Using Web Services to Control Remote Instruments for Online Experiment Systems

Yuhong Yan[1], Yong Liang[2], Xinge Du[2], Hamadou Saliah[3], Ali Ghorbani[2]

[1] NRC-IIT, Fredericton, NB, Canada,
Yuhong.yan@nrc.gc.ca
[2] Faculty of Computer Science, UNB, Canada
{Yong.liang, Xinge.Du}@unb.ca
[3] Télé-université, Montreal, Canada
Saliah@teluq.uquebec.ca

Abstract. Online experimentation allows students from anywhere to operate remote instruments at any time. Web service, as the latest technology for distributed applications, provides a new potential to build Online Experiment Systems (OES). The most valuable feature of Web service for OES is interoperability across platforms and programming languages. In this article, we propose a service-oriented architecture for OES enabled by Web service protocols. We present the methodology to wrap the operations of instruments into Web services. As the classic Web service is stateless, we discuss how to manage the instrument states in this application. Web service has intrinsic weaknesses on latency because it uses more transport layers for communication. Therefore we need to justify if the performance of Web services is feasible for online experiments.

1 The Service-Oriented Architecture for Online Experiment System

An Online Experiment System uses the scattered computational resources and instruments on the networks for experiments. The current online experiment systems commonly use classic client-sever architecture [1][2][3] [4] and off-the-shelf middleware for communication [4]. Normally, an online system relies on products from individual companies, such as National Instruments or Agilent. Windows™ is the common operating system for these instruments. The client side needs to install proper software to operate the remote instruments. The goals of resource sharing among the online laboratories and easy access via the web remain unachieved.

A Web service is a software system identified by a URI, whose public interfaces and binding are defined and described using XML (specifically Web Service Definition Language (WSDL) [5]). Its definition can be discovered by other software systems (e.g. via a registry server using Universal Description, Discovery and Integration (UDDI) protocol [6]). These systems may then interact with Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols (i.e. Simple Object Access Protocol (SOAP) [7]). Web service is designed to

support interoperable machine-to-machine interaction over a network. A service-oriented architecture based Web service is suitable to integrate heterogeneous resources for online experiment system.

We present a double client-server architecture for online experiment system (figure 1). The first client-server architecture is between the client browser and the web server associated with the online lab management system. The second client-server architecture is between the online lab management system and the scattered resources that are wrapped as Web services. SOAP message is used for communication between the online laboratory and the remote resources. The online lab management system is the key component in this architecture. It has functions like a normal Learning Management System, such as tutorial management, student management etc. It also can utilize the remote Web services. The system works in a series of steps. A service provider first registers its services in a UDDI registry server (step 1 in Figure 1). A service requester searches the registry server and gets all the potential resources. It selects the proper services based on its own criteria (step 2). The service requester sends SOAP messages directly to the service provider to invoke the remote service (step 3).

In this paper, we discuss how to interoperate the heterogeneous experiment resources using Web service, i.e. mainly the step 3 in figure 1. Process integration, i.e. how to discover the relevant resources and determine the operation process in a flexible way for an experiment, is not covered in this paper.

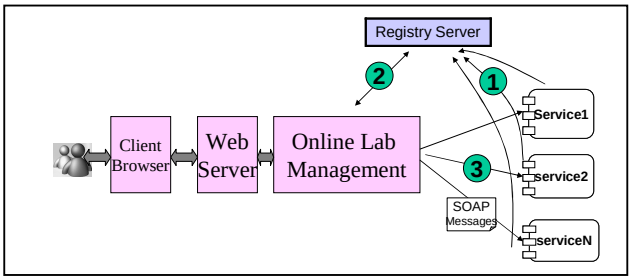

Fig. 1. Double Client-server Architecture for an Online Experiment System

2 Wrapping Instrument Functions as Web Services

A WSDL file contains the operations of the Web service and the arguments to invoke operations. Our WSDL file provides three kinds of information: 1) the input/output parameters to operate the instrument; 2) the information about rendering the GUI of the instrument panels; and, 3) the metadata about the instruments.

