Exploring Innovation in Education and Research
Tainan, Taiwan, 1-5 March 2005

A General Framework for Web Services and Grid-Based Technologies for Online Laboratories

H. Saliah-Hassane[1], D. Benslimane[2], I. De La Teja[3], B. Fattouh[4], L.K. Do[5], G. Paquette[6], M. Saad[7], L. Villardier[8], Y. Yan[9]

saliah@teluq.uquebec.ca; djamal.benslimane@iuta.univ-lyon1.fr, idelatej@licef.teluq.uquebec.ca, lkimdo@teluq.uquebec.ca; bfattouh@licef.teluq.uquebec.ca; Maarouf.Saad@etsmtl.ca; Louis_Villardier@teluq.uquebec.ca, Yuhong.yan@nrc.gc.ca

Télé-université[1,3,5,8], Montreal, Canada
LICEF/CIRTA[1,4], Montreal, Canada,
École de Technologie de l'Information[4], Montreal, Canada,
École de Technologie de Supérieure[7], Montreal, Canada,
Université Claude Bernard Lyon[2], Lyon, France,
NRC-IIT[9], Fredericton, Canada

ABSTRACT: The combination of Web Services and grid-computing technologies is currently of a major scientific revolution. It combines the middleware solution from Web Services and resource-sharing solutions of grid computing. We present a general framework based on Web Services and grid-based technologies for online laboratories. It is a distributed system model where computational resources and experimental devices throughout the networks are organized into federations. The benefits of this model are information processing capacity increase and resource sharing. We discuss a number of technical considerations using this framework. These include: the descriptions of tele-experimentation resources; the wrapping of instruments into a web service; the composition of Web Services, which is modeled as a planning problem, and; the design of an online laboratory brokerage system, which we dealt with in a former article. We also discuss some issues related to business logic and policy in a particular sector, such as tele-learning and network-supported research via information technologies.

INTRODUCTION

Online Laboratory is a typical distributed application. The next generation of the online laboratory system will go beyond the current client/server architecture to grid-based architecture. The components for an online laboratory (instruments, testing devices, or one entire experiment) can be scattered over the Internet as individual web services. The online laboratory system is able to discover these web services and use them in one experiment. This is seamless to the users in that they would not notice that the instruments and devices are from different physical locations. The Internet is a bus connecting all these services. Service providers can charge for services provided based on individual user requests.

The enabling technologies for building such a distributed environment come from two domains: the Grid Technology and the Web Services [1][2]. Grid Technology focuses on using distributed heterogenous resources to solve massive computational problems. Grid Technology sees the world from the resources point of view. It has specifications for resource management, such as descriptions of resource properties and performance. Scheduling and clustering are studied to ensure the computational performance of the grid. Web services see the world from a service point of view. The services are provided by software components over the internet. The services are invoked by sending XML-based SOAP message to the remote components. Web services rely on internet protocols, such as HTTP, BEEP and XML technology to ensure the interoperability of the components on different platforms and are implemented in different programming languages. W3C accepts the following standards: Simple Object Access Protocol (SOAP), a message-based communication for component interaction [3]; Web Service Description Language (WSDL), component interface definition [4], and; Universal Description, Discovery Integration (UDDI), service discovery and integration [5]. Grid Services are the combination of Web Services and grid technology [6][7]. Grid Services still work on the computational problems using widely accepted Web Services protocols as the transport layer. The resources in Grid Services have a Web Services interface in WSDL. The coming Web Service Resource Framework (WSRF) will unify the Grid Services and Web Services. WSRF fills out the Web services stack to be consistent with the Open Grid Services Infrastructure (see next section). We expect that, in the future, the individual Web Services will have the properties of the current Grid Services.

For online experiments, we have computational tasks and data sharing tasks, as well as the tasks to use the services of instruments or devices. Thus we deal with high performance issues as well as service composition issues. In this paper, we will present a general framework of an online laboratory system based on the current specifications of web services and grid services. We analyse the special requirements of individual web services for this application and present our solutions to meet these requirements. We also present a solution for Web Services composition based on Artificial Intelligence (AI) planning technology.

