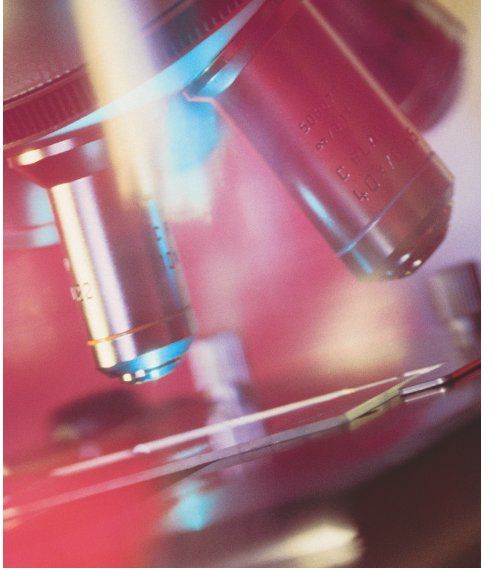


Web services provide a way to offer remote control of scattered scientific instruments, enabling online labs that students can use from anywhere, at any time.

Yuhong Yan, Yong Liang, Xinge Du, Hamadou Saliah-Hassane, and Ali Ghorbani

Putting Labs Online with Web Services

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. Current online experiment systems fall into two categories: *virtual laboratories* provide a simulation environment in which students conduct experiments; *remote laboratories*, our focus in this article, let students use a 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. (The "Online Experiment Systems: Other Architectures" sidebar lists sources.) 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 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

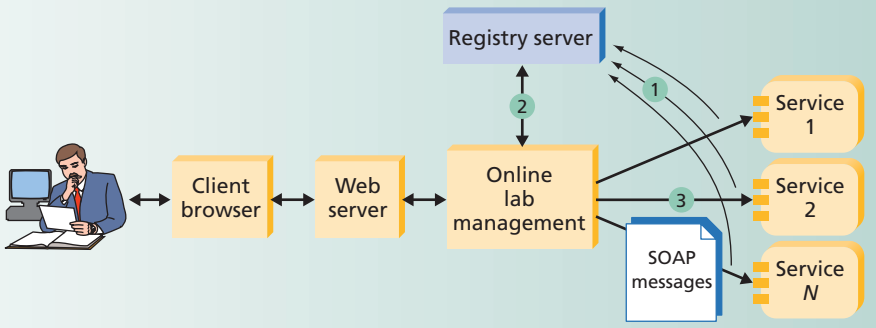
Inside

1520-9202/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

Online Experiment Systems: Other Architectures

- "Information Technology Enhanced Learning in Distance and Conventional Education," H.A. Latchman and colleagues, *IEEE Trans. Education*, Nov. 1999, pp. 247-254.
- "The Microelectronics WebLab 6.0—An Implementation Using Web Services and the iLab Shared Architecture," Hardison and colleagues, *Proc. Int'l Conf. Engineering Education and Research* (iCEER2005), Int'l Network for Engineering Education and Research, 2005; http://www-mtl.mit.edu/~alamo/pdf/2005/RC-107%20paper.pdf.
- "Model for a Distributed Telelaboratory Interface Generator," B. Fattouh and H.H. Saliah, *Proc. Int'l Conf. Engineering Education and Research*, Int'l Network for Engineering Education and Research, 2004.
- "The 'Remote Electronic Lab' as a Part of the Telelearning Concept at the Carinthia Tech Institute," M.E. Auer and W. Gallent, *Proc. Interactive Computer-Aided Learning* (ICL), Kassel University Press, 2000.
- "XML Technologies to Design Didactical Distributed Measurement Laboratories," A. Bagnasco, M. Chirico, and A.M. Scapolla, *Proc. IEEE Instrumentation and Measurement Tech. Conf.* (IMTC), IEEE, 2002, pp. 651-655.

Figure 1. Double client-server architecture for an online experiment system.



instruments and that users can access without installing special software.

