Web Service Enabled Online Laboratory

Yuhong Yan, Yong Liang*, Abhijeet Roy**, Xinge Du**
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
*Institute of Information Technology, National Research Council, Canada
**Faculty of Computer Science, University of New Brunswick, Canada
yuhong@encs.concordia.ca

ABSTRACT:

Online experimentation allows students from anywhere to operate remote instruments at any time. The current techniques constrain users to bind to products from one company and install client side software. We use Web services and Service Oriented Architecture to improve the interoperability and usability of the remote instruments. Under a service oriented architecture for online experiment system, a generic methodology to wrap commercial instruments using IVI and VISA standard as Web services is developed. We enhance the instrument Web services into stateful services so that they can manage user booking and persist experiment results. We also benchmark the performance of this system when SOAP is used as the wire format for communication and propose solutions to optimize performance. In order to avoid any installation at the client side, we develop Web 2.0 based techniques to display the virtual instrument panel and real time signals with just a standard Web browser. The technique developed in this paper can be widely used for different real laboratories, such as microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing.

KEY WORDS:
Web services, online experiment, online laboratory, remote control, e-learning, e-science

INTRODUCTION

In science and engineering education, experimentation plays a crucial role. The classic university science course entails lecture and lab: students' active participation in experiments enhances their understanding of the principles described in the lectures. However, not every educational institution can afford all the experimental equipment it would like. Moreover, colleges and universities increasingly offer distance-learning programs, allowing students to attend lectures and seminars and complete coursework using the Internet. In situations such as these, access to online laboratories or experiment systems can greatly enhance student learning - increasing the range of experiments available at an institution and giving the distance learners hands-on, real-time experience. Online laboratories, however, are not as mature as online courses. There is no matured software system to support online experimentation. Experimentation is also an important approach for scientific discovery. Sharing expensive equipment is a common practice in the scientific community. Some research facilities, e.g. synchrotrons and accelerators, are very expensive that a country normally invests to build one of the kind. These facilities are shared by the scientific community national wide and/or international wide. Currently, the scientists need to reserve a time slot in these facilitates and travel to the site to conduct the experiments. With the capacity of online experimentation, traveling cost can be saved. More importantly, online experimentation can allow the users to reserve shorter time slots, because the users do not need to finish an experiment during their travel. Therefore, the resource sharing can be more efficient.

Current online experiment systems fall into two categories (Naef, 2006): *virtual laboratories* provide a simulation environment in which students conduct experiments; and *remote laboratories*, with real instruments and equipments at the remote sites. The later is the scope of our research. The ultimate goal of our research is to provide IT techniques for remote experimentation over Internet. Our focus in this paper is to let students use a Graphic User Interface (GUI) to operate actual instruments via remote control.

The difficulty with creating an effective laboratory operated by remote control is making scattered computational resources and instruments operable across platforms. Existing online experiment systems commonly use a classic client-sever architecture and off-the-shelf middleware for communication (Hardison, *et al.* 2005, Auer and Gallent, 2000, Latchman, *et al.* 1999). Normally, to ensure interoperability, these systems rely on instruments from a single company—such as National Instruments or Agilent—and Microsoft Windows as the common operating system. Users must then install additional software to operate the remote instruments. For a student using an old laptop or the computer at a public library, this could be difficult. So, online labs configured this way can't achieve the ultimate goals of sharing heterogeneous resources among online laboratories and easy access via the Web. Our solution to these shortcomings is to base online experiment systems on Web services, which are designed to support interoperable, machine to-machine interaction over a network and can also integrate heterogeneous resources. We have devised a service-oriented architecture for online experiment systems, enabled by Web service protocols, and a methodology for wrapping the operations of the instruments into Web services. Although these methods probably aren't suitable for time-critical missions or applications that need real-time control, such as robot operation, they do work for controlling standard commercial instruments over low-speed or unreliable communication networks—the types of networks available to many college students. Using this framework, we can create an online experiment system for students—or an online research lab for scientists—that incorporates a great variety of instruments and that users can access without installing special software.

This paper is organized as follows: following the present of service oriented architecture for online experiment systems, we present the solution to wrap instruments as Web services and to display dynamic graphics for real time signals. Then we discuss the management of stateful instrumental Web services and benchmark the performance of Web services in this application and present the optimization methods to improve performance. At the last is the discussion and conclusions.

SERVICE ORIENTED ARCHITECTURE FOR ONLINE EXPERIMENT SYSTEMS

A Web service is a software system that typically relies on a set of W3C standards. Identified by a Uniform Resource Identifier (URI), a Web service has public interfaces and bindings defined and described in Extensible Markup Language (XML), specifically, the WSDL format (W3C, 2004a). Other software systems can discover the Web service definition—for example, via a registry server using Universal Description Discovery and Integration (UDDI) protocol (UDDI.org, 2004). These other systems can then interact with a Web service as its definition prescribes, using Simple Object Access Protocol (SOAP) (W3C, 2004b), an XML-based messages format, conveyed by Internet protocols such as Hypertext Transfer Protocol (HTTP).

An Online Experiment System (OES) uses the scattered computational resources and instruments on the networks for experiments. It is a Web-enabled distributed system: the user accesses an

OES via the Web interface; and the heterogeneous resources and devices interoperate with each other using Web service protocols. Our major efforts in this study are the service oriented architecture for OES and the techniques to operate remote instruments wrapped as Web services.

Figure 1 diagrams our service oriented architecture for an OES. It combines two client-server architectures. The first client-server architecture mediates between the client's browser and the Web server associated with the online lab management system. The second client-server architecture mediates between the online lab management system and scattered resources wrapped as Web services. The online laboratory uses SOAP messages to communicate with the remote resources. The online lab management system is the key component in this architecture. It has standard learning management functions such as tutorial management, student management, and so on. The system uses service oriented architecture to invoke remote services. It works in a series of steps, indicated by the numbered green circles in Figure 1:

1. A service provider registers its services in a UDDI registry server.
2. A service requester searches the registry server and finds all the potential resources. It selects the proper services based on its own criteria.
3. The service requester sends SOAP messages directly to the service provider to invoke the remote service.