This paper is organized as following: Section 2 discusses what the Grid Services and Web Services can provide to online experiment environment; Section 3 presents the general framework; Section 4 designs the web services for instrument services; Section 5 presents the web services composition solution; Section 6 describes a demo system, and; Section 7 concludes the paper.

WHY USE GRID SERVICES AND WEB SERVICES FOR ONLINE EXPERIMENT SYSTEMS

The enabling technologies for building an online experiment system come from two domains: the Grid Technology and the Web Services. Grid Technology focuses on using distributed heterogenous resources to solve massive computational problems. Grid Technology sees the world from the resources point of view. It has specifications for resource management, such as descriptions of resource properties and performance [8]. It uses scheduling and clustering functions to ensure the computational performance of the grid.

Web Services see the world from a service point of view. The services are provided by the software components over the internet. It uses internet protocols, such as HTTP, BEEP and XML technology to ensure the interoperability of the components on different platforms and is implemented in different programming languages. The components use SOAP message-based communication to talk to each other [3]. WSDL is used to define the interfaces of the components [4]. UDDI is used for service discovery and integration [5]. All of these specifications are W3C standards and are widely supported by industrial companies, such as IBM, Microsoft and BEA and others. Compared with the preceding middleware technology, Web Services are more interoperable and accessible. The development and deployment complexity are greatly reduced. Therefore, Web Services are becoming a widely accepted technology.

Grid Services were presented by Ian Foster in 2003 [6][7]. It proposes to use the Grid technology to manage resources and computational issues, and to use Web Services for the solution of middleware. It aims to take the advantages of both technologies. The resources are wrapped as Web Services. Open Grid Service Architecture (OGSA) [6] proposes to use GWSDL (Grid WSDL) to describe the interface of the remote operations. GWSDL extends WSDL by defining some portTypes (a WSDL tag for remote port) designed for Grid Services, e.g. GridServices and NotificationSource and more. GWSDL also defines Service Data Elements, such as SystemInfo (#CPU, system load, CPUBrand), and LastResults (Internal values). OGSA proposes functions such as Notification Service, Transient Service, Logging, Lifecycle Management, and Security. These services are not in the Web Service specifications. Grid Services use *Index Server* for service discovery. Its function is similar to UDDI server, but it uses different properties to describe a service.

Grid Services and Web Services will converge when the Web Services Resource Framework (WSRF) is released at the beginning of 2005 [9]. WSRF defines a family of specifications for accessing stateful (i.e. maintain state information between message calls) resources using Web Services. It includes the WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, and WS-ServiceGroup specifications. The motivation for these new specifications is that, while Web Service implementations typically do not maintain state information during their interactions, their interfaces must frequently allow for the manipulation of state, that is, data values that persist across and evolve as a result of Web Service interactions. WSRF fills out the Web Services stack to be consistent with the Open Grid Services Infrastructure.

For online experiments, we have computational tasks and data sharing tasks, as well as tasks to use the services of instruments or devices. Thus we deal with high performance issues as well as service composition issues. The Grid Services have solutions to computational resources management and data transferring. The Web Services have the solutions to services integration and composition. In our proposed framework, we will use both.

We notice that this is a rapidly changing domain. The trend is to merge the Grid Services and Web Services. A web service is going to have the properties of the computational resources when it wraps a computational resource. When WSRF is in use in 2005, we will see this more clearly. Currently, our framework considers the specifications from both the Grid Services and Web Services. We expect in the near future that the gap between the two domains will disappear.

THE GENERAL FRAMEWORK FOR ONLINE EXPERIMENT SYSTEM USING GRID SERVICES AND WEB SERVICES

An online laboratory system uses the scattered computational resources and instrument services on the networks for experiments. The online laboratory system we present here is a *web enabled distributed system*. It has two meanings: the user accesses the online laboratory system via web interface; the heterogenous resources and devices interoperate with each other via Web services standards. The goals of this framework are: 1) sharing the experimental resources among different labs via the Internet; 2) increasing the ability of computation and sharing data among different labs, and; 3) enabling users to access online labs any time and from anywhere.