SERVICE-ORIENTED ARCHITECTURE FOR AN ONLINE EXPERIMENT SYSTEM

A Web service is a software system that typically relies on any alphabet soup of standards, as listed in the "Web Service Glossary" sidebar. Identified by a URI, a Web service has public interfaces and bindings defined and described in XML, specifically, the WSDL format. Other software systems can discover the Web service definition—for example, via a registry server using UDDI. These other systems can then interact with a Web service as its definition prescribes, using XML-based messages conveyed by Internet protocols such as SOAP.

Figure 1 diagrams our double client-server architecture for an online experiment system. 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. It can also use remote Web services. The system 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.

Our focus in this article is how to integrate the heterogeneous experiment resources using Web services—that is, step 3 in Figure 1. We won't cover process integration—how to discover the relevant resources and flexibly determine the operation process for an experiment.

MIT's iLab is another online lab architecture using SOAP, as the "Online Experiment Systems: Other Architectures" sidebar describes. Its broker uses SOAP messages to access both online labs and users. Our online lab management system plays the broker role of interacting with resource services, but we don't use an addi-

tional SOAP client-server tier for the user. Our arrangement preserves the existing Web-based GUI that many online labs already have. Most importantly, this allows a lightweight client that doesn't need extra software to process SOAP.

WRAPPING INSTRUMENT FUNCTIONS AS WEB SERVICES

A WSDL file contains a Web service's operations and the arguments for invoking operations. Our WSDL file provides three types of information:

- input/output (I/O) parameters for operating the instrument,
- information about rendering a GUI of the instrument panels, and
- metadata about the instrument.

Instrument I/O features

Instrument I/O is a well-studied domain with established industrial standards. Most commercial products follow the Virtual Instrument System Architecture (VISA) and Interchangeable Virtual Instruments (IVI) standards, which enable instrument interoperability by using common APIs for the instruments (http://adn.tm.agilent.com/index.cgi?CONTENT_ID=239). Thus, using an I/O library based on these standards, we can control an instrument by sending an ASCII string to it and reading ASCII strings back from it.

IVI is an extension of VISA. Rather than sending the command as VISA does, IVI operates an instrument by referencing its properties. The following code uses VISA and IVI to set the frequency of an Agilent 33220A waveform generator to 2,500.0 Hz. IVI operates the frequency property directly, while VISA sends a string whose semantics define the operation.

```
//using IVI
Fgen->Output->Frequency = 2500.0;

//using VISA
Fgen->WriteString("FREQuency 2500");
```

IVI divides instruments into eight classes. Each class has basic properties that all instruments in the class share and extension properties unique to an individual instrument. IVI also includes more measurement functions than VISA, such as simulation, multithread safety, range checking, and state caching. (State caching means keeping track of current instrument settings to avoid sending redundant commands to the instrument. This sense of "state" differs from that in stateful Web services—discussed later in this article—which distinguish among different clients and invocations.) On the other hand, IVI drivers have a longer learning curve, and they execute more slowly because they don't invoke the instruments directly. An IVI driver consists of a class driver and an instrument-specific driver. But neither provides an appropriate route for interchanging two instruments from different classes that are capable of making the same measurement.

Wrapping instrument operations based on VISA and IVI standards

Because 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 all instruments can use the same Web services interface. We simply define a writeString operation 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 a readString operation for getting status or data from the instrument. Figure 2 is the snippet of WSDL for defining the writeString operation.

Let's say we want to operate a waveform generator to generate a sinusoid waveform. The set of control parameters for the sinusoid waveform contains instrument

Web Service Glossary

- **UDDI: Universal Description, Discovery, and Integration (http://uddi.org/pubs/uddi_v3.htm)**
- **URI: Uniform Resource Identifier**
- **SOAP: Simple Object Access Protocol (http://www.w3.org/TR/soap12-part1)**
- **WSDL: Web Services Description Language (http://www.w3.org/TR/wsdl)**
- **XML: Extensible Markup Language**

