Interoperability of Remote Laboratories Systems

doi:10.3991/ijoe.v6s1.1387

Herbert Yeung, David Lowe, Steve Murray
University of Technology Sydney, Australia

***Abstract*—There has been growing interest in, and development of, remotely accessible laboratories as a mechanism for improving access and flexibility, and enabling sharing of facilities. Differences in focus, philosophy, approach or domain have led to quite different technical solutions in supporting remote laboratories. Whilst this diversity represents a significant strength in terms of the ability to explore different issues and support diverse applications, it does however potentially hamper the sharing of labs between different institutions. Investigation into interoperability between two remote lab platforms has realized a need for a common application protocol to achieve the goals remote labs aims to provide. We describe our approach to providing a bridge between two current remote laboratory architectures – Labshare's Sahara and MIT's iLabs – and report on the issues that arise with regard to the protocol translations.**

***Index Terms*—Interoperability, Laboratory, Remote, System.**

I. INTRODUCTION

Currently, there are numerous architectures and implementations of remote labs across different institutions and geographical locations [1] – each having arisen from a different set of pedagogic or technical design parameters, philosophical approach, or simple evolutionary pathways. Given that each architecture will have different strengths and weaknesses, and potentially address a different set of needs, we believe that this diversity is an asset that should be encouraged rather than avoided. The diversity can, however, create problems with regard to laboratories that are implemented based on one architecture not being accessible to users who have adopted an alternative architecture. This indicates that there are likely to be significant benefits to be gained by developing approaches that allow different systems to co-exist, but to also interoperate – i.e. to identify common interfaces that allow a laboratory that is developed and managed in one system to be utilized by users supported by an alternative system. In this paper we investigate this concept in the context of two specific architectures: Labshare's Sahara[1] [2] and MIT's iLabs[2] [3]. These two architectures were chosen as they are relatively mature architectures which offer similar and sometimes complementary functionality, but also have some notable differences [4]. Our analysis will demonstrate the feasibility of using a common protocol to achieve interoperability and the lessons gained in trying to achieve this. We begin in Section II by providing a brief overview of the Sahara and iLabs architectures and a comparison of their respective approaches, strength, and weaknesses. In Section III we compare the functionality supported in each architecture and how the functionality in each architecture can (or in some cases cannot) be mapped to the other architecture. In Section IV we discuss a protocol for communication between the two systems and how this can be used to support interoperability. In Section V we discuss design issues that are relevant to the implementation of this approach and in Section VI we consider conclusions and future work.

II. ARCHITECTURE OVERVIEW

A. Sahara Architecture

The early UTS (University of Technology, Sydney) remote laboratory system dates to the period 2000-2005, and was originally developed to allow students to have flexible access to limited laboratory resources [2]. This system – which has subsequently come to be referred to as Sahara (Release 1) – was then adopted as the basis for the much broader Labshare project[1]. This project is aiming to establish a national approach, within Australia, to the use of remote laboratory technologies in support of the sharing of laboratory facilities between educational institutions. As a consequence Sahara has undergone a major redesign, initially to provide a much more scalable and robust basis for laboratory development (primarily the basis of Sahara Release 2), and subsequently to provide support for distributed user management and access accounting, amongst other adaptations (Sahara Release 3 and onwards). Despite these changes, Sahara has largely retained the core architectural elements that were incorporated in the earliest versions of the system.

One of the advantages of the Sahara design is the ability to have a short turnaround time and low costs in creating a new experimental rig (see [5, 6] for examples). This is due to a simple protocol and configuration for implementation purposes.

With Sahara Release Two, this concept has been extended with a requirement for some of the limitations from release one to be resolved. One of the limitations that is fixed in release 2 that existed in release one is the restricted number of connections that could be established due to database constraints and the number of processes involved. Another limitation that has been addressed is the non-functional requirement of extensibility. With the adoption of a SOAP application protocol interface (API) and the ability to add other capabilities in planned future releases, extensibility has become a prime requirement. Portability has also been considered, to allow the system to be installed on varying operating system architectures with the adoption of Java, Apache, PostgresSQL and PHP, which are all cross-platform.

Proposed changes in release 3 and beyond will allow the system to be accessible from different institutions, as part of the Labshare program. This will culminate in the

[1] See http://www.labshare.edu.au

[2] See https://wikis.mit.edu/confluence/display/ILAB2/Home

ability to allow different institutions to share rigs and possibly other features.

B. iLabs Architecture

iLabs was devised from the research by MIT to create a remote labs installation that would be able to be distributed readily across varying institutions in different geographical locations, but allow them to achieve the common goal of sharing labs. The premise was to be able to scale the architecture while at the same time offer an easy way for remote labs to be setup [3]. This led to a broker architecture being adopted [7]. This allows for all proxy agents to be managed under the guidance of the service broker and meant that any other components (known as proxy agents) can be installed on different machines.

Release 3 of iLabs has brought several significant changes. Noticeably, the biggest changes are to do with the support for Unicode, a more refined ticketing and coupon system as well as initial support for single sign on (SSO) capability.

The University of Queensland (UQ) has also conducted research and design in extending iLabs to provide a distributed means of sharing lab equipment, known as the 'lab farm'. This proposed architectural change to iLabs involves the addition of new components such as the 'experiment manager' and 'experiment engine', and removal of the lab-side scheduling server, thus shifting the functional responsibility towards the lab server and respective experiment manager, introduced to coordinate the 'lab farm' and assist in the scalability of adding or removing lab equipment [8].

C. Architecture Comparison

As depicted in Figure 1, Sahara is based on a client-server architecture. There are two forms of clients used. One is a web based thin client that allows the user to access the rigs via a web based interface, while another client provides a means for the server to manage the respective rigs. The core of Sahara is the Scheduling Server and this comprises a stack that consists of a persistence layer that sits at the bottom followed by a layer that manages the rig clients as well as the queuing, while at the top of the stack, the session creation and management is handled.

On the other hand, iLabs has adopted a broker architecture, as shown in Figure 2 and Figure 3. This architecture has the advantage of allowing for distributed components, which are centrally managed by a service broker component. This eases deployment, and allows different components to be installed on different physical platforms. Fig. 2, also shows the control flow that is made, in a holistic manner when a reservation is requested and how the data is transported, for an interactive experiment. Fig. 2 does not show the registration and tokenizing system that is adopted to allow for the authorization mechanisms that associate the different proxy servers. The batch experiment interfacing, shown in Fig. 3, can forgo the use of user side scheduling (USS) and lab side scheduling (LSS) as there is no need to manage session state as compared to an interactive experiment.