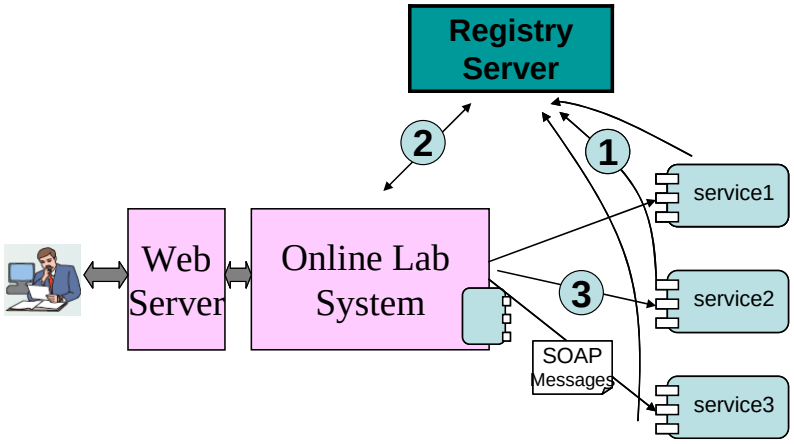

Figure 1. Service Oriented architecture for an Online Experiment System

Figure 2 shows our implementation of the online laboratory management system. We use Moodle, an open source learning management system, to realize the learning management functions. These modules are an authoring tool to generate tutorials, quizzes, and homework; a student management system to manage the enrollment, marks and progresses of students; a booking and scheduling module to book timeslots for the experiments; and multiple collaboration tools, such as discussion forums, wikis, and chat rooms for students to communicate and work in group. The tutorial pages contain Instrument GUI for displaying instrument panels and real time signals. The students can press the buttons and enter inputs from the Web interface to operate the remote instruments. The GUI code is designed to be very light weight, and no particular installation is needed at the client side. Our major focus of this paper is on wrapping instruments as Web services and designing the light weight GUI interface for operating the remote instrument Web

services.

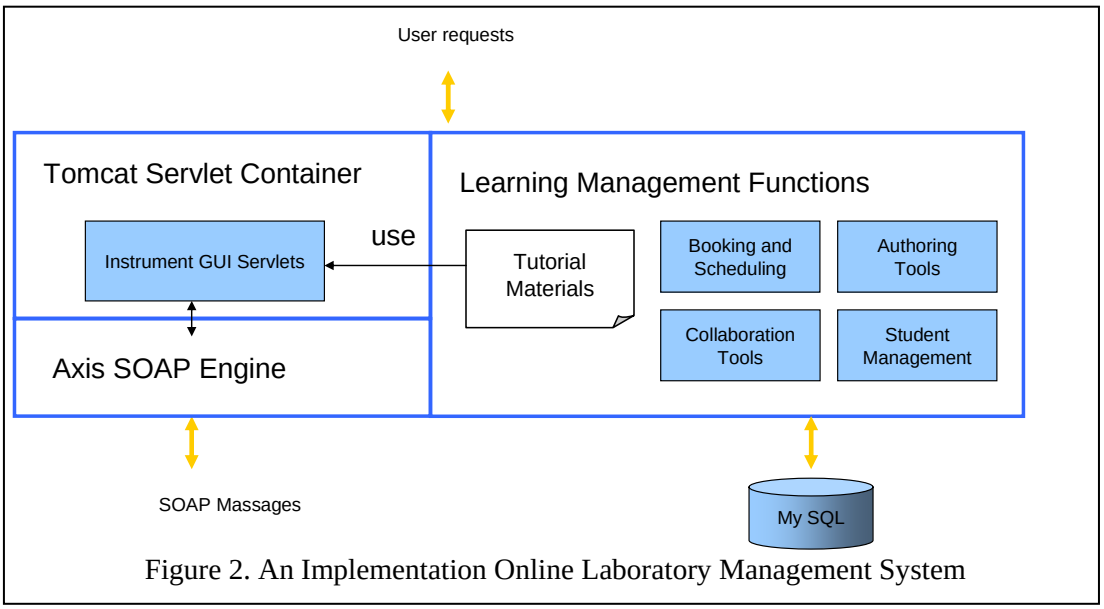

Figure 2. An Implementation Online Laboratory Management System

A Sample Experiment

Our research targets the online education at college level. We are especially interested in electronic circuit experiments, because the instruments in this domain commonly have digitalized interface to computer. A sample experiment is common emitter NPN transistor amplifier. The circuit diagram is as in Figure 3. The common emitter circuit comprises the load resistor R_C and NPN transistor with the output connected as shown. The resistors R_1 and R_2 are chosen to ensure the base-emitter voltage is approximately 0.7 volts, which is the "on" voltage for a transistor. These resistors, along with R_E, also determine the quiescent current flowing through the transistor and therefore its gain. The input signal V_{in} is generated by a waveform generator. The amplified signal is the output V_{out}.

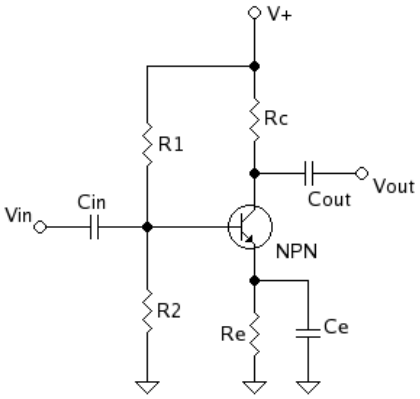

Figure 3. Common emitter amplifier

We use a waveform generator, Agilent 33220A, to generate the input signal V_{in}. And we use a data acquisition and switch unit, Agilent 34970A, to read in V_{in} and V_{out}. Please notice that in a real lab, people normally use an oscilloscope to observe the signals, but for online lab, everything has to be digitalized. Therefore, we need to use data acquisition devices. That means that we need to change the usual way to do experiment for putting an experiment online. In order to operate the instruments via GUI interface and via Web service protocols, we develop some techniques below to wrap the instruments into Web services and to display the instrument panels and the real time signals on Web pages.

WRAPPING INSTRUMENT OPERATIONS AS WEB SERVICES

A WSDL file contains the operations of the Web service and the arguments to invoke operations. When instrument functions are wrapped as Web services, the interface of the instrument Web service is described in a WSDL file. An instrument service needs to provide 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. These issues are described individually below.