Figure1 is the distributed system architecture. Its front end is web-based, i.e. a web server is used to render the GUI interface (see the next section). Its backend has the functions to manage the students and manage the experiments. Most importantly, the backend can use scattered resources on the Internet for one experiment. For example, it can use instruments and devices from different online laboratories for one experiment, and it can

use heterogenous computational resources to process the data generated from the experiment.

Web services serve as the transport layer of the system. The computational resources and the instrument services are wrapped by WSDL. So we just use "web services" for all kinds of resources and services. SOAP messages are sent to invoke a service. These protocols are widely accepted by different operation system and implemented by different programming languages. Therefore, the interoperability is ensured. To make the system work, a service provider first registers its service in a registry server (step 1 in figure 1). Web services use UDDI server, while Grid Services use Index Server. We expect the two standards will be merged. Otherwise, our system will accept both standards. A service requestor searches the registry server and gets all the potential resources. It selects the proper services based on its own criteria (step 2). The service requestor sends SOAP messages directly to the service provider to invoke the remote service (step 3).

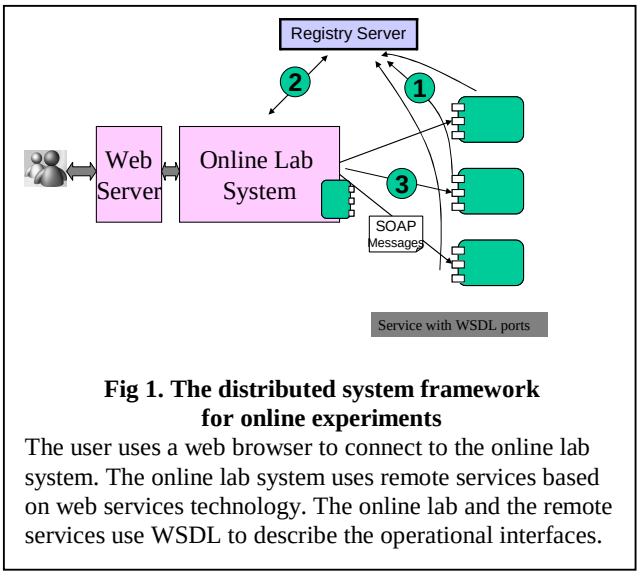

Fig 1. The distributed system framework for online experiments

The user uses a web browser to connect to the online lab system. The online lab system uses remote services based on web services technology. The online lab and the remote services use WSDL to describe the operational interfaces.

The architecture for the online experiment system is shown in Figure 2. The block below the "hosting environment" is the online laboratory system. It uses a web server for the front end representation. The back end has three layers. The top layer is the logic layer, where the learning scenarios are defined and the processes are managed. The learning scenarios are defined in four aspects [10]: learning objects, a pedagogical model, a media model and distribution. Among those, the pedagogical model defines the process of a course. The process is translated directly into Business Process Execution Language (BPEL) [11]. The ***BEPL engine*** is a tool to monitor and control the process automatically. The BEPL engine is able to automatically invoke remote web services. The activities in a learning scenario may need remote web services. The ***Service Broker*** determines if the services come from local services (e.g. the blocks under the "*LAN*"), or remote external services (e.g. the blocks of "*jini services*", "*web services*"). Service Broker knows the different protocols the remote services use. For Grid services, it sends the requests to the GSH/GSR (Grid Service Handler/Grid Service References) in the ***Grid Container***. GSH/GSR is a mechanism in Grid Service to get the reference of the remote objects and forward the requests to the remote objects. GSH/GSR is able to invoke the services either in middleware, (e.g. jini), or in web services. Service Broker can also invoke web services without the GSH/GSR interface by sending the request to the service objects in the application server (the bottom layer). Service Broker regularly calls the ***Service Lookup*** ("*srv lookup*" in Figure 2) and updates the local ***LDAP*** with the results. ***Registration Manager*** ("*Registration Mrg*" in Figure 2) helps to convert information from a service registry into LDAP. The bottom layer is the ***Application Server*** layer. The Application Server provides flexible mechanisms to manage the ***Service Objects*** and interface to the ***Web Service Engine***. Service Objects are some software components that process the data from remote web services. See the next section for one example of service implementation. The Web Service Engine sends the SOAP message to invoke the remote web services. This framework works with the computing resources using Grid protocols, software components using middleware, and web services components. Thus we think it covers all the resources needed for online experiments. As Grid Services will merge with Web Services in the future, we believe that the two lower layers in Figure 2 will, at some point, be united into one layer.