Figure 2. Snippet of WSDL for operating 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"
parameterOrder="in0">
<wsdl:input message="intf:writeStringRequest"
name="writeStringRequest"/>
.....
</wsdl:operation>
.....
</wsdl:definitions>
```

Figure 3. Principle for displaying an instrument panel from its XML description.

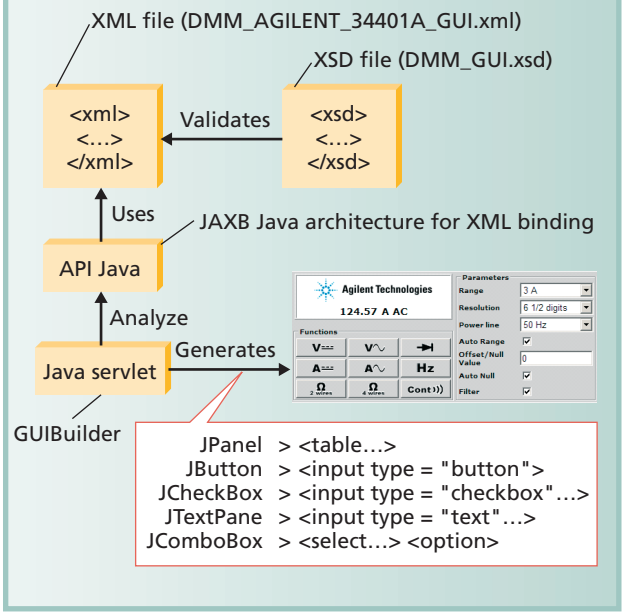


Figure 4. XML snippet describing the control panel of an Agilent 34401A, a digital multimeter.

```
<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>
```

address, wave shape, impedance, frequency, amplitude, and offset. To save the time of composing a separate SOAP message and establishing network connection for each of these, we combine multiple commands into one string and send only one SOAP message. After the server receives the string from the client, it parses the string according to the delimiter (here we use a semicolon) and sends 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 server parses the
commands into this code:

Fgen->WriteString("*RST");
Fgen->WriteString("FUNCtion SINusoid");
...
Fgen->WriteString("OUTPut ON");

With IVI, the combined string is a
little different, and it yields these
commands:

Fgen->Utility->Reset(); // Reset
Fgen->Output->Function =
Agilent33220OutputFunctionSinusoid;
Fgen->Output->Frequency = 2500.0;
Fgen->Output->State = VARIANT_TRUE;
//on
....
```

We can also define a generic WSDL snippet for each IVI instrument class, defining the operations for each property. This makes instruments in the same class, with the same extension properties, effectively interoperable.

DESIGNING INSTRUMENT WEB GUIS

Our next task is to display a remote instrument's control panel graphically on a Web browser, so that the user can operate the GUI to control the instrument. To do this, we serialize the instrument panel as an XML file and store it at the end of the remote instrument service ("Model for a Distributed Telelaboratory Interface Generator," B. Fattouh and H. Saliah-Hassane, *Proc. Int'l Conf. Engineering Education and Research*, Int'l Network for Engineering Education and Research, 2004). When a user chooses this service, the online lab management system downloads this file from the service. The system's Web server can then parse the file and render it to the client.

Figure 3 shows the process in detail. The XML schema for a digital multimeter is DMM_GUI.xsd, which is part of the online lab management system's knowledge. The online lab management system uses this schema to validate the file DMM_Agilent_34401A_GUI.xml, which defines the GUI for the Agilent 34401A multimeter, downloaded from the remote service. Then the online lab management system uses JAXB (an API and tools that automate mapping between XML documents and Java objects) 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 an HTML page. The box at the bottom right of Figure 3 shows the GUI page generated.

Figure 4 shows a snippet of the XML to create the GUI for the Agilent 34401A digital multimeter ("Design Instrumental Web Services for Online Experiment Systems," Yuhong Yan and colleagues, *Proc. World Conf. Educational Multimedia, Hypermedia, and Telecomm.* (ED-MEDIA), AACE, 2005). ParentFrame, parentPanel, and childPanel are container panel objects. A container object can contain other panel objects, such as labels and text boxes, and it includes a layout that describes how to render the objects it contains.

It's a little more complex to show arbitrary shapes such as waveforms. We have three options: generating a jpeg image for the waveform, using applets (for Java), and using activeX control (for Windows).