2.1 Instrument I/O Features

Instrument I/O is a well studied domain for which industrial standards have been established. Using the I/O library, we can control the instrument by sending an ASCII string to it and reading ASCII strings back from it. The commonly accepted industrial standards are Virtual Instrument Software Architecture (VISA) and Interchangeable Virtual Instruments (IVI) [9]. Most commercial products follow these standards. The purpose of these standards is to enable interoperability of instruments, which means using common APIs of the instruments.

IVI is a standard built on VISA. Compared with VISA, IVI can operate the instrument by referencing its properties. The IVI standard classifies the instruments into eight classes. Each class has basic properties that are shared by all the other instruments in the same class, and extension properties that are unique to the individual instrument. The following code sets the frequency of an Agilent Waveform Generator 33220A to 2500.0HZ by using VISA and IVI. IVI COM operates the property of frequency directly, while VISA COM sends a string whose semantics defines the operation.

```
//using IVI

Fgen->Output->Frequency = 2500.0;

 //using VISA

Fgen->WriteString("FREQuency 2500");
```

IVI add more measurement functions than VISA, such as state caching (i.e. keep track of current instrument settings to avoid sending redundant commands to the instrument[1]), simulation, multithread safety, and range checking. But on the other side, IVI driver has longer learning curve and slower execution speed. IVI drivers run slower because they do not invoke the instruments directly. IVI drivers consist of class driver and instrument-specific driver. They both do not provide an appropriate route for interchanging two instruments from different classes that are capable of making the same measurement.

2.2 Wrapping Instrument Operations based on VISA and IVI standards

Since both VISA and IVI send ASCII strings to control the instruments, the methodology of wrapping the instrument services can be generic to any instrument. This means the same Web services interface. We need only to define an operation ***writeString*** for sending commands or data to the instrument. The argument of this operation is always string, which is the same for any instrument. Similarly, we define an operation ***readString*** for getting status or data from the instrument. Table 1 is the

[1] For Web service, stateful service means to distinguish different client and invocation. It is different from the state here.

snippet of WSDL for defining the operation of writeString. See the WSDL snippet in Table 1.

Table 1. The Snippet of WSDL to Operate an Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ……>
  ……
  <!--define the response message -->
  ……
 <!--define the request message -->
  <wsdl:message name="writeStringRequest">
    <wsdl:part                    name="in0"
type="xsd:string"/>
  </wsdl:message>
  <!--define the operation -->
 <wsdl:operation   name="writeString"   param-
eterOrder="in0">
    <wsdl:input                          mes-
sage="intf:writeStringRequest"
name="writeStringRequest"/>
      ……
    </wsdl:operation>
```

For example, we want to operate the waveform generator to generate a sinusoid waveform. The set of control parameters for the sinusoid waveform contains "*instrument address*", "*wave shape*", "*impedance*", "*frequency*", "*amplitude*", and "*offset*". In order to save the time of composing SOAP message and establishing network connection, we combine multiple commands into one string, so that only one SOAP message is sent. After the server gets the string from the client, it will parse the string according to the delimiter (here we use ";") and send the command to the instrument. The following string combines multiple commands for the waveform generator.

```
"*RST;FUNCtion SINusoid;OUTPut:LOAD 50;FREQuency
2500;VOLTage 1.2;VOLTage:OFFSet 0.4;OUTPut ON";
```

Using VISA, the commands are parsed into the following code.

```
Fgen->WriteString("*RST");

Fgen->WriteString("FUNCtion SINusoid");

… …

Fgen->WriteString("OUTPut ON");
```

Using IVI, the commands are as following (the combined string is a little different from the one in VISA).