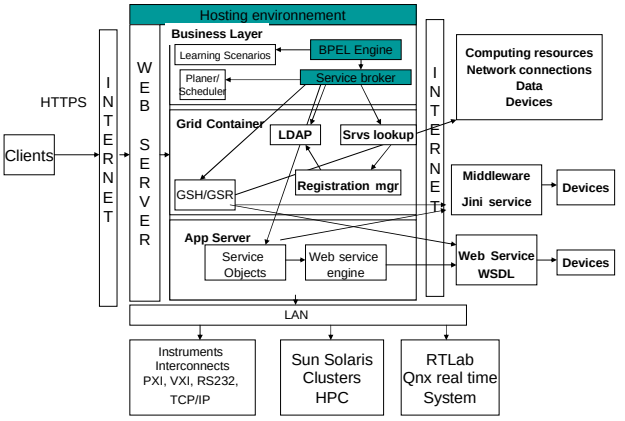

Fig2. Architecture of the online experiment system

DESIGN AN INSTRUMENT WEB SERVICE

An instrument service is a basic service in our application. In this section, we discuss how to wrap an instrument into a web service. Here we only consider an instrument service as a remote operation to invoke. We may need to use Grid to process the data generated by the instrument; however, that is beyond the scope of this paper. To design an instrument web service, we are concerned with the three following issues:

Design the GUI: an instrument has its individualized panel. In the Lornet project, we studied how to display the panel as a java distributed application [12]. DMM is the xml schema to define the syntax of an instrument panel. An xml file compliant to the DMM is a description of the panel. In [12], the xml file is parsed by JAXB, and its components are mapped to java AWT components. The DMM schema is defined in such a way that it is a straightforward 1-1 mapping between the panel objects and java AWT objects. In [12], the system is implemented using Jini technology. The xml is downloaded from the Jini

registration server and displayed on client side as a Java application. The user can operate the remote object from the GUI interface.

In this paper, our online experiment system is web-based. We want zero installation at the client side. The user needs only a browser to access the online laboratory any time and anywhere. We inherited part of the existing work. Instead of using AWT classes, we map the panel objects to JSP objects. The JAXB binding is inherited. The GUI generation principle is displayed in Figure 3 using the Agilent 33401A as an example.

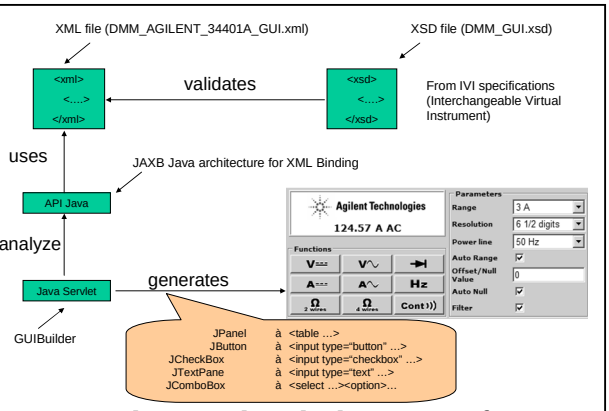

Fig3. The principle to display GUI interface using JSP objects

Design the WSDL for the instrument services:

The invocation of an instrument service is similar to any other web services. WSDL is a description of the remote operations (remote functions) and the arguments of the operations (type of the arguments and the sequential input order).