Generic Approach to Wrap Instrument Operations based on VISA standard

Instrument I/O is a well studied topic for which industrial standards have been established. Two methods to control instruments are by using an instrument driver or by making direct calls to the I/O library. If using an instrument driver, the user will call functions that cause the instrument to take some action. If using the I/O library, the user will control the instrument by sending an ASCII string to it and reading ASCII strings back from it. The commonly used languages to operate instruments are C, C# or Visual Basic. The commonly accepted industrial standards are Virtual Instrument System Architecture (VISA) and Interchangeable Virtual Instruments (IVI) (Aglient, 2005). 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. Therefore, it is possible to generate generic WSDL interfaces for instruments based on these standards. The relationship between VISA and IVI is shown in Figure 4. The individual instruments – Instr. A, B and C – have their own drivers. These drivers are wrapped by VISA complaint drivers. The IVI compliant drivers are built still on the top of VISA standard.

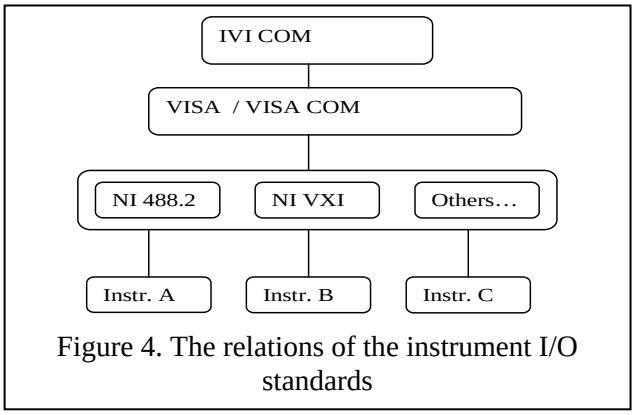

Figure 4. The relations of the instrument I/O standards

Both VISA and IVI standards operate the instruments by reading and sending ASCII strings to the instruments. 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 instruments in the same class, and extension properties that are unique to the individual instrument. As an example, Table 1 shows the code to set the frequency of an Agilent Waveform Generator 33220A to 2500.0HZ, using IVI COM. Table 2 is the code of VISA COM to implement the same function. Using VISA COM, people do not know the semantics of the parameters. That is to say, setting the Frequency or Voltage, people will use the same API.

```
IAgilent33220Ptr Fgen;
…….
Fgen->Output->Frequency = 2500.0;
…….
```

