Design and Implementation Issues in a Contemporary Remote Laboratory Architecture

M. F. Schulz[1] andA. Rudd[2]
[1] Centre for Educational Innovation & Technology, UQ, Brisbane, Australia
[2] School of Information Technology & Electrical Engineering, UQ, Brisbane, Australia

***Abstract*—MIT has been developing the iLab Shared Architecture (ISA) for remote laboratories since 1998. It has been based around concepts and implementation issues that were in vogue at the time. Recent developments in network architectures and implementation techniques offer opportunities to re-examine the original assumptions, and to contemplate expanded objectives. This paper explores one possible future being explored at The University of Queensland for a remote laboratory architecture based upon the original ideals of MIT's ISA.**

***Index Terms*—remote laboratories, REST, event driven programming, real-time communication, platform independent implementation.**

I. INTRODUCTION

Many remote experiments have been built over the years of the Internet, but very few of these designs have been created with planet-wide (or even cross institutional) engagement at the core of their design. The MIT iLab Shared Architecture (ISA) is one such architecture which has this level of engagement encompassed from its very conception.

Researchers at The University of Queensland have constructed a number of remote experiments (in control engineering, power engineering, and physics) since 2006 which exploit the iLab architecture. As a consequence of this experience, additional considerations about the original architecture have evolved. This paper looks at the original design considerations of the MIT iLab Shared Architecture, the original implementation considerations, and then proposes a number of changes to both these aspects that are being considered in relation to contemporary implementations methods as well as extensions to the architecture that have arisen in the light of development experience.

II. ILAB BACKGROUND

Many individuals and institutions have developed remote laboratories of the past 40 years [1]. The aim of the developer often has been to instrument hardware to grant access to a user from a remote location. Frequently, the issues of distributed access control, management and allocation of lab resources, and data storage administration were not issues that attracted deep consideration. MIT iLab Shared Architecture was designed with these considerations at the forefront.

MIT iLab project was initially formed with the aim of defining a standard approach to the development of remote laboratories, and providing software tools to simplify the development of new experiments. This approach led to the development of the iLab Shared Architecture [2].

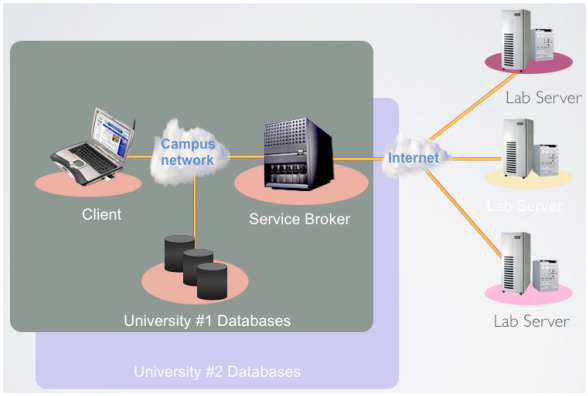

Figure 1. Topology of a batched experiment based on the iLab Shared Architecture.

In general terms, the iLab Shared Architecture divides any remote laboratory into three parts (refer to Figure 1):

- a *lab client*, which is the users lab specific interface to the experiment; and
- a *lab server*, which connects to the hardware and controls experiment execution; and
- a *service broker*, a middle ware layer that provides functionality that is common to all experiments - functionality such as user authentication and authorization, and data storage.

The ISA provides a framework which uses web services for a distributed deployment of experiments. Also, by placing services brokers in different institutions, the administration of users at each institution is handled at the service broker located at that institution.

Three different types of remote experiments to be supported by the ISA were initially identified:

- *batch* labs, where the experiment is completely specified prior to execution, and the experiments runs without intervention; and
- *interactive* labs, where the experiment or observations requires some sort of real-time interaction; and
- *sensor* labs, which run for extended periods of time and during which the experiment takes some snapshot of the entire period.

A feature of the batch lab is that the lab server and the client use web services to communicate entirely via the service broker. In fact, the experiment execution is progressed in three phases. First, the experiment

specification made on the lab client is validated at the service broker (and not transmitted to the lab server) to ensure that there will be no damage to the apparatus. Second, the validated experiment specification is submitted to the lab server for execution. Finally, the results are retrieved from the service broker and relayed back to the client. Each of these steps represent one premise upon which the iLab server broker has been built.

Interactive experiments require control of the lab hardware so that the user can set parameters and observe the results. Thus, the hardware requires dedicated access for a period of time (typically 15-20 minutes) and may require scheduling. Note also that because of the need for real-time control (and the potentially higher bandwidth requirements between the lab client and the lab server) the use of web services to route all communications between the lab client and the lab server will not work effectively in the case of this type of experiment (refer to Figure 2).