We use the multimeter Agilent 33401A as the example. In Table 1, we show how the remote operation ***getfunctionsresult*** for Agilent 33401A is defined. It is an *in-out* type service, which means it receives the input arguments embedded in a request SOAP message and returns the results in a response SOAP message. The bottom part of Table 1 defines the

```
<!--define the request message -->
<wsdl:message name="getfunctionsresultRequest">
  <wsdl:part name="in0" type="xsd:float" />
  <wsdl:part name="in1" type="xsd:float" />
</wsdl:message>

<!--define the response message -->
<wsdl:message name="getfunctionsresultResponse">
  <wsdl:part name="getfunctionsresultReturn"
type="xsd:float" />
  </wsdl:message>

<!--define the operation -->
<wsdl:operation name="getfunctionsresult">
  <wsdl:input name="getfunctionsresultRequest">
  </wsdl:input>
 <wsdl:output name="getfunctionsresultResponse">
  </wsdl:output>
</wsdl:operation>
```

Table 1: the snippets of WSDL for Agilent 33401A

operation. The two messages of ***getfunctionsresultRequest*** and ***getfunctionsresultResponse*** are defined at the beginning of the WSDL. ***getfunctionsRequest*** requires two float typed arguments, and ***getfunctionsResponse*** requires one float typed argument. We defined other operations, such as ***setVoltValue***, ***setOhmValue***, ***getVoltRange***, ***getOhmRange*** for this instrument. These operation names are actually the method names of the remote service objects.

The advanced requirements for instrument services:

While the main purpose for most existing web services is to provide information, the instrument web services involve operating the physical devices in real time. Improper design of the web services can cause damage to the instrument and create false measurement and control, causing the online experiment to fail. Therefore, we present the special requirements for the instrument web services and partially present the solutions.

1. ***Stateful service***: the service tracks the user information. It records the operations from one user and controls how the user can use the service. The states of service for a user are controlled. We use an application server to achieve a stateful service. The application server provides functions such as persistence and transaction.

2. ***Performance issues***. Generally speaking, web service is slower than middleware for two main reasons. It has more transport layers than middleware; the overhead of using SOAP, e.g. composing SOAP delays transport, and the payload of SOAP messages are much bigger than necessary. The latency of networking can cause the user to lose control of the device and even cause damage. We are working on benchmarking web services for online experiment applications. We want to answer two questions: 1) what are the metrics of QoS of the web services for this kind of application, and; 2) how do we adapt the instrument service to meet the QoS for different network conditions? Some technologies are available to optimize the performance. For example, we can use Abstract Syntax Notation (ASN) to save the payload, or we can keep the connection open to save transport time.

3. ***Server side reliability mechanisms.*** For an online experiment system built on web services, the signal generator, the measurement instrument and the testing devices may not be in a same physical location. The latency of signal can cause faulty measurement, and breaking the connections can lead to physical damage**.** When the experiment is about controlling a remote device, the control strategy has to consider the non-real time effects. We think a service has to have the following mechanism to improve its reliability:

a. *time out mechanism*. When the user does not give further instruction, the service cuts the connection.

b. *attach time stamp when transferring signal*. The time stamp marks the time point of the event. The measurement can happen after the event. The time stamp tells the true time of an event.

c. *trend predication for some critical variables*. The predication can be used to shut down the device when the variables go out of norm values, or it can be used to adjust the control strategies.

d. *proactively response to exceptions*. These exceptions can be both hardware or software exceptions.

We need to look into the specific experiments to design these mechanisms.