Table 1. Sample code of IVI COM

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

Table 2. Sample code of VISA COM

We consider that using the VISA standard, the methodology of wrapping the instrument services can be generic to any of the instruments, which means that many instruments can share the same Web services interface. Indeed, using the VISA standard, 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. Table 3 is the snippet of WSDL for defining the operation of *writeString*. Similarly, we can define an operation *readString* for getting status or data from the instrument, which is eliminated from Table 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ……>
      ……
  <!--define the response message -->
<wsdl:message name="writeStringResponse">
   <wsdl:part name="writeStringReturn" type="xsd:int"/>
 </wsdl: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" parameterOrder="in0">
    <wsdl:input message="intf:writeStringRequest" name="writeStringRequest"/>
    <wsdl:output message="intf:writeStringResponse" name="writeStringResponse"/>
        </wsdl:operation>
      ……
</wsdl:definitions>
```

Table 3. The snippet of WSDL to operate an instrument

In the example in Table 4, we demonstrate how 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 improve performance by reducing the time taken to send SOAP messages (ref Section 5), those parameters are put into one string. This means that only one SOAP message is transported to pass all the parameters from the client to the server. 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.

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

Table 4. Sinusoid waveform parameters in one string

Although we prefer to use the VISA standard to wrap the instrument functions, it is also possible to use the IVI standard. The difference is that each instrument class will have a common WSDL file in which the operations for the basic properties of this instrument class are defined. For

instruments having extension properties, the WSDL has to be generated separately to include the operations for the extension properties. Therefore, if using the IVI standard, the interoperability is satisfied if the instruments are in the same class and if they have the same extension properties.

Interfaces of Metadata.

The IEEE Learning Object Metadata (LOM) standard defines metadata for a learning object (LTSC, 1999). Any entities which can be used, reused or referenced during technology supported learning are learning objects. Examples of learning objects include multimedia content, instructional content, learning objectives, instructional software and software tools, and persons, organizations, or events referenced during technology supported learning. The LOM standard focuses on the minimal set of attributes needed to allow these learning objects to be managed, located, and evaluated.

An online course is a most common learning object. Relevant attributes of LOM for online courses are author, owner, terms of distribution, language, teaching or interaction style, grade level, and prerequisites etc. A part of an online experiment is similar to an online course as it also has tutorial materials and pedagogical attributes. (Bagnasco, Chirico, and Scapolla, 2002) extended the LOM standard for experimentation context to include instrument attributes and assignments. In their work, instruments are a part of an entire experiment and the information about them are not sufficient for searching and booking. Our work studies how to share instruments and operate instruments in more detail. Therefore, we need to extend their LOM extension for instrument attributes. Figure 5 shows the extended LOM attributes for instruments.

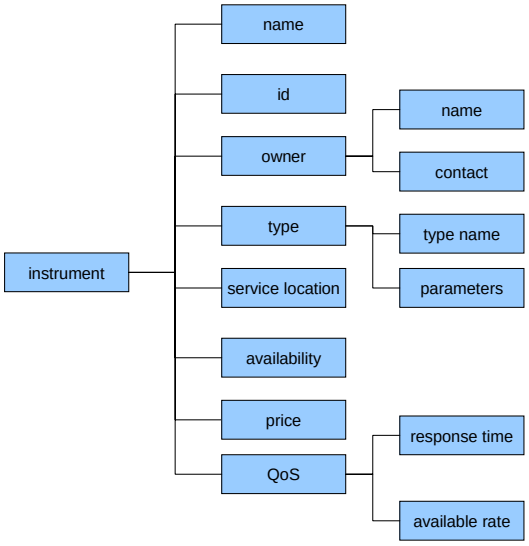

Figure 5. LOM Attributes for Instruments

The LOM metadata information is defined in an XML file. In the WSDL, we define the operation, *getLOMMetaData*, to download the information (c.f. Table 6). The LOM information can be used to search, evaluate, and utilize the proper instrument for an experiment. The information about availability and Quality of Service (QoS) are especially useful for evaluating and booking the instrument service. Therefore, we generate two operations, *getAvailabilityInfo* and *getQoSInfo*, for the two attributes. From *getAvailabilityInfo*, we can get all the available timeslots during a time interval, or query if the instrument is available for a specific time period.

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

Table 6. The operations to get metadata information in WSDL

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 and the response time to the instrument. QoS information is used when selecting available instruments for an experiment. The higher QoS of the instrument service, the more likely the OES selects this instrument or recommends it to the user to use. The operation *getQoSInfo*, is designed for this.

LIGHTWEIGHT WEB GUI FOR INSTRUMENTS

The primary advantage of a Web application over a desktop application is universal access. The client side of a Web application, in the best case, does not require local installations other than a standard Web browser. Therefore, Web applications are highly portable and platform independent. The Web interface to remotely operate an instrument needs to be user friendly and interactive, and gives the user a similar look and feel as the real instruments. For efficient reason, we should also reduce the amount of data transferred from the server to the client. We have solved the following two problems using only JavaScript enabled Web browser at the client side: to display the instrument panel graphical; to display dynamic graphics for real time signals.

The Web GUI for Virtual Instrument Panels

The panel of a remote instrument should be displayed graphically on a Web browser. The user operates the GUI to control the instruments. The methodology to describe instrument panels is presented in (Fattouh and Saliah, 2004). The principle is to design an XML schema which defines the syntax of the panel of a kind of instruments. An XML file compliant to the schema describes the panel of an individual instrument. Then the XML file can be parsed and rendered at the client side. We use the multimeter Agilent 34401A as an example. A snippet of the XML for its panel is in Table 5.

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

Table 5. A snippet of the XML to describe the panel of Agilent 34401A

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. If one is familiar with java, one can see the objects can be mapped one by one to the classes in a java swing GUI package.

Figure 5 shows the principle to display the panel from its XML description. The XML schema for the Digital Multimeter is in DMM_GUI.xml. It validates the file DMM_Agilent_34401A_GUI.xml which defines the GUI for the Aglient 34401A. The JAXB is used to parse DMM_Agilent_34401A_GUI.xml. Then a java servlet is used to display the panel object on an HTML page. The generated GUI page is displayed on the right bottom section of Figure 5.

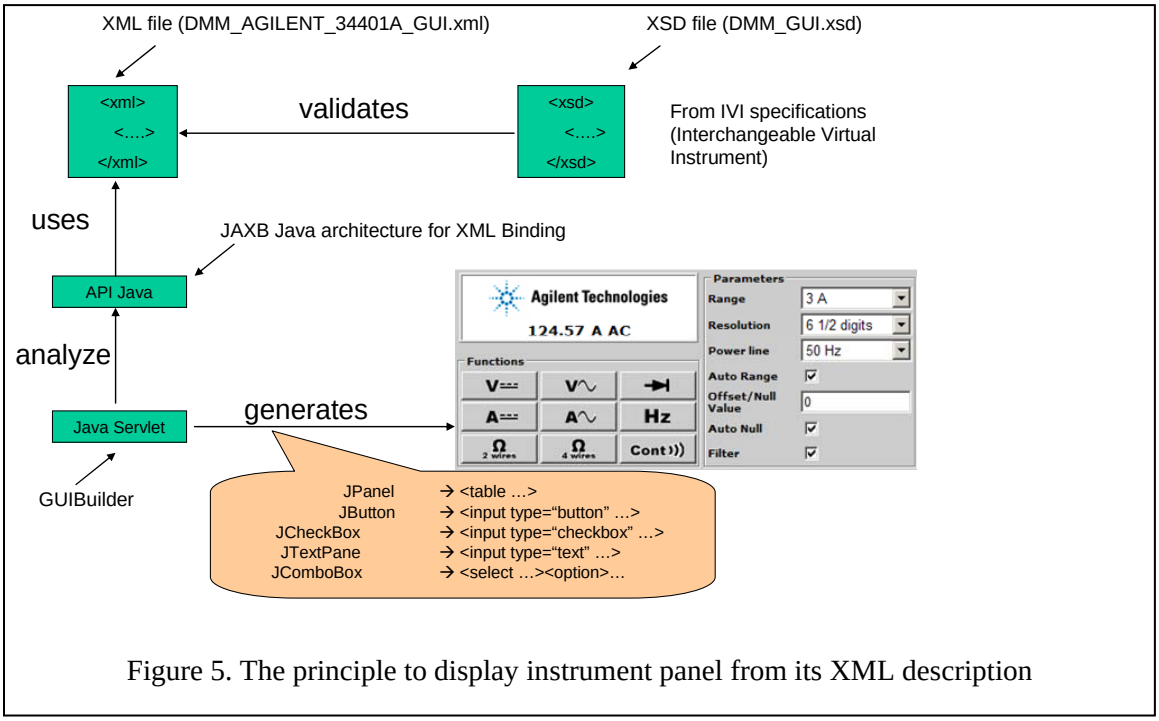

Figure 5. The principle to display instrument panel from its XML description

Display Real Time Signal on Web GUI

A large number of instruments display a coordinate diagram (graphs and charts) in a panel. A vital feature of the instrument Web interface is to display experiment results that closely resemble the output of the instrument itself. Displaying dynamic textual results in a browser is straightforward. However, displaying dynamic pictorial results is challenging due to the limited graphical capabilities of prevalent Web browsers.

AJAX, shorthand for "Asynchronous JavaScript and XML", is a development technique for creating interactive Web applications (Garrett, 2005). The main technologies used in an AJAX enabled web application are asynchronous data retrieval using XMLHttpRequest and dynamic manipulation and display of html elements based on the retrieved data using Document Object Model (DOM). The intent is to make Web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire Web page does not have to be reloaded each time the user requests a change. AJAX is a good solution for displaying and updating the real time signals. We can transfer just the discrete data points for the signals and update just part of the screen for displaying the data points. Then whole screen looks still and the signal is updating. Bandwidth requirement is reduced making Web pages load faster. This approach also allows keeping data independent of the presentation layer.

JavaScript Object Notation (JSON) is used for transferring the data points between the end user and the servlet. JSON is a text based data interchange format (JSON.org, 2007). What makes JSON rather useful is its inherent support within JavaScript. JavaScript, being the most widely supported scripting language for Web browsers, is very efficient in parsing JSON messages. JSON is not going to replace XML anytime soon, but it provides a viable alternative.

Parsing XML messages within a browser results in a Document Object Model (DOM) tree, which makes the code to manipulate it complicated. On the other hand, parsing JSON results in JavaScript objects and the code to manipulate it is straight forward. Table 7 shows a JSON text and Table 8 shows the equivalent JavaScript objects that can be obtained from parsing the above JSON text. In JSON, an object can be represented by a name value pair separated by a colon and surrounded by curly brackets. An array can be represented within square brackets, and values can be separated by a comma. The value of an object can be a string, an integer, another object, or arrays. In addition, the Boolean literals "true" and "false" are supported. The void concept of "null" is also supported. JavaScript "eval" function, which invokes the JavaScript compiler, can be used for parsing JSON text.