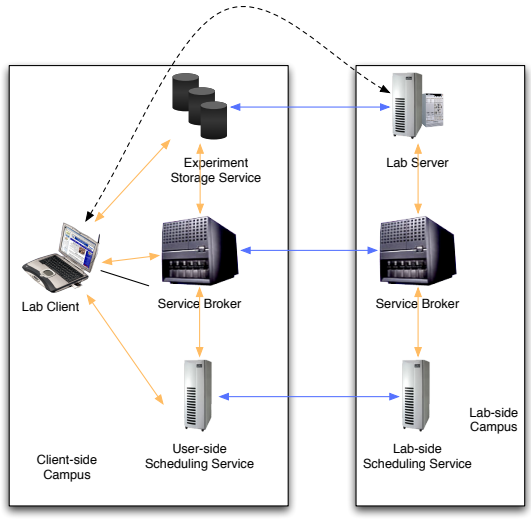

Figure 2. Topology of a interactive experiment based on the iLab Shared Architecture.

Another main difference between interactive and batched labs involves the role of the Service Broker. Interactive experiments require real-time control and, potentially, much greater bandwidth between the lab client and the lab server. Because of this, the batched notion of a Service Broker that uses web services to route all communications between the lab client and lab server will not work effectively in an interactive iLab.

Formal specifications for the sensor labs have not been completed at this stage.

III. iLAB WEB SERVICES

In the design of the service broker those tasks that were common to or desired by most labs were termed "domain independent". The five high-level mechanism that were deemed to be desirable for most Internet-accessible labs were:

1. An experiment-storage mechanism to store all specifications pertaining to an experiment type or run, and all results returned by an experiment.
2. An authentication/security mechanism to both establish the identity of the user and to set up a secure web-connection with the remote lab.
3. An authorization mechanism to specify the privileges on the lab server and database for each user of a remote lab.
4. A reservation mechanism to allocate time slots for experiments, on both the client side and on the lab server side.
5. An administrative mechanism to manage user subscriptions, accounts, and group memberships.

The infrastructure created performs two central operations:

1. facilitates the tasks that are common to Internet-accessible labs (referred to as the internal architecture), and
2. allows for the exchange of messages between this infrastructure and the lab components (referred to as the external architecture).

Web services were chosen to implement the external architecture, as this allows clients and servers to communicate and yet to be run on different architectures and platforms. The choice of the time was XML over SOAP to implement a remote procedural call. It was recognized that this choice restricted data transfers to be text-based and so sacrifice some data transfer speed, but it was felt that the interoperability advantage gained with a web service implementation far outweighed its disadvantages [3].

IV. CURRENT RESTRICTIONS

There exist a number of shortcoming in the existing ISA and in its implementation. We propose a number of extensions and enhancements which are detailed in this section. We look at implementation issues first.

The original implementation of the Service Broker was made using a Microsoft environment. The server is coded in Microsoft C#, .NET (and ASP.NET), Microsoft SQLserver. In fact, as we have discovered over time, the tool chain is highly dependent upon particular versions of each software package. In order to maintain backwards compatibility with those experiments developed at the time of the initial specification of the APIs, it has proved very difficult to move off this original code base. But it is the fragility of the tool chain that causes most concern with recently trained graduates in software engineering.

The choice of C# as the programming language has also restricted the operating systems upon which this software can be run. Any move to change the code to a new operating system is stymied as C# is only available on a Microsoft Windows platform; in fact, to Microsoft Windows 2003 Server Enterprise.

If we want to draw upon a large pool of developers to aid with the evolution of the iLab system we need to work with languages with large active developer bases. C# does not appear to fit this need.

It has been observed that problems occur when it is proposed to move the database from the Microsoft SQL Server to an open source SQL database such as MySQL or Postgres. Again, users are restricted to Microsoft SQL Server 2000 or 2005.

In order to work with the code base, users are restricted to Microsoft Visual Studio.NET 2005 Professional and Library. This means that many existing software tools are precluded from use.

Finally, there is no defined protocol for interactive labs to use to communicate between the lab client and the lab server. From one point of view this is an advantage, as this allows experiments to utilize existing communications

links with commercially available instruments where there is little or no access to the underlying protocol stack. Unfortunately, there are also very few packages available that enable users to develop their own interactive lab from scratch. Many engineers have resorted to using LabVIEW from National Instruments [4]. This software allows the creation of rich and powerful user interfaces in the lab client. However, the software is a commercial package and incurs a cost that some institutions may find prohibitive. LabVIEW is a graphical programming language that requires a reasonable period of familiarization. It use tends to be restricted to scientists and engineers, and is daunting for software developers to learn and retain. It would be useful if there was available a simpler and freely available toolkit to develop and tinker with user interfaces in a lab client.

