI

In order to dynamically invoke your Web service we need to know four things. We must know the inquiry URL of the UDDI registry, the publish URL of the UDDI, the name of the business entity, and the name of the business service. This information can be found in following code

```
public class DynamicInvoke {

  private String uddiInquiryURL =
      "http://localhost:80/uddisoap/inquiryapi";
  private String uddiPublishURL =
      "http://localhost:80/uddisoap/publishapi";
  private String businessName = "ABCD Ltd.";
  private String serviceName =
      "WebMathService";
…
}
```

This locates the UDDI and once that is found, proper business service can be selected. We use the UDDI URLs to create a proxy to the UDDI registry which can be used to query the registry to find the business service. This is done as explained in following code.

```
 // create a proxy to the UDDI
 UDDIProxy proxy = new UDDIProxy(
        new URL(uddiInquiryURL), new
        URL(uddiPublishURL));
 // we need to find the business in the UDDI
 // we must first create the Vector of business
   name
 Vector names = new Vector();
 names.add(new Name(businessName));
 // now get a list of all business matching our
   search criteria
 BusinessList businessList =
     proxy.find_business(names, null, null, null,
     null, null,10);
 // now we need to find the BusinessInfo object for
   our business
 Vector businessInfoVector =
businessList.getBusinessInfos().getBusinessInfoVector();
 BusinessInfo businessInfo = null;
 for (int i = 0; i < businessInfoVector.size(); i++) {
   businessInfo =          (BusinessInfo)
businessInfoVector.elementAt(i);
   // make sure we have the right one
  if(businessName.equals(businessInfo.getNameString())) {
    break;
   }
 }
```

We obtain a Definition object for the WSDL implementation from the implementation URL. We then obtained another Definition object, this one representing the WSDL interface, by searching all the imports defined in the implementation WSDL. A well-formed implementation WSDL should have an import pointing to its corresponding WSDL interface. It's simple now to find the target namespace which to be passed to Axis. We can make a call to Definition.getTargetNamespace() on the WSDL implementation object.

Advantages and Disadvantages of UDDI

The mechanism of service registry enables mutual Web Service integration, effectively improve the inter-operation ability of Web Service and promote wide application. Nevertheless, there are some drawbacks in present UDDI:

1) Service registry accepts service information passively.

When a service or the access point changes without updating service information in the registry, service requestor will probably use false information and fail to invoke Web Service.

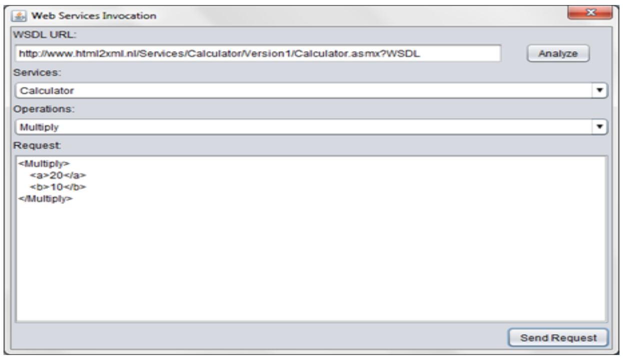
2) Present service registry contains too much complex classifications and information. It is not convenient for service requestor to quickly find and choose suitable Web Service and will influence the efficiency of program.

3) It cannot meet the requirement of dynamic invocation of Web Service. The Enterprise always uses the same fixed Web Services. When one service is down, service requestor can choose another service to guarantee uninterrupted service invocation. At present service requestor has to bind every Web Service till a suitable one is found. This process not only adds more works to service requestor, but also reduces the efficiency of program.

Therefore, to invoke Web Service in a dynamic and transparent way and to reduce the searching time for suitable service in the registry, efficient mechanism is required. We can get all the ports declared in the interface WSDL and just choose the first one. After choosing the port to use, we need to find the service name and port name which will be supplied to Axis. This can be done by finding the binding in the WSDL interface which corresponds to the chosen port. Thus we get the service and port in the WSDL implementation that provides an endpoint for this binding. We need to make sure that the binding we choose in the interface WSDL contains the same port we have already chosen. Then we need to make sure that the service chosen in the implementation WSDL contains a port which has a binding attribute value which is the same as the interface WSDL binding's name attribute value. If these are the same, then we have found the service and binding which refer to chosen port. Web Service has gradually become a new web application form and widely accepted by corporations and research institutions. Based on current UDDI and SOA, by adding notification mechanism, private service registry and UDDI Proxy, this paper designs and realizes a dynamic Web Service framework, which enhances the reliability, transparency and dynamics of Web Service invocation.