Both architectures have adopted a SOAP based interface, using extensible markup language (XML), for the means of communication. Also, the underlying core is still predominately based on a client-server architecture (in the case of iLabs it is segregated between the web client and the user side scheduling server) with the adoption of an off the shelf framework, as most remote lab implementations currently have used [9]. This is exemplified by iLabs currently adopting the ASP.NET framework while Sahara release 2 predominately operates on the Java OSGi framework.

III. FUNCTIONALITY MAPPING

The Sahara and iLabs architectures were developed to address varying needs by each host institution, and therefore have varying architectures, but also differing objectives, terminology, and usage patterns. In order to create a communications protocol that supports interoperability we

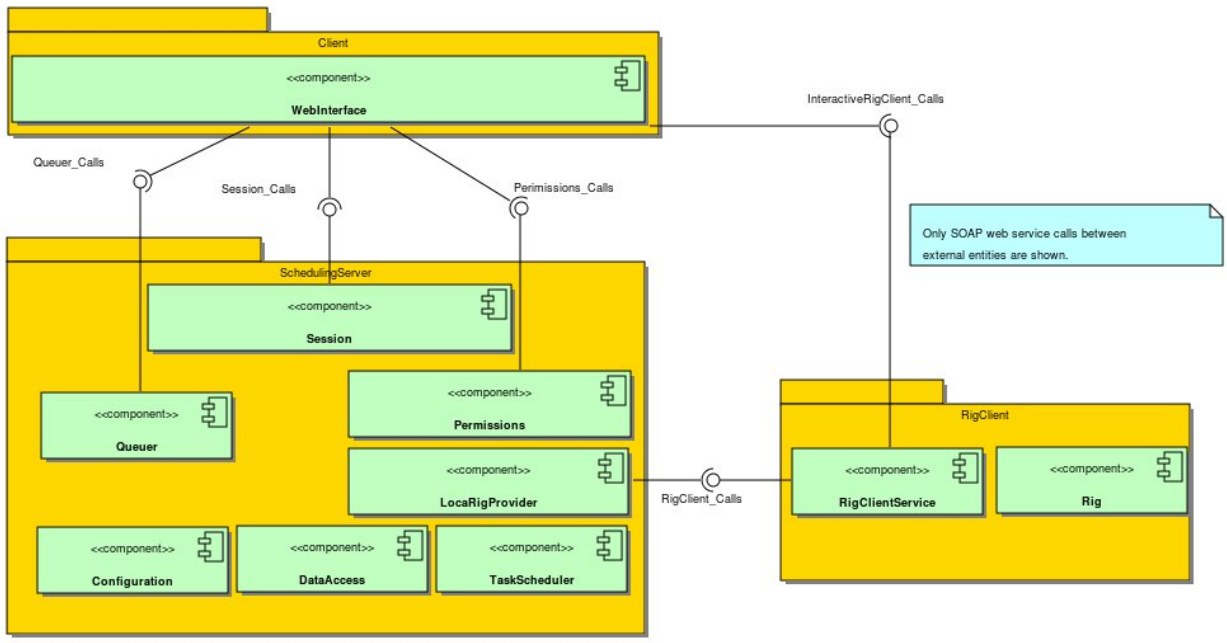

Figure 1. Component Diagram depicting interfaces of Sahara Release 2. Consists of Scheduling Server, Web Interface Client and RigClient

therefore need to consider the differences and similarities, and how these might affect our design.

We begin by considering a common nomenclature used between the two designs. Analyzing the nomenclature used not only assists in the understanding of both iLabs and Sahara but also allows for the development of a standardized communication platform, similar to developing an agent based service platform [10]. The functional mapping approach is similar to the schema mapping approach for achieving metadata interoperability [11]. Sahara came from an initial emphasis for the provision of engineering labs [12] whereas iLabs appears to have centered on accommodating science/applied science based labs [3].

These different domains, as well as different conceptualizations of the nature of the problems being addressed have led to the adoption of quite different terminologies. Table 1 draws a comparative mapping between the two separate nomenclatures to understand where certain commonalties can be reached. From the table, it can be seen, that the concept of a scheduling server holds true in both designs, while the concept of experiments from iLabs does not deviate significantly from the engineering nomenclature adopted by Sahara. One thing that is noticeably different in the terminology is the concept of the service broker, represented in iLabs but not in Sahara. This is due to the architectural differences between Sahara and iLabs.

In terms of developing a communication protocol for supporting interoperability, a key question relates to how we handle functionality that is required (or assumed) by components of one architecture, but which is not present in the other architecture. This requirement may be an operational requirement, or it may relate to functionality that is provided to the user. As an example, the iLabs Batch mode Lab Server provides a notification to the relevant