In the development of lab clients, there is no preferred, defined or specified development environment. ASP.NET has been used extensively in the Service Broker. Clients have been written in a range of languages, but Java seems to have had a lead in this area. This is a run-anywhere language but it does require yet another language to be in the developers toolkit, along with the C# and ASDP.NET already mentioned. Also, the Java applets are dynamically loaded and it has been our experience that many of these applets tend to be large unless specific attention is paid to keeping the applet small. This often requires a redesign of the user interface to enable this to occur, e.g., in the redesign of the MIT Microelectronic [5].

In the case of interactive labs that have used LabVIEW, the machine where the client runs is required to have a LabVIEW application installed. This is difficult to achieve in some locations, and impossible when using publicly accessible machines outside the institution where the client is normally expected to be run.

At the time of the original development of the ISA, the choice of SOAP and remote procedural calls using XML would have been obvious. However, we have encountered problems caused by passing authorization information in the SOAP header and not in the payload. For example, one can consider that a remote lab is but a part of a larger experimental workflow. The workflow could be that the experimental parameters could be selected from a database subject to some selection criteria. This data is then passed to the remote lab for execution. Data is returned from the remote lab and then could be passed to a statistics package such as R which is also available as a service. This package may then be loaded with the data and a script which analyzes the data and displays outputs in the form of graphs and charts. Such workflow engines already exist, are open source, and are quite common in the area of bioinformatics and scientific analysis, for instance [6][7]. In its current form, an iLab cannot be incorporated into the workflow because of the need to manually log into the Service Broker directly. It would be more convenient if the workflow engine could request this information and have the user complete this process as a normal part of the data handling in the normal flow.

V. DESIGN DECISIONS

We will address several topics here: the choice of service architecture, the choice of programming language, the extension to real-time communication, and the need to develop a new collaboration model between multiple lab clients supported via intercommunication between the lab clients.

A. Browser-based Lab Clients

The first step in the design decisions was that to move lab clients to be fully browser-based, thus removing the requirement to utilized any installed software components, including Java interpreters. This choice means that lab clients must be written in HTML and CSS, and can also make use of JavaScript as all modern browsers now support this language. There is a rich set of community developed, freely available packages which permits the development of powerful applications running in the browser.

Having made this decision, we now have access to a much wider pool of developers than had previously been the case. One of the problems we have always faced is the design and development of appropriate GUIs in the lab client. We can now draw on a huge community of developers who can produce new designs, or modify and extend existing designs. Also, with the advent of the new graphical interface that we have seen with the work by Zornig and his team [8], we expect that the need for JavaScript programming will be significantly reduced. It is our hope that end-users will soon be able to customize the lab client GUI, especially teachers of school children.

The effect of this choice profoundly affects many of the other decisions we then make, as seen below.

B. Migration to a RESTful interafce

The concept of REpresentational State Transfer (REST) ful APIs has gained popularity as an architectural style between web apps and HTTP servers since the publication of the thesis by Fielding [9]. Using this approach greatly simplifies the design of servers and applications that call upon those services provided. Zornig [8] has already run some tests on using this architecture for the server with promising results, so this appears to be a straightforward decision.

Along with the development of RESTful architectures has been the move away from the data representation format of the eXtensible Markup Language (XML) to the JavaScript Object Notation (JSON) [10]. We find that support for JSON comes with the JavaScript engine in the browser, further simplifying the development of new software applications.

This choice of a RESTful architecture of itself does not have great impact on the original ISA design, as we could still pass the original XML (converted into JSON format for ease of processing) as the payload to and from the Uniform Resource Identifiers (URIs) of the Service Broker. However, the choice of programming language will have a profound effect on the eventual compatibility with the ISA architecture.

C. Web Service Programming Language Choice

The distributed ISA code base for the Service Broker and for the representative lab client is entirely based upon Microsoft products. This has proven problematic for some institutions around the world where the cost of this products is prohibitive. For some time there has been discussion about the move to a “run anywhere” platform that might be able to run under the common operating systems (Linux, MAC OS X, Microsoft). A prominent proposal has always to develop this code in Java.

However, the last few years has seen a push for the development of server-side JavaScript. This has the advantage that the server code is now open to all the

JavaScript developers that come from client-side development - a not insignificant population. The learning curve for code development is considerably reduced. Also, software development environments support JavaScript development and debugging are widely available, often as open source code.