DESIGN THE PLANNER MODULE FOR SERVICE COMPOSITION

Composing Web services rather than accessing a single service is essential and offers more benefits to users. Composition addresses the situation of a user's request that cannot be satisfied by available component services, whereas a composite service obtained by combining the available services might be used for satisfying the request [13]. For online experiments, we can select the instruments and devices scattered over the Internet for one experiment, or we can select the experiments from different locations for one course.

Web service composition involves service discovery and service integration. Service discovery normally is a key word search process on UDDI. We can get detailed descriptions of the services in the UDDI registry. Service integration is to select the best composition from the potential partners and determine the interaction and sequences between them for a new service.

In the rest of this section, we present our solution of service composition using artificial intelligence (AI) planning [14].

AI planning is fundamentally based on search techniques in the problem space. The classic AI planning includes forward chain search, backward chain search and many other varieties from the two basic techniques. The result of an AI planner is the ground formula that describes the sequential/parallel orders of the activities that satisfy all the constraints. There are many sound and complete AI planners. The complexity of AI planning is NP-complete.

The web service composition problem can be modelled as planning problem.

Definition 1 Let $L=\{p_1,\ldots, p_n\}$be a finite set of proposition symbols. A ***planning domain*** on L is a restricted state-transition system $\sum= \{A,S, \gamma\}$:

- $A = \{a_1,a_2,\ldots a_n\}$ is the set of available actions (services for web services composition);
- An action a_i is a triple $a_i=(\text{precond}(a_i), \text{effects}^-(a_i), \text{effects}^+(a_i))$. The set $\text{precond}(a_i)$ is the preconditions of a_i and the sets $\text{effects}^-(a_i)$, $\text{effects}^+(a_i)$ are called the effects of a_i;
- $S = \{s_i,s_2,\ldots,s_n\}$ is the set of states (the states in problem space). An action a_i is applicable to s_j, if s_j satisfies the $\text{precond}(a_i)$.
- γ is a transition function which defines the effects of applying a_i to s_j: $\gamma(s_j,a_i)=(\text{s-effects}^-(a_i))\cup\text{effect}^+(a_i))$, and $\gamma(s_j,a_i)\in S$.

Definition 2 A ***planning problem*** is a triple $P=(\sum, s_0, g)$, where:

- $s_0 \in S$ is the initial sate
- $g\subseteq L$ is a set of propositions called goal propositions that give the requirements that a state must satisfy in order to be a goal state. The set of goal states is defined as follows: $S_g=\{s\in S| g\subseteq s\}$.

We can describe the instrument selection problem in the following way:

Definition 3 A ***plan*** π is a sequence of actions $\pi = \langle a_i, \ldots, a_k\rangle$ which is a solution for P if $g\subseteq \gamma(s_0, \pi)$.

The planning problem gives the solution to determine the sequence order of the services to satisfy the requirements of the new service. However, web services composition has its own characteristics which are out of the definition of planning. First, only the most optimal plan will become the new composed service. Though a planner can provide complete solutions, it is just a distraction. Domain-based utility function will be designed to evaluate the plans and select the best one. Additionally, the problem space is not finite. For example, it is not possible to extract all the available time slots for all the instruments. We can only get the availability information by sending a SOAP query to the service. Only part of the data is exposed to us. Second, the problem space is not finite. For example, the schedule of an instrument may be managed by a booking system. We do not own the database of the booking system. We can only use SOAP queries to get the availability information on a certain time interval. Thus, though it is possible to explore the whole problem space, practically the problem space is partially exposed.

We propose an ***incremental planning process*** to solve the above problems. We have an AI planner as the core of the composition module. An evolution algorithm, such as Genetic Algorithm (GA), is used to select the best plans and guide the incremental planning process. The evolution an algorithm provides the optimization possibility and explores the problem space incrementally. When the algorithm is used in a scenario in which multiple partners are involved, the partners can contribute to the direction of search at each turn of evolution. This was first presented by [15].