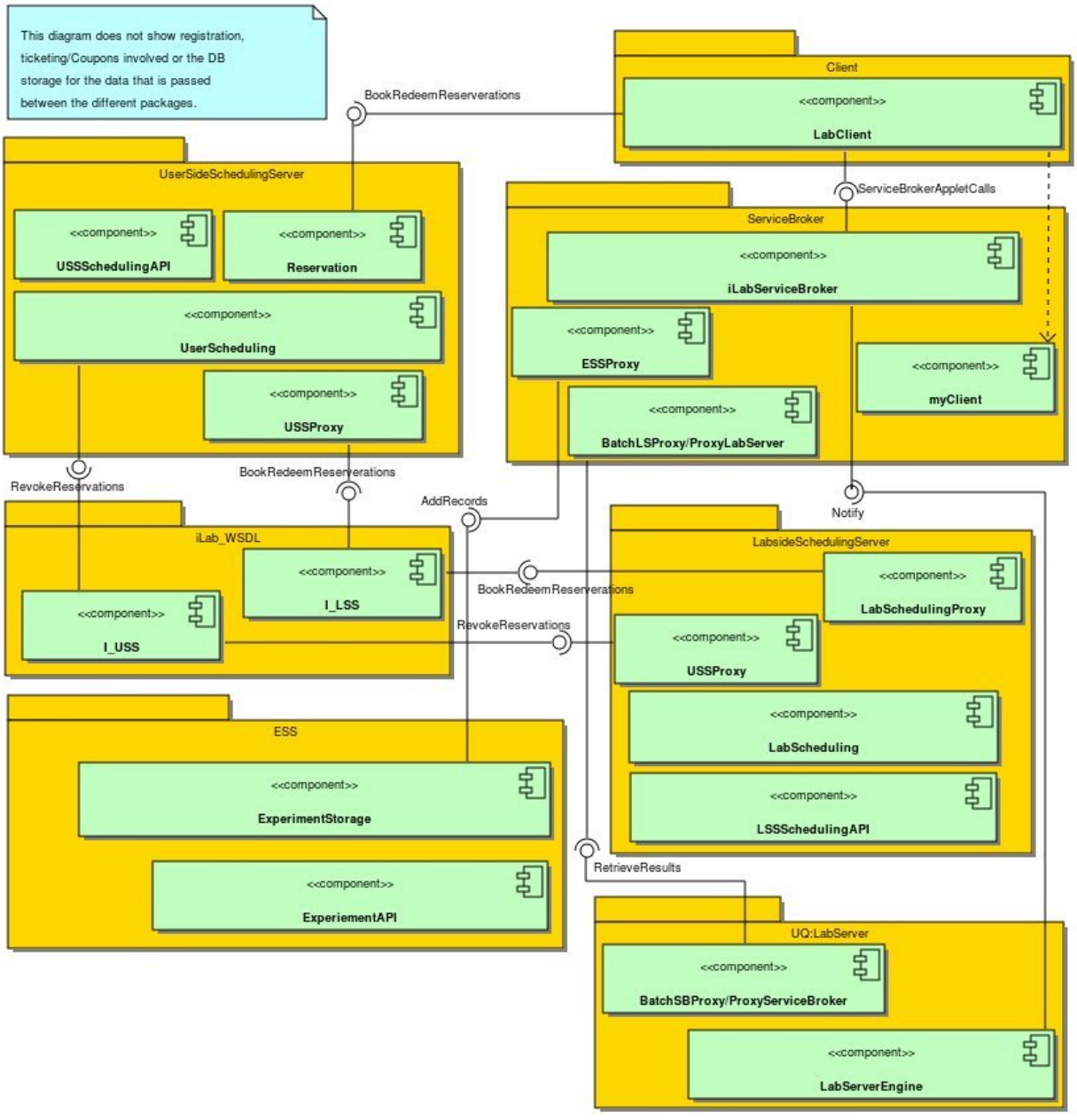

Figure 2. Component Diagram showing interfaces of iLabs version 3. Consists of an LSS, USS, iLabServiceBroker, ESS, Client and LabServer

Service Broker once a queued experiment has been completed to retrieve back the experiment results. No such functionality is provided in Sahara (primarily because it was designed to support interactive rather than batch laboratories).

The use of a web service calls mapping table (refer to Table 2) allows the identification of a set of common functionality that exists in both architectures. From the analysis it can be inferred that there is a relatively seamless mapping for most of the core functionality. However, there are disparities in some of the mappings, due to differing design decisions or understandings. For example, in Sahara, the validation process is conducted as part of the experiment submission process. However, this contrasts in iLabs where the validation process is treated separately. We needed to account for this in our design decisions, which required examining the costs and benefits of keeping a particular functionality in the common architecture protocol. Another aspect that Sahara offers that is not offered in iLabs is a more granular and flexible ability to control and administer interactive experiments via web service calls, such as the capability for a user being 'locked out of an experiment' (resulting in more granular authorization). As part of the design of the architecture, other things that need to be considered are the common functionality that is understood across both platforms. This can be categorized into the following areas: lab access, lab and experiment operations (encompassing status and execution), retrieval of results and the ability for the laboratory manager or administrative staff to conduct maintenance activities if need be.