```
{
  "name"    : "Jack Sullivan",
  "student"   : true,
  "subjects" : ["Web Programming" , "Discrete Math" ,
                "Psychology" , "Operating Systems"]
}
```

Table 7. Code snippet: A JSON document

```
var name = "Jack Sullivan";
var status = true;
```

```
var subjects = new Array("Web Programming","Discrete Math",
                         "Psychology", "Operating Systems");
```

Table 8. Code snippet: JSON string to Java Script Object

JSON can represent semi-structured data very efficiently compared to XML. Table 6 provides a comparison between XML and JSON representation that shows the similarity and differences between these two formats. XML is relatively verbose and would almost always require more characters to represent the same data. JSON has been touted as the "fat free alternative to xml".

XML … 146 characters without blanks	JSON … 103 characters without blanks
<student fulltime="false"> <name>Wallace</name> <subject>Rabbit psyche</subject> <subject>Carrot care</subject> <subject>Cage building</subject> </student>	{ "student" : { "fulltime" : false , "name" : "Wallace" , "subjects" : ["Rabbit psyche" , "Carrot care" , "Cage building"] } }

Table 6. Code snippet: Comparing XML and JSON

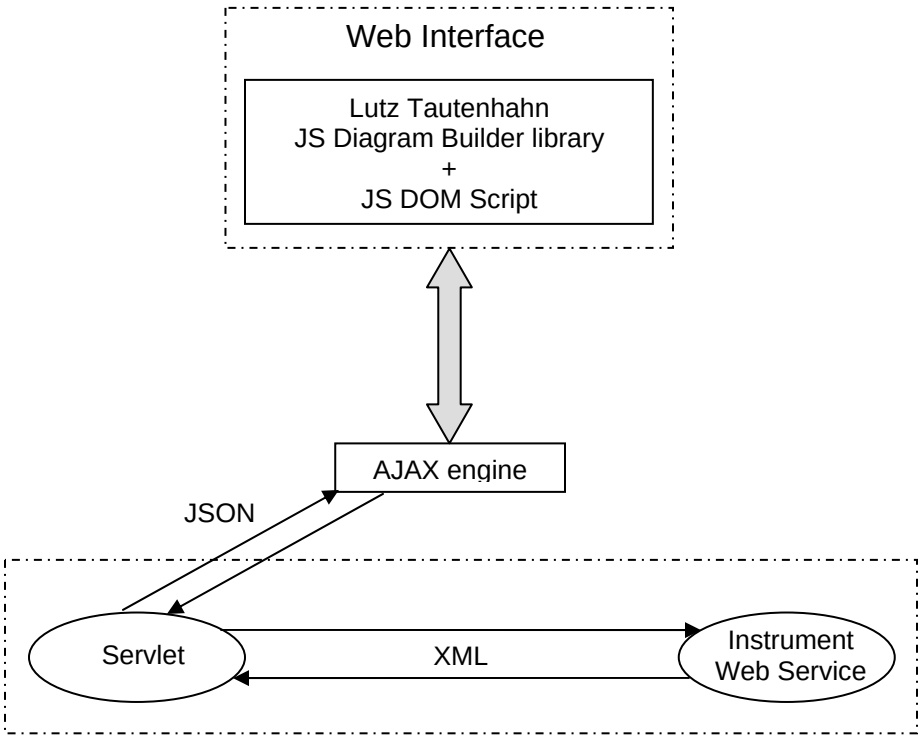

Figure 6 Web page dynamic graphics technique

Figure 6 depicts the technique used for creating dynamic coordinate diagrams for the Web interface. As stated earlier, a browser is not an ideal platform to construct dynamic images. HTML 4.01 specifications recommended by the W3C does not even state the basic unit of graphics – the pixel as one of the HTML elements. Lutz Tautenhahn, a German software developer, has created a JavaScript library that can be used to display coordinate diagrams (Tautenhahn, 2005). The JavaScript Diagram Builder library is available as a freeware. In our solution we utilise this library to draw the coordinate space and to translate coordinates to positions within a Web page.

The div html element of 1px width and 2px height is used to mark a coordinate in the graph panel. The divs are placed within the Web page according to the translated coordinate. The AJAX engine as seen in Figure 6 is JavaScript method that creates XMLHttpRequest object binds it with a timer and fetches the data from the Servlet. AJAX technique is used to fetch the coordinates from the application server in JSON format and is dynamically displayed over the output panel. In order to mimic the constant output of a wave generator we continuously call the server periodically for new data. The final interface looks like in Figure 7. The demo can be found at our testing Web site http://flydragontech.com/prototypes/lms/oisee/OISEE.htm.

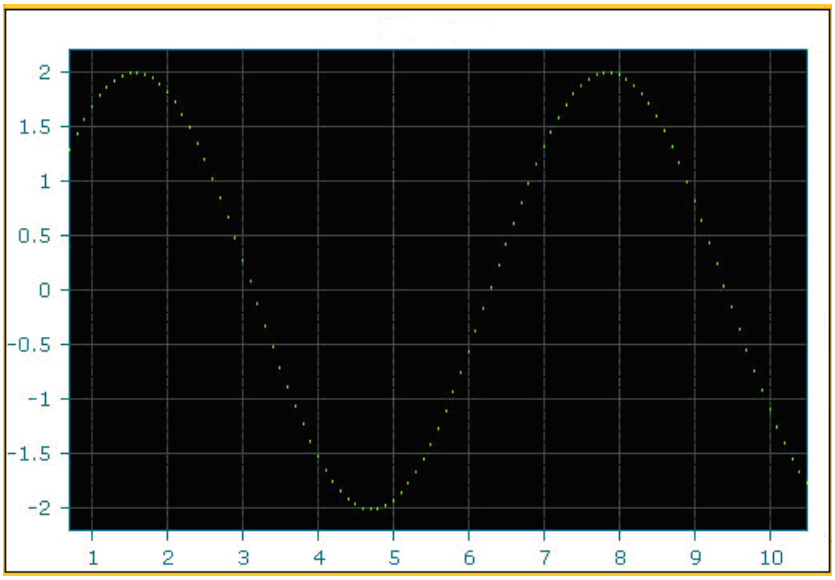

Figure 7 Display real time signal on Web GUI with AJAX and JSON

MANAGING STATEFUL INSTRUMENT WEB SERVICES

Instrument Web services involve remotely operating real devices in real time. Improper design of the Web services can cause damage to the instrument, and can lead to false measurement and control, which in turn will result in failure of the online experiment. In (Yan, *et al.*, 2005), we present the special requirements for the instrument Web services, such as reliability mechanisms and communication strategies. By using proper software technologies, these requirements can be satisfied. In the following sub-sections, we will focus on how to manage the instruments as resources.

Stateful Service for Stateless Resources.

It is well known that classic Web services 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 this subsection, we discuss if additional effort is needed to manage the instrument Web services.

An instrument itself is a stateless resource. This is because 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 can only be occupied by one user at a time. Unlike the resources in Grid Services, instruments can only accept one user at a time because an instrument needs to be set to a specific working mode before it can work for a certain experiment. Normally it is not possible to recover an instrument's status without a proper procedure, so many mechanisms in Grid Services are not useful in our application. The use of an instrument is booked by time slots. On some occasions, the tasks of an instrument can be managed by a queue (Hardison, *et al.*, 2005).

An instrument service needs to be stateful for several reasons. It must be stateful when it needs to record the operations from one user for payment accounting, or to support booking and scheduling, or to control how the user can use this instrument, or when the results need to be transported among several resources asynchronously. In the next subsection, we present a method to build the stateful instrument Web services.

Design the Stateful Service for Instrument Resources