The incremental planning process works in this way: we select *n* web services as the inputs of the planner. These are the first generation chromosomes for the GA. They are some points in the search space. The planner will generate all possible plans. A domain-based utility function is designed to evaluate the plans. We evaluate the plans in two dimensions; the cost of the service and the preferences of the time slots. The best *m* plans will be kept, all others will be abandoned. The *m* plans correspond to *n'* web services. The *n'* web services will pass the operations of GA, i.e. the crossover and mutation. These operations help the optimization to jump out of local optimal points. One possible effect of crossover and mutation is that it varies the time slot for a web service. Then, we need to query the web service again to get the availability information. The new *n* web services are given to the planner again. The new loop starts. Normally we can expect that after generations, the process is converged to some global optimal points. That is the final plan we want.

A simple version of the web service composition problem is that the process of an experiment is fixed. That means we do not need to determine the sequence of activities. Then the composition problem is simplified as a pure optimization problem. We can just unplug the planner from the process.

We will present the incremental planning in a separate paper.

Context mediation of Web Services composition

Because Web services originate from different providers, thus are heterogeneous, achieving the semantic composition of Web services consists of resolving the semantic heterogeneity conflicts among Web services. These semantic conflicts arise when (i) the same concept has different data structures, (ii) the same concept has different meanings in different Web services, (iii) different values have the same meaning, or (iv) the meaning of a value change from one context (application) to another.

In Web Services composition with semantic data exchange (Figure 4), these Web Services interfaces are enriched with a contextual description. Its objective is to associate context with input and output parameters. Context is any information that is relevant to the interactions between a user and an environment. Therefore, the values exchanged between Web services are not simple values; instead they are semantic values (i.e., no ambiguities in their meaning).

The context information that is associated with a Web service supports the composition of a Web service process to automatically whether data conversion is needed. This conversion is done by an active component called context mediator in [16]. The conversion functions are defined outside the Web services and can be shared by all Web services.

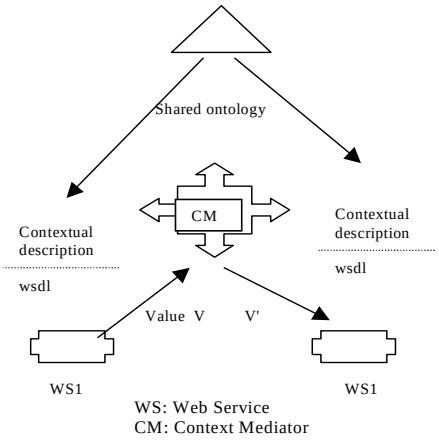

Fig4. Web Services Composition with context mediation

SOME SNAPSHOTS OF THE DEMO SYSTEM

We built a demo system to verify and illustrate the feasibility of our ideas. The demo system shows how a user can connect to the online lab using a web browser and operate the instruments wrapped as web services. The other sophisticated features as discussed in this paper are not implemented in the demo system. Web server layer uses Apache Tomcat. It is a servlet and JSP container. The GUI is implemented by JSP and servlet. The SOAP engine is an Apache AXIS engine. The local service object is implemented by servlet. BPWS4J is the IBM BPEL server. It can manage the experiment process and handle the remote service invocation.

Figure 5 shows the principle of the demo system. The user connects to the online lab server through a web browser. He/she meets the login page. After login, the online lab system displays a list of available instruments. Since some of the instruments are remote web services, the online lab system actually "downloads" an xml file of remote instrument lists (labinstruments.xml in Figure 5). The online lab system parses the xml and lists the instruments to the user. The user can select an instrument from web page (jsp pages). For example, the user selects a multimeter Agilent 34401A. The online lab system will go to the web services of this instrument, "download" the xml for the description of its panel (Agilent34401A_Gui.xml in Figure 5). The xml is parsed, and the jsp page for displaying the panel is dynamically generated by the GUI builder. Then the user can operate the instrument via the GUI. The commands of the operations are sent to the instrument manager. The instrument manager encodes the command into SOAP messages according to the WSDL of the instrument service, and sends the SOAP to the instrument service. The instrument manager receives the SOAP response messages and decodes the returned data. The instrument manager calls the GUI builder to update the GUI so that the received data can be displayed on the panel.