```
Fgen->Utility->Reset(); // Reset

Fgen->Output->Function =
Agilent33220OutputFunctionSinusoid;
```

```
Fgen->Output->Frequency = 2500.0;
Fgen->Output->State = VARIANT_TRUE; //on
....
```

We can also define one generic WSDL for each IVI instrument class, in which the operations for each property are define. The interoperability is satisfied if the instruments are in the same class and if they have the same extension properties.

3 Design the Web GUI for the Instrument

We need to display the panel of a remote instrument graphically on a web browser, so the user can operates the GUI to control the instruments. The principle is as following. The instrument panel is serialized as an XML file [4] and stored at the end of the remote instrument service. When the service is chosen, this file will be downloaded from the service to the online lab management system. The Web server of the online lab management system can parse it and render it to the client.

Figure 2 shows the process in detail. The XML schema for the Digital Multimeter is in DMM_GUI.xml. It is the knowledge of the online lab management system. The online lab management system uses it to validate the file DMM_Agilent_34401A_GUI.xml which defines the GUI for the Agilent 34401A (downloaded from the remote service). Then JAXB (an API and tools that automate the mapping between XML documents and Java objects) is used to parse DMM_Agilent_34401A_GUI.xml and map it to java servlet objects. The Web service associated with the online lab management system displays the panel objects on a HTML page. The right bottom section of Figure 2 shows the generated GUI page.

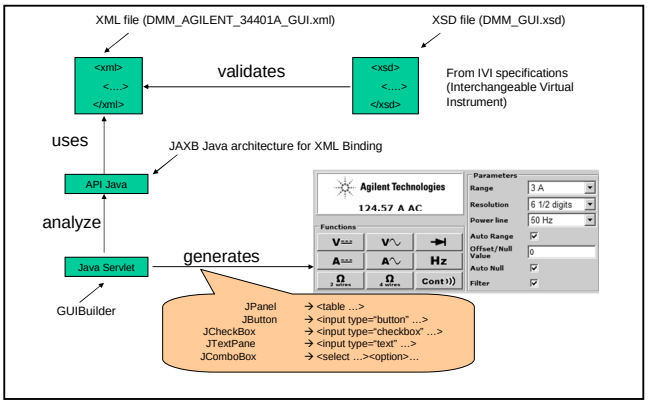

Fig. 2. The principle to display instrument panel from its XML description

It is a little more complex to show arbitrary shapes, such as waveforms. We have three options, generating a jpg image for the waveform or using applets (for java) or activeX control (for windows platform).

Table 2 shows a snippet of the XML for Agilent 34401A [10]. One can see the container panel objects are the ***parentFrame***, ***parentPanel*** and ***childPanel***. A container object can contain other panel objects, such as labels and text boxes. A container object has a layout that describes how to render the objects inside the container.

Table 2. A snippet of the XML to describe the panel of Agilent 34401A

```
<parentFrame parentFrameName="Frame Container">
  <parentFrameLayout> … </parentFrameLayout>
</parentFrame>
<parentPanel parentPanelName="Parent Panel">
   <parentPanelLayout>GridBagLayout</parentPanelLayout>     <par-
entPanelDimension>…</parentPanelDimension>
</parentPanel>
<childPanel childPanelName =     "ExternalParametersChildPanel">
  <childPanelLayout> … </childPanelLayout>
 <component className="jLabel">
      <componentName> … </componentName>
      …
  </component>
  …
</childPanel>
```

4 Interfaces of Meta Information

The IEEE Learning Object Metadata (LOM) standard defines metadata for a learning object [11]. LOM is designed for the objects of an online course. It includes information such as the author, the organization, and the language. Instruments can be taken as a kind of Learning Object. In [3], the LOM standard is extended for experimentation context. For operating an instrument, we add two additional types of information, the ***availability*** and the ***quality of services*** *(QoS)*.

Table 3. The operations to get metadata information in WSDL