As stated previously, we know that the instrument service has to identify clients and maintain a history of the operation. This kind of stateful service is different from the available stateful framework in Grid Services and WSRF. We design the stateful service for instrument resources as in Figure 8. The states are managed by the resource management layer. The client ID is transferred in SOAP to identify the states of the services. In detail:
(1) The client sends the request to the Web service. The request should contain the ID of the client to identify the session.
(2) The Web service returns the identifier of the reference.
(3) The client always contacts the service using the resource identifier.
(4) – (5) The online experiment is executed and the results are returned to the Web service.
(6) The Web service records the results in a proper manner and returns the results to the client.

Compared to Grid Services, our method does not use a service factory to create service instances for different users, because an instrument service is a single user service, thus no service instances are created. Compared to WSRF, the resource in our method itself remains stateless. We add a layer to manage the states.

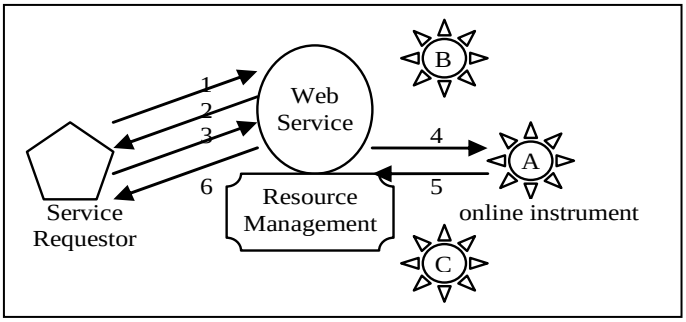

Figure 8. The stateful service for instrument resources

THE PERFORMANCE ISSUES FOR WEB SERVICES FOR ONLINE EXPERIMENT

Although Web services have strong advantages on interoperability, it has intrinsic weaknesses on latency and scalability because it uses more transport layers.

The trade-off of the high interoperability of Web services is its lower performance. Web services have intrinsic performance weaknesses for two main reasons: there are more transport layers than for middleware; and the overhead of using SOAP. Many researchers have analyzed the problem of SOAP efficiency and identified some factors that can affect the latency performance of Web services and SOAP (Chiu, Govindaraju and Bramley, 2002) (Litou, 2002) (van Engelen, 2003). For each factor that could cause the latency, there are some proposed methods to improve the performance. In this paper, we benchmark the SOAP efficiency in this context and propose the solutions to improve performance.

Benchmark of Latency

This benchmark test is aimed at determining the time to transport a service request from the requester to the provider. The time involves marshalling the SOAP message and binding it to the HTTP protocol at the request side, and the transportation time and decoding time on the service side. This test takes place when the instrument Web service and the OES are on the same host, thus, the delay by the Internet is not considered. In above, we described that instruments accept ASCII strings as input according to VISA and IVI standards. Therefore we use ASCII strings for encoding a volume of the floating numbers in SOAP message. In our test, we assumed each of the floating numbers had 16 digits to provide adequate precision. Therefore the size of the strings for floating numbers is directly proportional to the number of digits. We measured the time delay starting before the call of the service and ending as the request reaches the service endpoint.

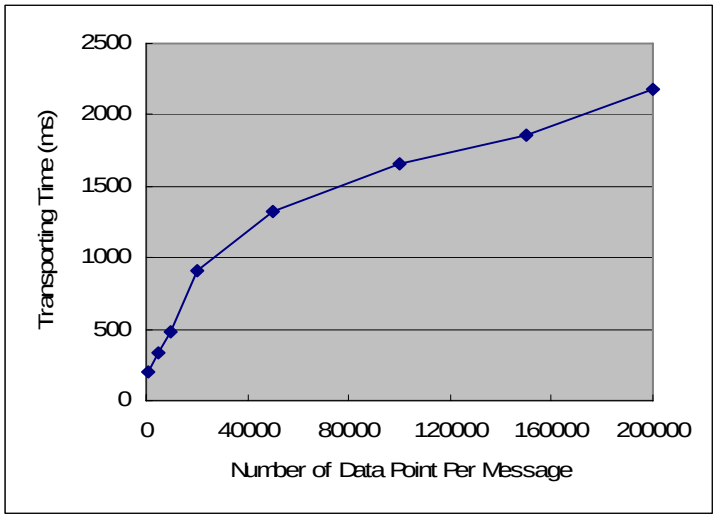

Figure 9. The delay vs. number of data point

Figure 9 shows the relation of the delay time vs. the number of data points per message. One can see that the delay increases quasi-linearly as the data points increase. There is also a basic overhead for the transportation, which is primarily the time for setting up the TCP/IP connection.

Optimize the SOAP Efficiency

Latency of SOAP message is caused by the time of transportation, which is proportional to the size of SOAP, and the delay caused by the TCP/IP layer.
The most straightforward method of optimization is to reduce the SOAP message size by extracting the string out of the XML, compressing it into binary format (we use ZIP compression format here) and sending it as an attachment. The size of the payload is reduced to approximately 40 to 50 per cent of its original size. The SOAP messaging protocol supports Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) attachments. The difference is that MIME is designed to provide flexibility, while DIME is designed to be simpler and to provide more efficient message encapsulation. The results of applying different attachment approaches are shown in Figure 10. One can see that the transportation time can be reduced dramatically by compressing the SOAP content.

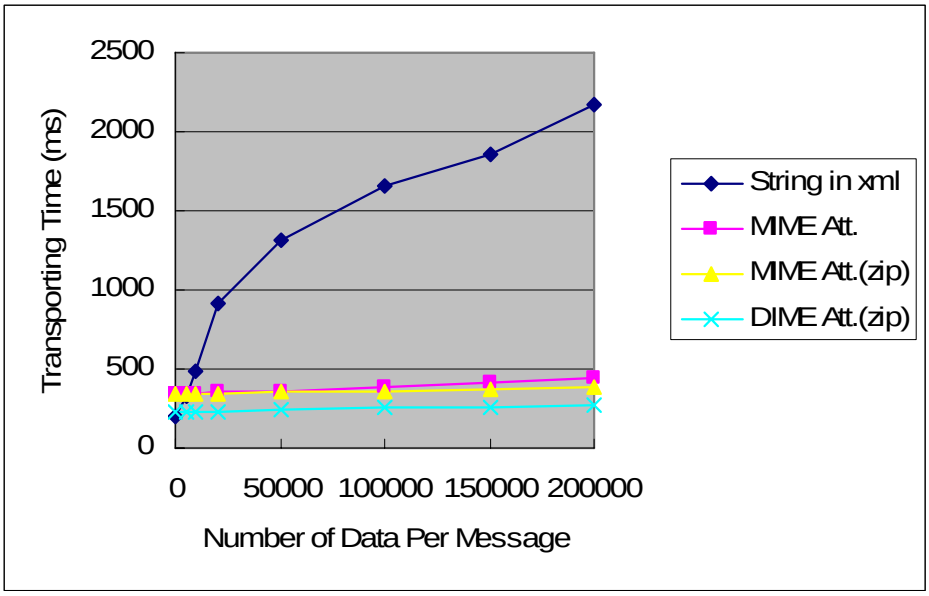

Figure 10. Different methods to send string data through SOAP

We can also optimize the underlying HTTP and TCP protocols for SOAP messaging. We present the possible methods below without testing results:

Persistent HTTP Connection. Persistent connection could "keep-alive" a connection and save the time needed to establish HTTP connection every time. For HTTP 1.0, the persistent connection works only if there is no proxy between the client and server. For HTTP1.1, the persistent connection can be used with more than one proxy between a client and a server.