Anatomy of an Online Experiment

Online Common Emitter Amplifier Experiment

The common emitter amplifier is one of the three basic transistor amplifier configurations. In this experiment, the student investigates the principle of a basic NPN common emitter transistor amplifier. The remote lab prepares the amplifier circuit and connects it with a waveform generator for the input signal, a multichannel data-acquisition system for the observation points, and an electricity power source. The students can access tutorial materials and book time slots for this experiment via the online experiment system.

At experiment time, the student controls the waveform generator to give different input signals to the circuit and observe the measurement of the observation points. The experiment requires the student to observe how the amplifier works with DC and AC signals, measure the amplification magnitude, and measure the input and output impedance of the amplifier. The student can save data into files and use online tools to draw graphs.

Other tests can't be as easily implemented using existing equipment. For example, when students are present in the lab, they can insert a test resistor in series with the signal input to the amplifier and measure how much of the AC generator signal actually appears at the input of the amplifier. But since the student can't reconfigure the circuit online, this test is not yet possible as an online experiment. Many online experiments will require the design of special equipment.

INTERFACES OF META INFORMATION

The IEEE Learning Object Metadata (LOM) standard, designed for creating online classes, defines learning-object metadata such as author, organization, and language (IEEE 1484 Learning Objects Metadata, IEEE, 1999; http://www.ischool.washington.edu/sasutton/IEEE1484.html). In an online lab, instruments qualify as learning objects, and researchers have extended the LOM standard for the experimentation context ("XML Technologies to Design Didactical Distributed Measurement Laboratories," A.M. Bagnasco, M. Chirico, and A.M. Scapolla, *Proc. IEEE Instrumentation and Measurement Tech. Conf.* (IMTC), IEEE, 2002). To operate an instrument, we add two additional types of information: availability and quality of service.

In WSDL, we define the operation getLOMMetaData to download the information, and getAvailabilityInfo to obtain availability information for booking the service; Figure 5 shows the definition of these operations.

An instrument's QoS information, accumulated from a Web service's history, consists of the successful connection rate, response time, and customer's rating. Users examine all the information to select the most effective instruments for an experiment. QoS can be an important selling point, differentiating among services with similar functionality.

Figure 5. Operations for getting metadata information in WSDL.

```
<!—define the operation—>
<wsdl:operation name = "getLOMMetaData">
<wsdl: output name="getLOMDataResponse">
</wsdloutput>
<!—define the operation—>
<wsdl:operation name = "getAvailabilityInfo">
<wsdl: output name= "getAvailabilityResponse">
</wsdloutput>
<!—define the operation—>
<wsdl:operation name = "getQoSInfo">
<wsdl: output name= "getQoSResponse">
</wsdloutput>

</wsdl:operation>
```

MANAGING STATEFUL INSTRUMENT WEB SERVICES

The classic Web service is stateless, which means that it doesn't maintain states between different clients or different invocations. HTTP, the commonly used transport protocol for Web services, is a stateless data-forwarding mechanism. So a Web service guarantees neither packet delivery to the destination nor the order of the arriving packets. Thus, classic Web services are suitable for services

Figure 6. Stateful service for instrument resources.