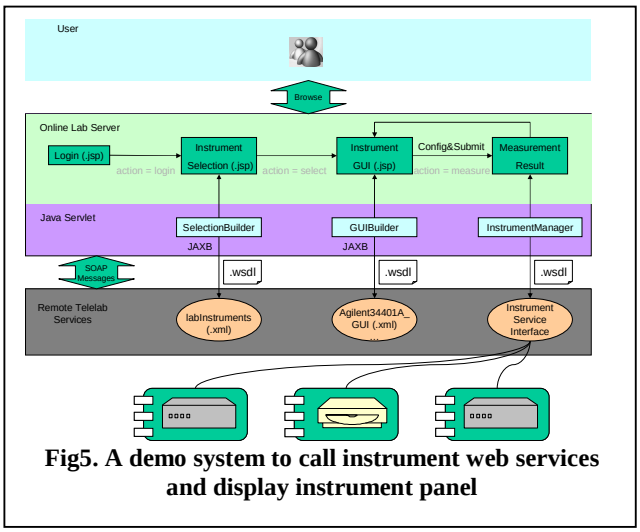

Fig5. A demo system to call instrument web services and display instrument panel

CONCLUSIONS

Our system framework can be evaluated using the following characteristics:

Interoperability: Web Service is language-independent and platform-independent.

Scalability: may not be an issue for online experiment, since the service can only accept one user.

Portability: java-based, portable for UNIX, Linux, and Windows.

Affordability: open source, affordable for e-learning applications, open source does not mean low quality. Many of the open source products we used are among the best.

Accessibility: from any web browser

In this paper, we discussed some advanced features for online experiment web services. That is the direction in which we are currently working.

REFERENCES

1. Chung, J.Y., K.J. Lin, and R.G. Mathieu. Web Services computing. Advancing Software Interoperability. *IEEE Computer*, 36(10), Octobre 2003.
2. Curbera, F., R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. *The next step in Web Services. Communications of the ACM*, 46(10), October 2003.
3. SOAP Specification, http://www.w3.org/TR/soap12-part1/
4. WSDL Specification, http://www.w3.org/TR/wsdl
5. UDDI homepage, http://uddi.org/pubs/uddi_v3.htm
6. Foster, I., C. Kesselman, J. M. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, *http://www.globus.org/research/papers/ogsa.pdf.*
7. Foster, I., C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *http://www.globus.org/research/papers/atatomy.pdf*
8. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
9. OASIS, OASIS Web Service Resource Framework, *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf*
10. Paquette, G., Meta-knowledge Representation for Learning Scenarios Engineering, *Proceedings of AIEd'99*, Le Mans, France, July 99.
11. Andrews, T., Curbera. F., *et al*., Specification : Business Process Execution Language for Web Services Version 1.1, *http://www-128.ibm.com/developerworks/library/ws-bpel/*
12. Fattouh, B. and H. H. Saliah, Model for a Distributed Telelaboratory Interface Generator, *Proceedings of Int. Conf. On Engineering Education and Research*, Czech Republic, June 27-30, 2004.
13. Berardi, D. D. Calvanesse, G. De Giacomo, M. Lenzirini and M. Marcella. A foundational Vision for e-services. *In Proc. Of the Workshop on Web services, e-Business, and the semantic web (WES'2003)*, held in conjunction with CAISE 2003, Austria, 2003.
14. Ghallab, M., D. Nau, and P. Traversp, "*Automated Planning: theory and practice*", Elsevier, 2004.
15. Yuhong Yan, etc. "A Genetic Algorithm for Conflict Resolution in Concurrent Production Development", *IEEE Int. Conf. on Man System and Cybernetics*, Orlando, USA, Oct. 1997.
16. Sciore, E., M. Siegel, and A. Rosenthal: Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems,*19 (2), 1994.