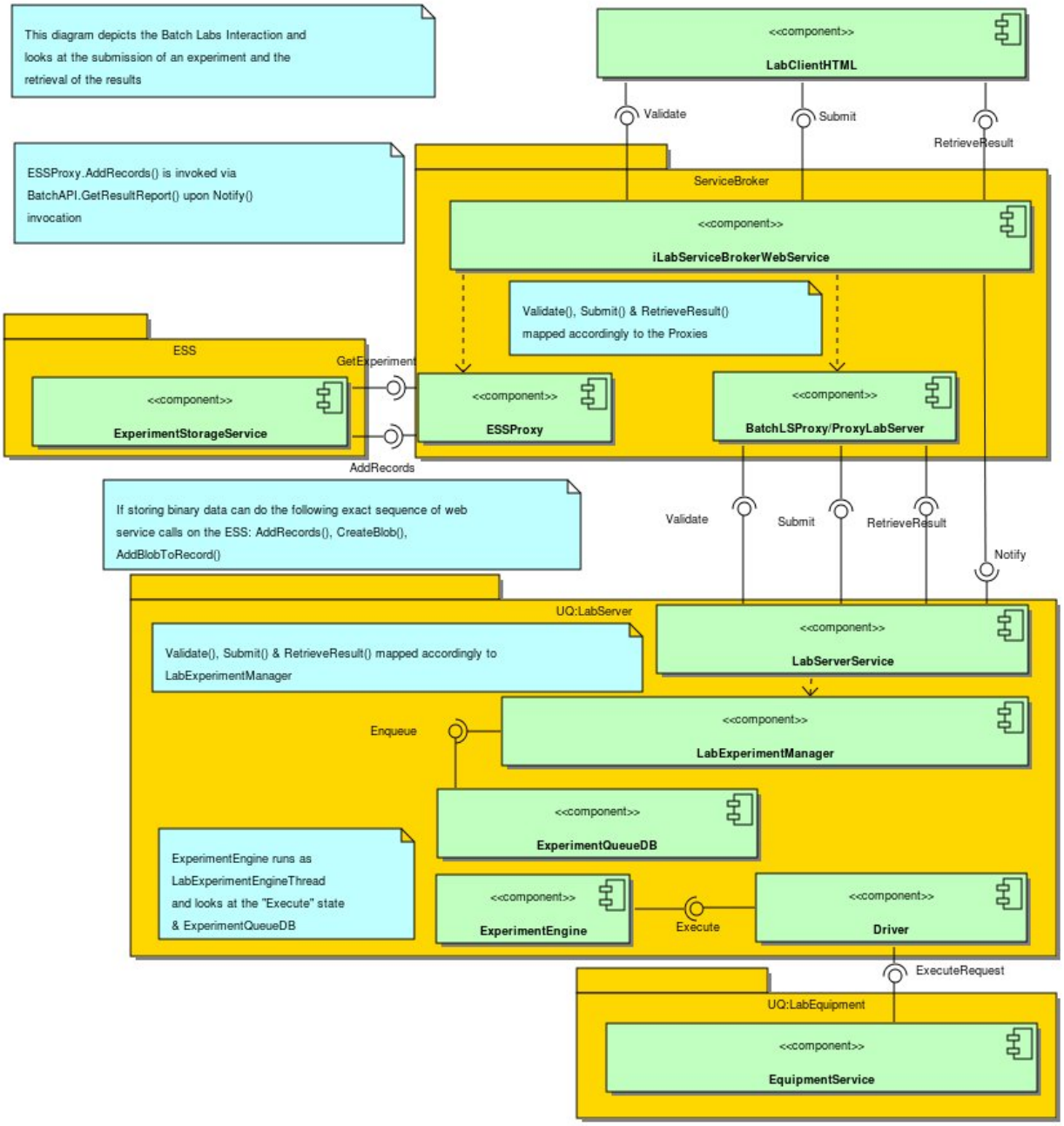

Figure 3. Component Diagram of iLabs Batch based experiment

Other commonalities that exist include the concept of queuing. Both architectures understand that requests for rigs/lab equipment needs to be queued. Both currently adopt a first in first out (FIFO) approach to queuing. There is also a common understanding between the differences of interactive versus batch labs [7].

IV. COMMON APPLICATION PROTOCOL

We propose that a common application protocol interface (API) be adopted that has the capability of offering both batch and interactive lab functionality, while maintaining the quality aspects of both existing systems, such as consistency, reusability and availability. Our defined interface to accomplish this is called LabConnector.

TABLE I.
NOMENCLATURE MAPPING BETWEEN SAHARA AND ILABS

Nomenclature Mapping	
Sahara	***iLabs***
Rig	LabEquipment
RigClient	LabServer
Capability	Annotation
Lesson	Experiment Specification
Queued	Experiment (Queue)
Session	Experiment (Running)
Scheduling Server	LSS + ESS
Distributed File System	Experiment Storage Server
(*No Equivalent*)	Service Broker

TABLE II.
WEB SERVICE CALLS MAPPING SNIPPET

Web Services API		
Sahara (SchedServer)	Common	iLabs (ServiceBroker)
performBatchControl	Yes	Submit
[No equivalent]	N/A	Register
[No equivalent]	No	RetrieveSpecifciation
performBatchControl	Redundant	Validate
abortBatchControl	Yes	Cancel
getBatchControlStatus	Yes	GetExperimentStatus
addUserClass	No	[No equivalent]

The LabConnector adopts both the proxy and façade design patterns [13] in order to offer a disparate number of web service calls as one, while at the same time offering the same interface for both iLabs and Sahara, respectively. This is shown in Figure 4, from the perspective of an iLabs user, whereby the *submit* and the *validate* message calls in iLabs are interfaced to LabConnector as though it is an iLabs LabServer. Conversely, Figure 5 shows that an Sahara user will use the LabConnector interface as though it is part of the Scheduling Server interface. The web interface across the LabConnector bridge is defined in the Web Service Definition Language (WSDL) file. This is, in turn, mapped to an equivalent experiment submission and validation call in Sahara. The full list of Web services is as follows:

submitExperiment	releaseExperiment
getExperimentResults	setUserPermissions
setMaintenanceTime	scheduleBookingTime
saveUserExperimentInput	saveExperimentResults
releaseSlave	getUserPermissions
getSavedUserExperimentInput	getToken
getMaintenanceTime	getLabStatus
getLabInfo	getLabID
getInteractiveExperimentSession	getExperimentType
getExperimentStatus	getExperimentSpecs
deleteSavedUserExperimentInput	getExperimentID
cancelMaintenanceTime	cancelBookingTime