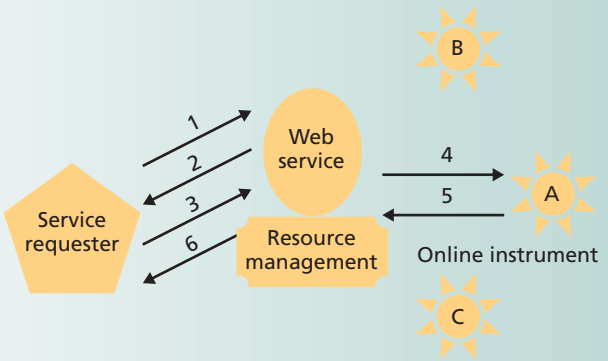
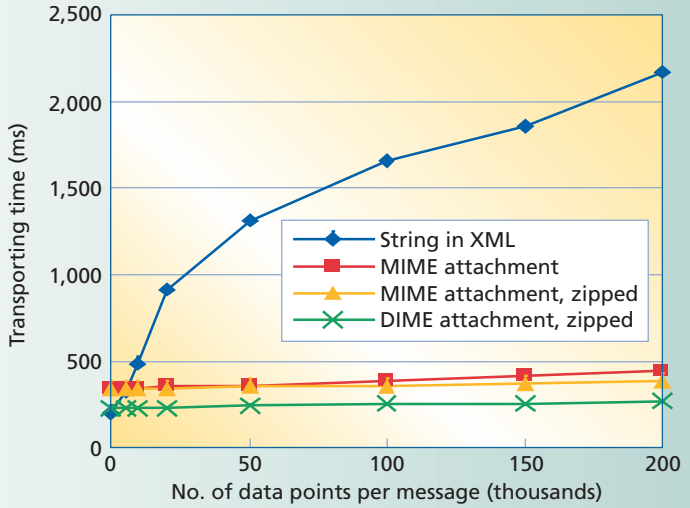


Figure 7. Different methods of sending string data through SOAP.



providing nondynamic information. Managing Web services for instruments takes additional effort.

An instrument itself does not record client information or invocations. Indeed, an instrument is reactive: 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 states in the context of Web services.)

An instrument service must be stateful for two reasons. First, we must have a way to record a user's operations for accounting and payment purposes and to control how the user can use this instrument. Second, we need to transport results among several resources asynchronously.

Stateful services always rely on a database or another persistency mechanism to maintain states and recover from failures. But different schemas are available for defining the context of the states and how to pass the context between requests. Grid services, such as GT4.0 from the Globus alliance (www.globus.org), use the "factory" pattern to generate an instance of the service for each client, and each client's service instance manages stateful service. This mechanism works well for a resource that can accept multiple users—for example, a computer that runs multiple processes. But measurement instruments are single-user resources, so the factory mechanism won't work well in our application.

Another possibility, the Web Service Resource Framework (WSRF), relies on the resource itself to manage states. To point to a stateful resource, WSRF uses WS addressing, which it passes as the context of a request between the client and server. But our instruments are stateless, so this framework also won't work for our online lab.

Figure 6 diagrams the stateful service we've designed for instrument resources. The client ID and resource identifier (a URI) identify the state context. The process follows these steps:

1. The client sends the request to the Web service. The request contains the client's ID to identify the session.
2. The Web service returns the resource's identifier.
3. The client always contacts the service using the resource identifier.
4. The instrument conducts the online experiment.
5. The instrument returns the results to the Web service.
6. The Web service records the results appropriately and returns the results to the client.

BENCHMARKING LATENCY AND OPTIMIZING SOAP EFFICIENCY

The downside of Web services' high interoperability is slower performance than other middleware because of the additional transport layers for the SOAP messages. The delay involves marshalling the SOAP message, binding it to the HTTP protocol at the request side, transportation time over the network, and decoding time on the service side. To study and improve latency, we first designed a benchmark test to determine the time to transport a service request from the requester to the provider. We conducted this test with the instrument Web service and the online experiment system on the same host, so this test doesn't consider Internet delay.

We used ASCII strings to encode several floating-point values in a SOAP message. We assume each floating-point value has 16 digits to provide adequate precision. The size

of the strings is directly proportional to the number of digits. We measured the time starting from the call of the service and ending as the request reaches the service endpoint. The blue line in Figure 7 shows the relation of transportation time to the number of data points in the message.

The most straightforward optimization method is to reduce the SOAP message size. By sending data as a SOAP attachment, we reduce the message size and save time on XML encoding. The overhead of this method is time processing the attachment. Our first test used a Multipurpose Internet Mail Extensions (MIME) attachment. As we see in Figure 7, when the data volume is small, it is faster to transport the XML message than to use an attachment. This is because for small volumes of data, XML encoding takes less time than attachment processing. However, with a large volume of data, the attachment provides the faster method, because attachment processing time doesn't increase much as the volume of data increases, whereas XML encoding time increases in proportion to the volume of data.