```
<!--define the operation -->
<wsdl:operation name="getLOMMateData">
  <wsdl:output name="getLOMDataResponse">
  </wsdl:output>
<!--define the operation -->
<wsdl:operation name="getAvailabilityInfo">
  <wsdl:output
name="getAvailabilityResponse">
  </wsdl:output>
<!--define the operation -->
<wsdl:operation name="getQoSInfo">
  <wsdl:output name="getQoSResponse">
  </wsdl:output>
```

In the WSDL, we define the operation, ***getLOMMetaData***, to download the information and ***getAvailabilityInfo***, to get availability information for booking the service (Table 3).

QoS information is accumulated from history and can become an important selling and differentiating point of Web services with similar functionality. We record the successful connecting rate to the instrument, the response time to the instrument, and customer's rating to use its service. QoS information can be used for selecting proper instruments for an experiment (not covered in this paper). We design the operation ***getQoSInfo*** for this in WSDL (table 3).

5 Managing Stateful Instrument Web Services

It is well known that classic Web service is stateless, i.e. it does not maintain states between different clients or different invocations. HTTP, the commonly used transport protocol for Web services, is a stateless data-forwarding mechanism. There are no guarantees of packets being delivered to the destination and no guarantee of the order of the arriving packets. Classic Web services are suitable for services providing non-dynamic information. In order to manage the instrument Web services, we need additional effort.

An instrument itself does not record client information or invocations. Indeed, an instrument acts in a reactive way. It receives commands, executes them accordingly, and returns the results. If we say an instrument has "states", these are the parameters of its working mode, which have nothing to do with the states of a web service.

An instrument service needs to be stateful for two reasons. First we need to record the operations from one user for payment accounting and controlling how the user can use this instrument; and second we need to transport the results among several resources asynchronously.

Stateful services always rely on database or other persistency mechanism to maintain the states and recover from failures. But there are different schemas for defining the context of the states and how to pass the context between requests. Grid Services, such as GT4.0 from Globus alliance (www.globus.org), uses the pattern of "factory" to generate an instance of the service for each client. The service instance manages the stateful service for the client. This mechanism works well for a resource that can accept multiple users, e.g. a computer that can run multiple processes. Since the measurement instruments are single user resources, this factory mechanism does not work well in this application. Web Service Resource Framework (WSRF) is another proposed framework which relies on the resource itself to manage the states. WSRF passes the WS-addressing to point to the stateful resource. And this WS-addressing is past as the context of request between the client and server. Since our instruments are stateless resources, this framework does not work in this application.

We design the stateful service for instrument resources as in Figure 4. The state context is identified by the client ID and the resource identifier (an URI). In detail:

- Step 1. The client sends the request to the web service. The request should contain the ID of the client to identify the session.

- Step 2. The web service returns the identifier of the reference.
- Step 3. The client always contacts the service using the resource identifier.
- Step 4, 5. The online experiment is executed and the results are returned to the Web service.
- Step 6. The Web service records the results in a proper manner and returns the results to the client.

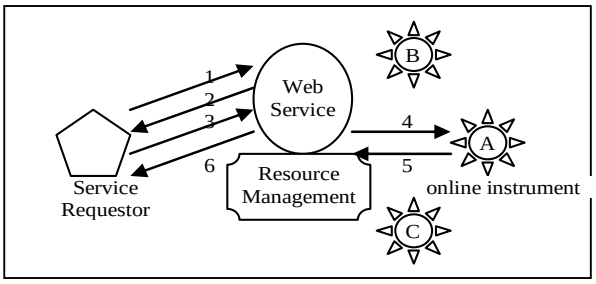

Fig. 3. The Stateful Service for Instrument Resources

6 Benchmark of Latency and Optimize the SOAP Efficiency

The trade-off of high interoperability of Web service is lower performance than other middleware due to more transport layers used for SOAP messages. The delay involves marshalling the SOAP message, binding it to the HTTP protocol at the request side, the transportation time over network and decoding time on the service side. Our benchmark test is aimed at determining the time to transport a service request from the requester to the provider. This test takes place when the instrument web service and the OES are on the same host. Thus, the Internet delay is not considered. We use ASCII strings for encoding a volume of the floating numbers in SOAP message. We assume each floating number has 16 digits to provide adequate precision. The size of the strings for floating numbers is directly proportional to the number of digits. We measured the time delay starting from the call of the service and ending as the request reaches the service endpoint. The dark blue line in Figure 4 shows the relation of the delay time vs. the number of data points per message.