An example of the WSDL used can be shown below for the *getExperimentType* web method:

```
<xsd:element name="getExperimentType">
  <xsd:annotation>
  </xsd:annotation>
  <xsd:complexType>
  <xsd:sequence>
     <xsd:element name="experimentID" type="xsd:int"
           maxOccurs="1" minOccurs="1"/>
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="getExperimentTypeResponse">
   <xsd:complexType>
   <xsd:sequence>
      <xsd:element name="experimentTypeValue"
           type="tns:ExperimentType"/>
   </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

The common web service calls chosen for the LabConnector bridge were based on a few criteria. The first was choice of nomenclature, with the term 'labs' and 'experiment' being a common term not only with iLabs, but also used interchangeably between other architectures, such as WebLab [14], WebLab-Duesto [15] and Lab2go [16].

From the analysis of the functionality mapping, all functionality that was common between Sahara and iLabs was directly adopted in the LabConnector architecture. Conversely, any functionality that was present in one architecture but not the other (i.e. present in iLabs, but not in Sahara, or vice versa) was separately evaluated. If this partially-supported functionality was considered a specific feature that was paramount to the workings of either batched or interactive labs then the functionality was included in the LabConnector definitions. An example of this is the URL to access the remote desktop interface present in iLabs. We also considered that if the partially-supported functionality offered a comparative advantage then it was incorporated as well. Designing the system showed that iLabs offered a relative comparative advantage in batched labs while similarly Sahara offered advantages for interactive labs usage. The basis of this is that Sahara has predominately dealt with interactive labs. This has led to the maturity of the design and also choice of particular functionality, such as the ability to allow master/slave usage. Similarly, iLabs has predominately been active in the batch labs design and this has led to the maturity in its design decisions, such as the concept of being able to define an experiment specification. For these reasons, LabConnector has adopted some of the interactive functionality that exists in Sahara while also complementing with the batch lab functionality from iLabs.

The choice of data format was also considered as well as data persistence (including persistence of experimental results). This was based on the use of XML, which is adopted in other frameworks, including RELATED [17]. This is a generally accepted approach to achieve remote lab interoperability, especially with the use of web services [18]. We have also adopted SOAP as the messaging protocol as this is not only a standardized protocol but is used in other remote labs, such as the Distributed Control Lab (DCL) [19] and is also considered in generic online lab frameworks [9].

V. DESIGN CONCERNS

The LabConnector has limitations. As it tries to restrict the use of disparate web service calls, this commonality has meant that the system needs to be rigid and cannot offer such concepts as 'register' (ability to create a dynamic web service call for a lab) that iLabs provides.

The current design of the LabConnector also does not consider certain features which exist on other platforms (neither implemented on Sahara release 2 nor iLabs version 3.0). We would however expect that ongoing work on LabConnector will consider similar mapping to that carried out above for numerous other architectures. For instance LiLa offers a directory service using the Shareable Content Object Reference Model (SCORM) protocol [20]. This is very similar to the proposed cataloging system that is being considered for latter releases of Sahara. This offers a means for the formation of institutional networks to show a directory service of the available labs across dif-

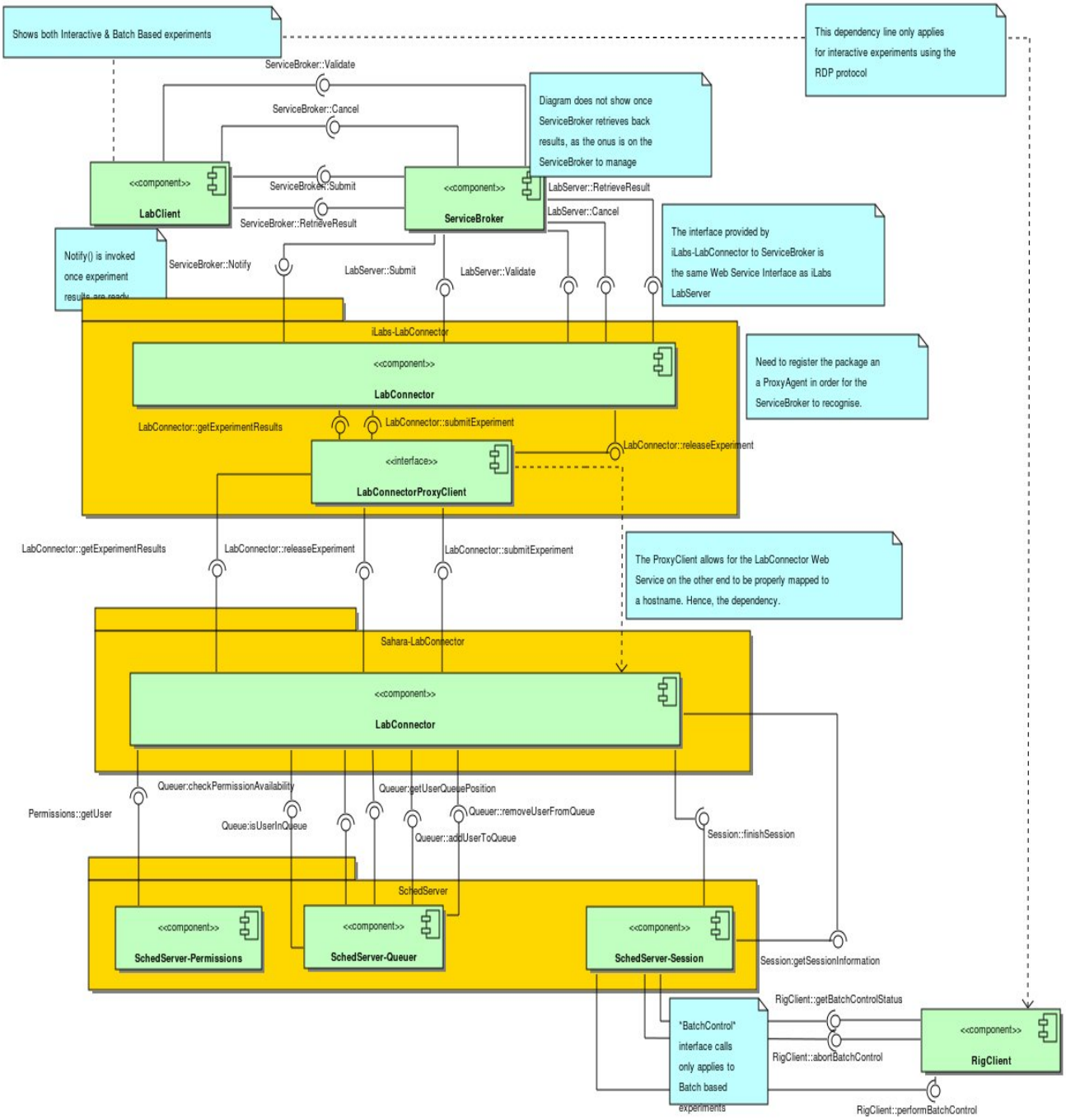

Figure 4. Component Diagram of LabConnector (perspective from Sahara user)

ferent institutions and will facilitate more efficient sharing. Lab2Go is another implementation that needs to be considered. This has a different approach, with the adoption of using a wiki based system [16]. Another implementation is WebLab-Deusto which offers a web based architecture, based on the Python platform [15]. The current implementation of the system allows for a light weight platform, unlike Sahara and iLabs.

There could also be performance and latency issues inherent with the use of web services and SOAP, but these should be able to be addressed by the adoption of optimization techniques such as reducing the size of the SOAP message or the number of SOAP messages [9] that are sent between the systems in the LabConnector architecture.

VI. IMPLEMENTATION AND EVALUATION

At the time of writing the LabConnector and associated components have been implemented for accessing an iLabs Lab Server (and hence rig) from a Sahara server. To demonstrate the concept, an iLabs radioactivity experiment located at the University of Queensland has been connected to a Sahara server located at the University of Technology, Sydney. Figure 6 shows a screendump of the Sahara interface during the submission of a set of parameters to the Radioactivity experiment, and Figure 7 shows the camera view of the hardware. The results returned from this experiment submission (obtained by selecting the "*Experiment Results Available"* button) are as follows:

```
<experimentResult>
  <timestamp>Mon 12 Jul 2010 11:50:37
        PM</timestamp>
  <title>Radioactivity</title>
  <version>3.1</version>
  <experimentId>124</experimentId>
  <unitId>0</unitId>
  <setupId>RadioactivityVsTime</setupId>
  <setupName>Radioactivity over Time</setupName>
  <sourceName>Strontium-90</sourceName>
  <absorberName>None</absorberName>
  <distance>20,40,60</distance>
  <duration>5</duration>
  <repeat>4</repeat>
  <dataType>Real</dataType>
  <dataVector>307,341,320,331</dataVector>
  <dataVector>123,119,129,126</dataVector>
  <dataVector>45,55,53,62</dataVector>
</experimentResult>
```

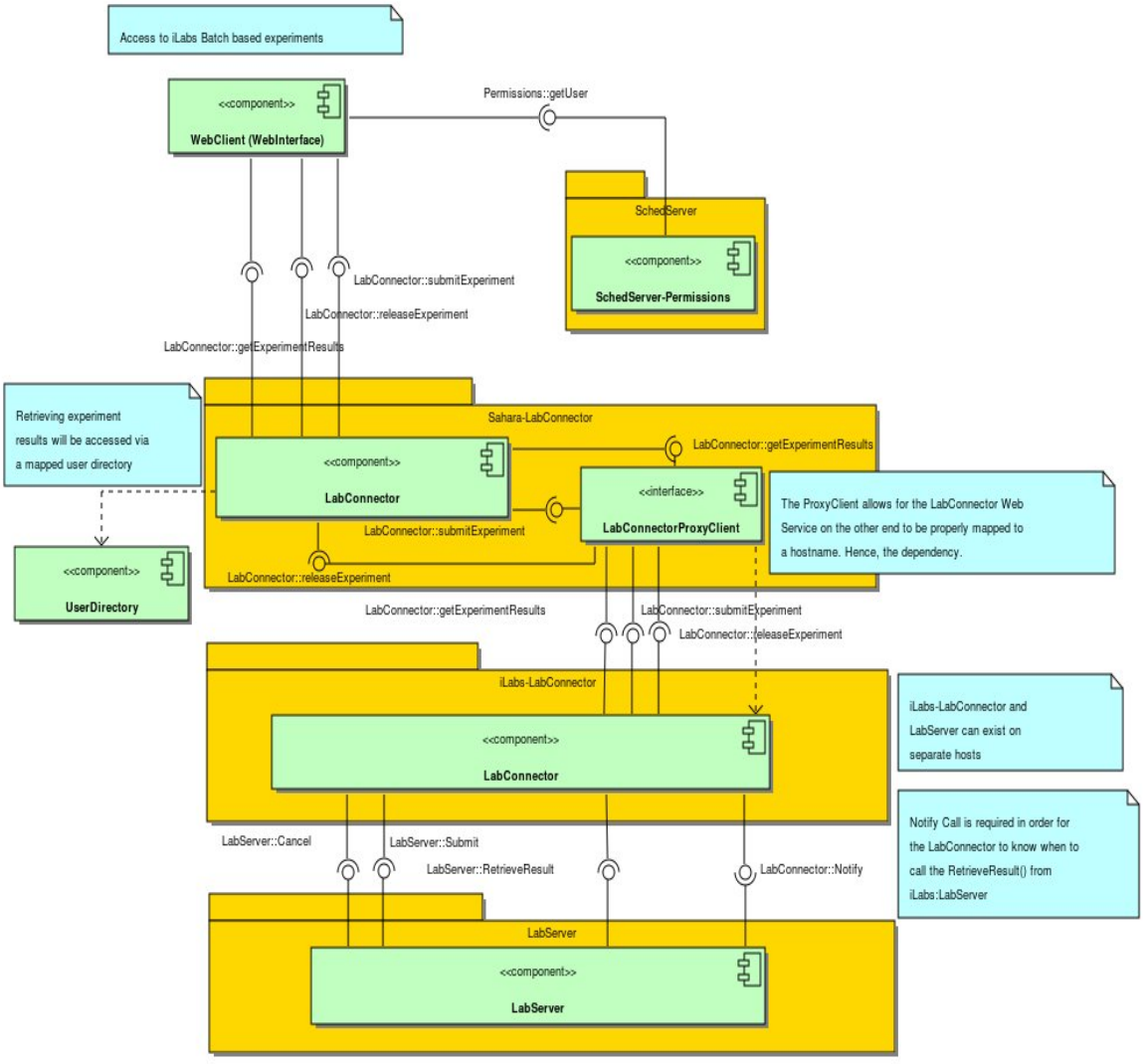

Figure 5. Component Diagram of LabConnector (perspective from iLabs user)

Figure 6. Radioactivity Experiment Camera View

This initial implementation demonstrates a successful bridge between Sahara and iLabs, though it is worth noting that the current implementation is only partially complete. We have yet to complete support for interactive iLabs experiments, or a bridge operating in the reverse direction – i.e. allowing Sahara-based rigs to be accessed through an iLabs service broker. Work on these aspects is ongoing and we do not expect their implementation to pose any significant difficulties.

The current bridge implementation of the LabConnector involves a sequence of events starting with a Sahara user accessing a rig that is designated of type 'ILABS'. When a user has filled in the appropriate input fields (UQ's radioactivity experiment for example requires the following input parameters: radioactivity setup, source name, distance, duration and repeat, as shown in Figure 6) and the submit button is invoked the submitExperiment() web service call from Sahara-LabConnector is called which is relayed to the iLabs-LabConnector.

Validation is conducted in the submitExperiment() web service call for the user identifier (userID) and the LabServer identifier (labID). UQ's LabServer submit() web service call is invoked upon validation. The iLabs-LabConnector submitExperiment() call gets back a unique experiment identifier (experimentID) which is tracked and persisted on file is returned back to Sahara-LabConnector. A visual notificaiton is displayed to the user that the experiment submission is successful if the experimentID is greater than or equal to zero.

When UQ's LabServer has completed running the experiment iLabs-Labconnector's Notify() web service call is invoked. This call is an asychronous call (derived from iLabs Service Broker), that allows the iLabs LabServer to identify that the experiment results are ready for retrieval. This allows the iLabs-LabConnector to retrieve the results (via UQ's Labserver RetrieveResults() call.) iLabs-LabConnector calls Sahara-LabConnector getExperimentResults(). Sahara-LabConnector then invokes the saveExperimentResults() web service call, which saves the results to file,

Similarly, when a user wants to cancel an experiment the releaseExperiment() is invoked that calls the iLabs LabServer cancel() web service method with the specified unique experimentID. A return value is returned back indicating whether the experiment has been cancelled via a boolean expression.

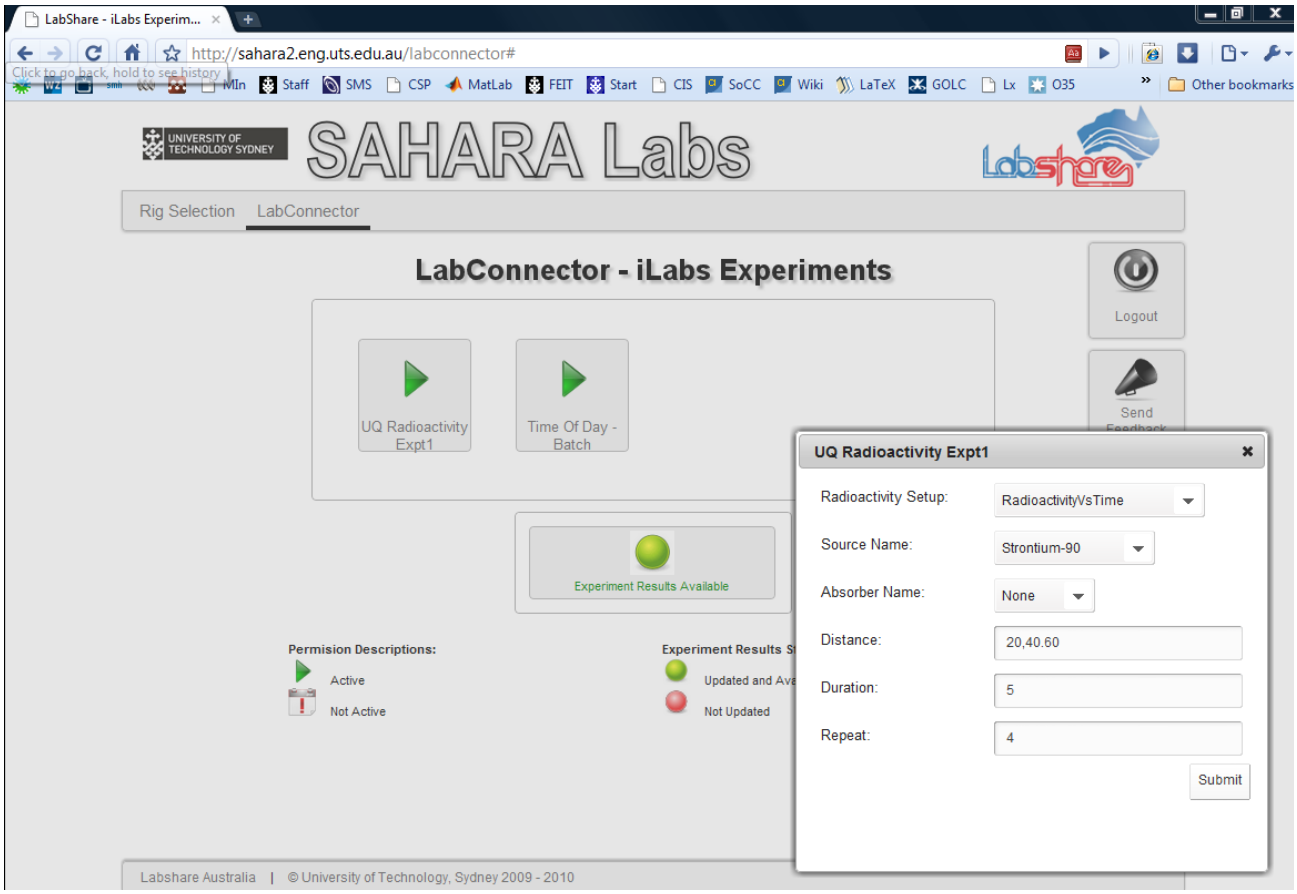

Figure 7. Screendump showing access to the University of Queensland Radioactivity iLabs experiment through a UTS Sahara system.

Whilst this demonstrates the feasible of cross-system interoperability, and provides the start of the definition for a translation protocol between systems, much further would be required before this could be established as a formal standard. In particular, we would need to explore interconnections with a number of other systems. Current plans for improving on the batch experiment implemention will be to implement validation support for the input fields based on the GetLabConfiguration() to provide better usability by specifying maximum and minimum values allowed by the input fields. This will mean the user will be informed if the input values are out of the allowed boundary conditions for the iLabs LabServer experiment. Also, plans will be to create a more comprehensive key-value schema which maps between the different implementations of userID's (Sahara uses a string format while iLabs uses an integer value) as well as mapping iLabs LabServer ID to Sahara's Rig name.