We can further reduce SOAP transportation time by compressing the payload. For our second test, we compressed the data into the zip format, making the payload size 40 to 50 percent of its original size, and sent it as a MIME attachment (yellow line in Figure 7). The slope of the yellow line is about half that of the red one (uncompressed MIME attachment), but its height on the vertical axis is basically the same. This means that ZIP compression saved time at transportation time, but other costs (such as preparing the attachment and establishing the connection) remained the same.

Direct Internet Message Encapsulation (DIME), another SOAP attachment format, handles SOAP messages with attachments quickly and efficiently by using chunking and a specially designed record header. DIME is simpler and provides more efficient message encapsulation than MIME, although MIME provides greater flexibility. The green line in Figure 7 shows the results of our test using DIME attachments with zip compression. It has the same slope as the yellow line, reflecting the size of the payload transported. The basic offset is lower because DIME provides more efficient attachment processing.

Our tests show that we can reduce transportation time dramatically through optimization. The optimized delay falls into the feasible range for the context of this application when it must transport large amounts of data. Of course, we designed our online experiment system for the e-learning domain, so the tasks aren't mission-critical, and don't require lightning-fast responses.

Finding a Lab Online

- **eMerge, http://www.emerge-project.net. According the project proposal, eMerge "will involve the creation of innovative solutions concerning the networking of partner laboratories, server technologies, access control, security, queuing, scalable user interfaces, and pedagogical approaches."**
- **iLab, http://ilab.mit.edu/ServiceBroker6_0/home.aspx. MIT's remote laboratory gives access to equipment for experiments in microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing. From the iLab Web site: "The iLabs vision is to share expensive equipment and educational materials associated with lab experiments as broadly as possible within higher education and beyond."**
- **Prolearn, http://www.prolearn-project.org. Another European-based project, Prolearn includes contributions from numerous universities worldwide. Virtual and remote experiments are available in fields including probability and statistics, chemistry, earthquake engineering, software design, nanotechnology, and many others. For experiment descriptions see http://prolearn-oe.org/bin/view/OE/ExperimentDescriptions.**
- **Telelabs Project, http://telerobot.mech.uwa.edu.au/index.html. Run out of the University of Western Australia's School of Mechanical Engineering, Telelabs offers UWA undergraduates online engineering experiments; meanwhile, graduate students in the mechatronics engineering degree program "learn mechatronics integration and system development techniques by developing and extending the Telelabs system to equipment across the engineering faculty." The site's Links page also maintains a list of remote laboratories worldwide.**

By focusing more on data integration than process integration, our methodology solves several technical problems of integrating heterogeneous experiment resources, using Web services. The next step of this project will be to study how to describe the resources in UDDI semantically and how to match the proper services to specific experiment requirements. SOAP performance remains an important issue as Web services evolve, and, as this article goes to press, researchers are presenting additional methods for SOAP optimization. ■

Yuhong Yan *is a research officer in the Institute for Information Technology at Canada's National Research Council and an adjunct professor on the faculty of computer*

science at the University of New Brunswick. Contact her at yuhong.yan@nrc.gc.ca.

***Yong Liang** was a master's student at the University of New Brunswick. When this work was done, he was a visiting worker in the Institute for Information Technology at Canada's National Research Council. Contact him at yong.liang@unb.ca.*

***Xinge Du** is a master's student at the University of New Brunswick and was a visiting worker in the Institute for Information Technology at Canada's National Research Council when this work was done. Contact him at xinge.du@unb.ca.*

***Hamadou Saliah-Hassane** is a professor of informatics at Université du Québec à Montréal, Télé-université, department of Science and Technology. He is the research team leader on Laboratories at Distance for Education and Research (Lab@DER). Contact him at saliah@teluq.uqam.ca.*

***Ali Ghorbani** is a professor on the faculty of computer science at the University of New Brunswick. Contact him at ghorbani@unb.ca.*

For further information on this or any other computing topic, visit our Digital Library at http://computer.org/publications/dlib.