Disable Nagle Algorithm and Remove TCP Delay ACK. The Nagle algorithm in combination with the TCP delayed ACK (the acknowledge response in TCP) are used to prevent network congestion (Litou, 2002), but they cause unnecessary delays when sending a SOAP message (Elfwing, R., Paulsson, U., and Lundberg, L., 2002). It is possible to disable Nagle on both the server and client side to get considerable improvement for the response time.

Better Pipelined Connection by Using HTTP 1.1. The use of HTTP inherits some of the TCP features such as the three-way handshake. This can cause delays. HTTP1.1 attempts to solve these problems. The result shows that HTTP1.1 can reduce the RTT (Round Trip Time) to half of HTTP1.0 implementation (Nielsen, *et al.*, 1997).

DISCUSSIONS

Service oriented architecture is the new technique to build distributed system. It has attracted attention for building online experiment systems as well. iLab at MIT adopts SOAP for communications between users and labs (Hardison, *et al.* 2005). Their architecture has three roles: lab server, end user and service broker. The service broker mediates the communication between the user and the lab server and provides storage and administrative services that are generic and can be shared by multiple labs within a signal university. Their focus is on providing such a kind of broker. In our architecture, the broker role is rather weak and not more than the function of registration. Our focus in this paper is to study how to control a remote device via Web service protocol and how to design lightweight Web GUI for remote instruments using Web 2.0 techniques. We consider that the communication between the end user and the online lab should be very light that SOAP is too heavy to use. So far as we know, our work is unique in this domain.

During our study, we find that the current devices and the way of experiments are not completely suitable for online experiments. For example, the users have to do everything by sending command, not using their hands. In our sample experiment, the users cannot assemble the circuits, nor tune the resistor to change the amplitude. Therefore, we need to design a special circuit to allow the users to configure the circuit by sending commands. Technically it is possible in our experiment. But in some other cases, to achieve reconfiguration is not so straightforward. Moreover, in real lab, the input and output signals are observed by oscilloscopes. In this paper, we digitalize the signals and display them on the Web site. We consider it is a cheaper way than using digital oscilloscopes. So people can see that we have different ways to do online experiments than real experiments. We have also found that not all the instruments can be shared online. For example, waveform generator produces analog signals which are not sharable without digitalization. Therefore, it is hard to say that it is useful to share a waveform generator online. Under current experiment devices and methods, it is commonly feasible to share a whole experiment or share a group of devices that has digital interface. We suppose more devices and experiment methods will be designed for online experiment and online engineering domain.

Collaborative working environment is helpful for online experiment. Currently, we just use the standard online tools. In the future work, we would use camera and video conferencing tools to enhance our environment. Other future tasks are to optimize the SOAP messaging and to analyse the resource description and integration issues.

CONCLUSIONS AND FUTHER WORK

Our vision is to share expensive equipment and educational materials associated with lab experiments as broadly as possible within higher education and beyond. In this paper, we propose to wrap the remote instruments as Web services for online experiment systems. The advantage of Web services is its inter-operability across platforms and programming languages. Its trade-off is low efficiency caused by SOAP messaging. This paper covers the essential issues to build instrument Web services, such as WSDL design, stateful service management and performance issues. This paper also presents the lightweight Web GUI to display virtual instrument panels and real time signals using Web 2.0 techniques. The technique developed in this paper can be widely used for different real laboratories, such as microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing.

As the paper is published, we have got a Canarie (http://www.canarie.ca/about/index.html) funded project called Science Studio for enabling remote experimentation with the synchrotron within Canada Light Source (CLS) (http://www.lightsource.ca/), a national science research laboratory. Currently the scientists who use the facilitate to do experiments need to travel to Winnipeg where the CLS is located and reserve the beamline time for 3 days as a minimum unit. With the capacity of remote experimentation, the scientists can save traveling time and cost. Furthermore, the occupy time of an experiment can be shrunk to hours instead of days. Therefore, the facilitate can be more efficiently shared in the scientific community. Our work within Science Studio is much beyond the topics touched in this paper.