VII. FUTURE WORK AND CONCLUSIONS

Authentication and authorization still remains an issue with current remote lab designs. The ability to have single sign on (SSO) functionality will allow the use of such technologies to share labs more readily and assist with interoperability as there will be a defined means of user management. Currently, there exists authentication and authorization protocols that can be adopted in the implementation of remote labs. These protocols include Shibboleth and OAuth.

Another area which will garner interest in remote labs is the use of virtual environments [21] and game play in order to both simulate and stimulate the students' cognitive processes while reduce the burden of using physical labs remotely. Cataloging is another capability that will create a means of allowing more sharing between the different implementations of remote labs and hopefully encourage more inter-institutional participation. Cataloging is analogous to a directory listing and will allow a user to not only search and find experiments/labs pertinent to the subject they are interested in but will also allow for the access and running of the labs without any hindrance. Both iLabs and Sahara currently do not offer any such cataloging capability.

There has also been a growing interest in research and implementations of remote labs that are integrated with a learning management system (LMS), including Moodle based LMSs [18, 22, 23]. This capability does not currently exist in either iLabs or Sahara. This functionality will help with the coordination and management of lesson material and will allow the user to have a single access portal, assisting in usability.

Internationalization is also a growing concern where iLabs and Sahara both currently only support English (though iLabs has recently added support for Unicode). Providing online web services has given the ability for remote labs accessibility across institutions residing in varying geographical locations. As the implied nature of each different geographical region has separate language and possibly cultural nuances, the importance of internationalization becomes more of a concern. In later releases of LabConnector, we will investigate the lab capabilities of other systems more closely, to further remote lab research, including features such as cataloging and interoperability with LMSs.

ACKNOWLEDGMENT

The authors wish to thank Len Payne, Phil Long, Mark Schulz and John Zornig for providing access to the University of Queensland Radioactivity rig, and support in connecting the LabConnector bridge to the iLabs Lab server.

The authors also wish to acknowledge the generous support for this work provided by the Commonwealth of Australia's Department of Education, Employment and Workplace Relations, though the Diversity and Structural Adjustment Fund. We also wish to thank the project participants from the Australian Technology Network: UTS, Curtin, UniSA, RMIT, and QUT.

REFERENCES

[1] C. Gravier, *et al.*, "State of the art about remote laboratories paradigms-foundations of ongoing mutations," *iJOE*, vol. 4, p. 19, 2008.

[2] S. Murray, *et al.*, "Experiences with a Hybrid Architecture for Remote Laboratories," in *FiE 2008: The 38th Annual Frontiers in Education Conference*, Saratoga Springs, USA, 2008.

[3] V. Harward, *et al.*, "The iLab shared architecture: A Web Services infrastructure to build communities of Internet accessible laboratories," *Proceedings of the IEEE*, vol. 96, p. 931, 2008. doi:10.1109/JPROC.2008.921607

[4] D. Lowe, *et al.*, "Evolving Remote Laboratory Architectures to Leverage Emerging Internet Technologies," *IEEE Transactions on Learning Technologies*, vol. 2, pp. 289-294, 2009. doi:10.1109/TLT.2009.33

[5] V. L. Lasky and S. J. Murray, "Implementing viable remote laboratories using server virtualisation," in *Web-based Education*, Chamonix, France, 2007, pp. 68-72.