The most straightforward method of optimization is to reduce the SOAP message size. By sending data as a SOAP attachment, we can reduce the message size and also save the time for XML encoding. The overhead of this method is the time for processing the attachment. The first test is using Multipurpose Internet Mail Extensions (MIME) attachment (purple line in Figure 4). We can see that when the volume of data is small, it is faster to transport XML message than to use attachment. It is because the encoding time is little for small volume of data, while the attachment processing costs more. But when there is large volume of data, the attachment way is faster, because the time for attachment processing does not increase much as the vol-

ume of data increases, while the encoding time increases proportionally to the volume of data. Transportation time of the SOAP message can be reduced further by compressing the payload. The second test is compressing the data into ZIP format and sending it as MIME attachment (yellow line in Figure 4). The payload size is 40~50% of its original size. We can see that the ratio of the yellow line is about half of the purple one. The intersection to the vertical axis is not changed. It means that the time is saved at transportation time, while other costs (e.g. preparing attachment, establishing connection) are unchanged.

Direct Internet Message Encapsulation (DIME), another specification for SOAP attachment format, is especially designed to address the basic features required for applications handling SOAP messages with attachments in a minimal and efficient manner by providing chunking and a special designed record header. Thus, DIME is simpler, and provides more efficient message encapsulation than MIME, while MIME provides the most flexibility. Sky-blue line in Figure 4 is for the test using DIME with ZIP method. It has the same ratio as the yellow line due to the size of payload is transported. The basic offset is lower due to that DIME is more efficient of processing attachment.

One can see that the transportation time can be reduced dramatically after the optimization. And the optimized delay falls into the feasible range for the context of this application when large amount data needs to be transported. We should point out that our application is in the e-learning domain and the tasks are not mission-crucial.

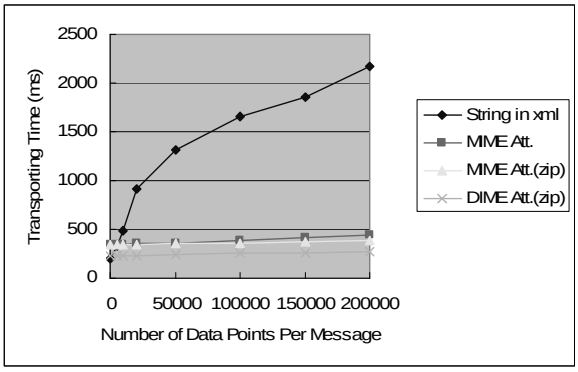

Fig. 4. Different Methods to send String Data through SOAP

7 Acknowledgement

The authors thank Hamadou Saliah-Hassane from Télé-université, Montreal, Canada for his discussions and support for this work.

8 Summary

We present our solutions to several technical problems to integrate heterogeneous experiment resources using Web service. It focuses more on data integration, rather than process integration. For the next step, we will study how to describe the resources in UDDI semantically and match the proper services for specific experiment requirements. SOAP performance is also an important topic for Web service too. As we write this paper, more optimization methods are presented.

References

1. Latchman, H. A., Ch. Salzmann, Denis Gillet, and Hicham Bouzekri, "Information Technology Enhanced Learning in Distance and Conventional Education", *IEEE Transactions on Education*, Vol. 42, No. 4, Nov. 1999, p247-254.
2. Auer, M.E.; Gallent, W. (200) , "The 'Remote Electronic Lab' as a Part of the Telelearning Concept at the Carinthia Tech Institute", *Proceedings of Interact Computer Aid Learning (ICL)*, Villach, Austria, Sept 28-29, 2000.
3. Bagnasco, A., M. Chirico, A. M. Scapolla, (2002) XML Technologies to Design Didactical Distributed Measurement Laboratories, IEEE IMTC2002, Anchorage, Alaska, USA.
4. Fattouh, B. and H. H. Saliah, (2004), Model for a Distributed Telelaboratory Interface Generator, Proceedings of Int. Conf. On Engineering Education and Research, Czech Republic, June 27-30, 2004.
5. W3C, (2004b), WSDL Specification, *http://www.w3.org/TR/wsdl*
6. UDDI.org, (2004), UDDI homepage, *http://uddi.org/pubs/uddi_v3.htm*
7. W3C,(2004a), SOAP Specification, *http://www.w3.org/TR/soap12-part1/*
8. Hardison, J. D. Zych, J.A. del Alamo, V.J. Harward, *et al.*, The Microelectronics WebLab 6.0 – An Implementation Using Web Services and the iLab Shared Architecture, *iCEER2005*, March, Tainan, Taiwan.
9. Agilent Inc. About Instrument I/O http://adn.tm.agilent.com/index.cgi?CONTENT_ID=239, 2005.
10. Yan, Y., Y. Liang, X. Du, H. Saliah-Hassane, A. Ghorbani, "Design Instrumental Web Services for Online Experiment Systems", *Ed-Media 2005*, Montreal, June 27-July 2, 2005, Montreal, Canada.
11. IEEE Learning Technology Standards Committee, (1999), IEEE 1484 Learning Objects Metadata (IEEE LOM), *http://www.ischool.washington.edu/sasutton/IEEE1484.html*.