REFERENCES

Agilent Inc., (2005), “About Instrument I/O”, http://adn.tm.agilent.com/index.cgi?CONTENT_ID=239, retrieved in 2005.

Auer, M.E., and Gallent, W., (2000), "The 'Remote Electronic Lab' as a Part of the Telelearning Concept at the Carinthia Tech Institute", Proc. Interactive Computer-Aided Learning (ICL), Kassel University Press.

Bagnasco, A., Chirico, M., and Scapolla, A. M., (2002) XML Technologies to Design Didactical Distributed Measurement Laboratories, IEEE Instrument and Measurement Technology Conference (IMTC), Anchorage, Alaska, USA.

Chiu, k., Govindaraju, M., and Bramley, R., (2002) "Investigating the Limits of SOAP Performance for Scientific Computing", 11th IEEE international Symposium on High Performance Distributed Computing HPDC-11, 2002.

Elfwing, R., Paulsson, U., and Lundberg, L., (2002), Performance of SOAP in Web service Environment Compared to CORBA, Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSE'02), 2002, IEEE.

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.

Garrett, J.J., (2005), "Ajax: A New Approach to Web Applications", available at http://adaptivepath.com/publications/essays/archives/000385.php, last retrieved March 29 2007.

Hardison, J. D. Zych, J.A. del Alamo, V.J. Harward, et al. (2005), "The Microelectronics WebLab 6.0 – An Implementation Using Web services and the iLab Shared Architecture", iCEER2005, March, Tainan, Taiwan.

Hardison, J., Hardison, D.J., Zych D., del Alamo, J. A. and *et al* (2005), "The Microelectronics WebLab 6.0—An Implementation Using Web services and the iLab Shared Architecture", Proc. Int'l Conf. Engineering Education and Research (iCEER2005), Int'l Network for Engineering Education and Research, 2005; http://wwwmtl.mit.edu/~alamo/pdf/2005/RC-107%20paper.pdf.

JSON.org, (2007) "Introducing JSON", available at: http://www.json.org/, last retrieved in March 29 2007.

Latchman, H.A., Salzmann, Ch., Gillet, D., Bouzekri, H. (1999), "Information Technology Enhanced Learning in Distance and Conventional Education", *IEEE Trans. Education*, Nov. 1999, p247-254.

Litou, M., (2002), Migrating to Web services – Latency and Scalability, Proceedings of Fourth Int. Workshop on Web Site Evolution (WSE), 2002, IEEE.

LTSC (IEEE Learning Technology Standards Committee) (1999), IEEE 1484 Learning Objects Metadata (IEEE LOM), http://www.ischool.washington.edu/sasutton/IEEE1484.html.

Naef, Olivier, (2006), Real laboratory, virtual laboratory or remote laboratory: what is the most efficient way?, *International Journal of Online Engineering*, v2(n3), http://www.i-joe.org/ojs/sitemap.php, retrieved in May, 2007.

Nielsen, H., J, Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, (1997), Network Performance Effects of HTTP/1.1, CSS1, and PNG, http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html, June 1997.

Salzmann, C., and D. Gillet, (2002), Real-time Interaction over the Internet, Proceedings of IFAC2002.

UDDI.org, (2004), UDDI homepage, *http://uddi.org/pubs/uddi_v3.htm.*

Tautenhahn, T., (2005), "JavaScript Diagram Builder 3.3", *http://www.lutanho.net/diagram/*, retrieved in March 2007.

van Engelen, R. (2003) Pushing the SOAP Envelop With Web services for Scientific Computing, in the proceedings of the International Conference on Web services (ICWS), 2003, pages 346-354.

W3C, (2004a), WSDL Specification, *http://www.w3.org/TR/wsdl.*

W3C, (2004b), SOAP Specification, *http://www.w3.org/TR/soap12-part1/.*

Yan, Y., Liang, Y., Du, X., Saliah-Hassane, H., and Ghorbani, A., (2005), "Design Instrumental Web services for Online Experiment Systems", Ed-Media 2005, Montreal, June 27-July 2, 2005, Montreal, Canada.

ABOUT THE AUTHOR

Yuhong Yan is an assistant professor in the Department of Computer Science and Software Engineering in Concordia University, Canada. She is also an adjunct professor at University of New Brunswick. Her research interests are Web service modeling and computing, monotonicity analysis in model abstraction and data mining.

Yong Liang is a research associate at Institute of Information Technology at National Research Council of Canada. His research interests are Web services; middleware and network communication.

Abhijeet Roy is an undergraduate student at University of New Brunswick. He was working as an intern at Institute of Information Technology at National Research Council of Canada when the work was done.

Xinge Du is a Master's student at University of New Brunswick. He was working as a visiting worker at Institute of Information Technology at National Research Council of Canada when the work was done.