[6] S. J. Murray and V. L. Lasky, "A Remotely Accessible Embedded Systems Laboratory," in *Tools for Teaching Computer Networking and Hardware Concepts*, Sarkar, Ed., ed Hershey: Information Science Publishing, 2006, pp. 284-302.

[7] J. Hardison, *et al.*, "Deploying Interactive Remote Labs Using the iLab Shared Architecture," in *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, Saratoga Springs, NY, 2008, pp. S2A-1.

[8] L. Payne. (2009, Mar. 15, 2010). *iLab Batch LabServer Architecture Overview*. Available: http://ceit.uq.edu.au/wiki/images/9/9f/ILabBatchLabServerArchitectureOverview.pdf

[9] Y. Yuhong, *et al.*, "Putting labs online with Web services," *IT professional*, vol. 8, pp. 27-34, 2006. doi:10.1109/MITP.2006.45

[10] P. Charlton, *et al.*, "Dealing with interoperability for agent-based services," presented at the Proceedings of the fifth international conference on Autonomous agents, Montreal, Quebec, Canada, 2001.

[11] B. Haslhofer and W. Klas, "A survey of techniques for achieving metadata interoperability," *ACM Computing Surveys (CSUR)*, vol. 42, pp. 1-37, 2010. doi:10.1145/1667062.1667064

[12] S. Murray, *et al.*, "Experiences with a Hybrid Architecture for Remote Laboratories," presented at the FiE 2008: The 38th Annual Frontiers in Education Conference, Saratoga Springs, USA, 2008.

[13] E. Gamma, *Design patterns : elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley, 1995.

[14] A. Agrawal and S. Srivastava, "WebLab: A Generic Architecture for Remote Laboratories," in *Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on*, 2007, pp. 301-306.

[15] J. Garcia-Zubia, *et al.*, "Towards an extensible weblab architecture," in *E-Learning in Industrial Electronics, 2009. ICELIE '09. 3rd IEEE International Conference on*, 2009, pp. 115-120.

[16] C. Maier and M. Niederslatter, "Lab2go - A Repository to Locate Online Laboratories," *International Journal of Online Engineering (iJOE)*, vol. 6, pp. 12-17, 2010.

[17] R. Pastor, *et al.*, "Development of an XML-based lab for remote control experiments on a servo motor," *International Journal of Electrical Engineering Education*, vol. 42, pp. 173-184, 2005.

[18] M. Wulff, *et al.*, "Content management and architectural issues of a remote learning laboratory," in *Proceedings of the 2nd International Workshop on elearning and Virtual and Remote Laboratories*, Hasso-Plattner-Institut, Universitatsverlag Potsdam, Germany, 2008, pp. 13-19.

[19] A. Rasche, *et al.*, "Predictable interactive control of experiments in a service-based remote laboratory," presented at the Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments, Athens, Greece, 2008.

[20] T. Richter, *et al.*, "Lila: A european project on networked experiments," in *Proceedings of the Sixth International Conference on Remote Engineering and Virtual Instrumentation (REV 2009)*, Bridgeport, CT, USA, 2009.

[21] J. Ma and J. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Computing Surveys (CSUR)*, vol. 38, p. 7, 2006. doi:10.1145/1132960.1132961

[22] B. Ozdogru and N. E. Cagiltay, "How Content Management Problem of a Remote Laboratory System can be handled by integrating an open source learning management system? Problems and solutions," in *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, 2007, pp. 1-5.

[23] H. Benmohamed, *et al.*, "Generic framework for remote laboratory integration," in *Information Technology Based Higher Education and Training, 2005. ITHET 2005. 6th International Conference on*, 2005, pp. T2B/11-T2B/16.

AUTHORS

Herbert Yeung is with the University of Technology of Sydney, 15 Broadway Ultimo NSW Australia 2007 (e-mail: Herbert.Yeung@eng.uts.edu.au).

David Lowe is with the University of Technology of Sydney, 15 Broadway Ultimo NSW Australia 2007 (e-mail: David.Lowe@uts.edu.au).

Steve Murray, is with the University of Technology, Sydney. P.O. Box 123 Broadway, NSW 2007 Australia (e-mail: stevem@eng.uts.edu.au).

This article was modified from a presentation at the REV2010 Conference at KTH, Stockholm, Sweden in June 2010. Submitted July 15th, 2010. Published as resubmitted by the authors July 29th, 2010.