Conclusions

In order to take full advantage of the flexibility and power of Web services, the user must be able to dynamically discover and invoke a Web service implementation. This is the ultimate promise of Web services and the original reason why technologies like UDDI were developed. It has even been proposed that businesses could provide publicly-accessible implementations of Web services. Although, this idea is not yet ready, there is still use for such dynamic invocation. In this paper, we demonstrate how a Web services client can dynamically discover and invoke a Web service without any prior knowledge of its design. We used dynamic Web service invocation method to address maintainability and reliability issues without losing the overall system performance. To achieve these goals, we propose a Dynamic Web Service Invocation Framework (DWSIF). The dynamic invocation of Web services allows both service providers and service consumers to remain autonomous and maintain the loosely coupled relationship without scarifying the performance. Through a series of experiments and objective evaluations, we have shown that the dynamic web service invocation can serve its client better than static invocation, particularly in maintainability and reliability. Some of the screen shots of Dynamic Web Service Invocation Framework (DWSIF) are shown in Fig. (5) and Fig. (6).



WSDL URL: http://www.html2xml.nl/Services/Calculator/Version1/Calculator.asmx?WSDL

Services: Calculator

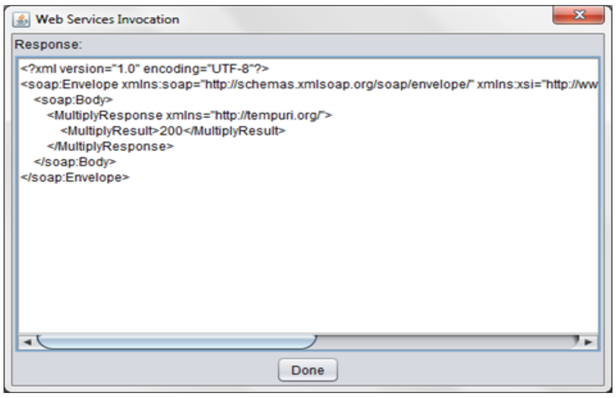
Operations: Multiply

Request:

```
<Multiply>
  <a>20</a>
  <b>10</b>
</Multiply>
```

Fig. 5- Web service invocation application tool

Result of multiplication operation is shown in following Fig. (6)



Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://ww
  <soap:Body>
    <MultiplyResponse xmlns="http://tempuri.org/">
      <MultiplyResult>200</MultiplyResult>
    </MultiplyResponse>
  </soap:Body>
</soap:Envelope>
```

Fig. 6- Result of Calculator Web service

Acknowledgement
We wish to thank teachers of Department of Computer Science, Shivaji University, Kolhapur for motivating us for this research.

References

[1] Haller A., Cimpian E., Mocan A., Oren E. and Bussler (2005) *IEEE International Conference on Web Services*.
[2] Medjahed B. and Bouguettaya A. (2005) *IEEE Transactions on Knowledge and Data Engineering*, 17(7).
[3] Greenwood D., Buhler P. and Reitbauer A. (2005) *IEEE International Conference on e-Technology, e-Commerce and e-Service* (EEE).
[4] Shen D., Yu G., Yin N. and Nie T. (2004) *IEEE International Conference on E-Commerce Technology for Dynamic EBusiness*.
[5] Doug Tidwell (2001) *IBM developer Works.*
[6] Hansen M., Madnick S. and Siegel M. (2002) *WES*, *LNCS* 2512, 12-27.
[7] Mrissa M., Benslimane D., Ghedira C. and Maamar Z. (2004) *IDEAS Workshop on Medical Information Systems: The Digital Hospital*.
[8] Pires P., Benevides M. and Mattoso M. (2003) *Symposium on Applications and the Internet* (SAINT).
[9] Shah D., Patel D. (2008) *SEEC*, 172-175.