Note that Javascript in the browser is event driven, and an event driven programming style is both resource efficient and has high performance in servers. This move from a remote procedure call model used in SOAP (basically a blocking paradigm) to an event driven paradigm is perhaps the most significant change. Interoperability with existing ISA software may be profoundly compromised. This aspect is the subject of ongoing research within our centre.

The Javascript server framework we have adopted (for the time being) is Node.js [11]. It has an extensive range of user-contributed support libraries that, at this stage, seem to meet the needs of the project.

D. Real-Time Communications Support

The existing ISA architecture does not supply and support real-time data communication. In the case of interactive experiments, this is outside the specifications. A reference interface has been developed to enable LabVIEW support of real-time interactivity, but this is not the only case of real-time communications that needs considering. We also have an interest in have the lab server and the Service Broker supply data detailing their operational status, data that needs to be supplied in real time.

At the moment the ISA architecture supplies two ways for a lab client to know the its execution status. The lab server can be polled continuously to ask about jobs in the execution queue, or the client interface can wait for a notification from the Service Broker that the job has completed and results are available for collection. This is large grained status.

We are interested in finer granularity of this result. Consider the case of two different status queues that have been requested by one developer in the lab client GUI for a batch experiment. First, a job queue which lists all the jobs pending execution, with information detailing their estimated execution time and thus the waiting time in the queue. Second, an execution queue which exposes exactly what is happening during the execution of the experiment. Both of these queues need to be updated in real-time to provide the required feedback to the lab client user. (refer to Figure 3.)

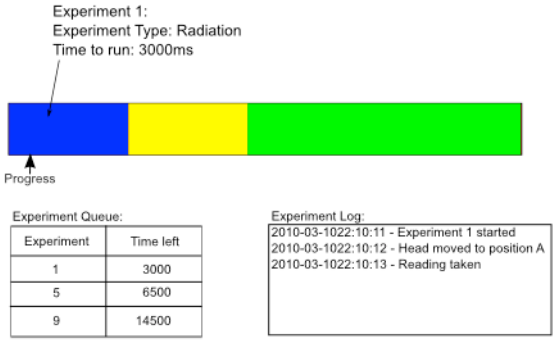

Figure 3. Views of a job queue and an execution queue displayed in a batch experiment GUI.

E. Lab Client Intercommunication

Almost every remote lab emphasizes the remote access to laboratory equipment as the primary outcome. However, it is becoming equally important to the learning outcomes of the students that they collaborate over the choices made before, during, and after the execution of the experiment. The DIESEL system [12] achieved collaboration by coordinating links to multiple remote desktops which link to a central experiment-controlling desktop. This is very expensive for bandwidth and does not scale well with the number of users.

We are investigating distributed collaboration in a more direct manner. Given that the JavaScript running in each lab client is event driven, it is worthwhile to consider if events generated in one lab client can be propagated to all other clients, i.e., an event generated in one Javascript-based client can be injected into other JavaScript-based clients.

To this end, we are examining the efficacy of using a topic-based publish/subscribe paradigm [13]. In this model, events that originate in one lab client are published to a topic on a message server. The message server then passes that message to all lab clients that are subscribed to that topic. This is an extremely efficient mechanism for distributing the events, it scales well as these message servers are optimized for message throughput, and there are a number of suitable server available in the open source community [14] [15].

Adopting real-time communication in a batch experiment will now break the original assumption that the lab client only communicates with the lab server via the Service Broker. This aspect will need further research as implementation of the Service Broker is pursued.

F. ISA Service Oriented Architecture Support

Implied in all of this is that the new architecture continue support for the same basic services identified in the initial ISA model specification. Work is in progress on an implementation of these services in the new architecture.

One model being actively researched is the design of a bridge between the existing MIT ISA and the new architecture. Given the disparity in the underlying models (RPC vs event driven), this work is progressing more slowly.

In this early phase of the project we are not implementing an underlying security model. We are following the normal practice of determining the new APIs before we look at the most appropriate security model.

VI. CONCLUSION

This paper outlines progress to date on a redesign of the the original MIT iLab Shared Architecture. Work has commenced on a batch lab version of the time-of-day reference design. The research team is work on a number of aspects simultaneously in a bid to rapidly progress a working pilot system. We will be looking to the remote lab community for feedback as work progresses.

ACKNOWLEDGMENT

The authors wish to thank the staff members and students who work at the Centre for Educational Innovation and Technology for their continuing conversations on this topic, in particular John Zorning who has provided the push for the use of Javascript in both the client and the server. John leads the work in the development of the new client framework for the proposed iLab architecture.

REFERENCES

[1] J. Ma and J. V. Nickerson, “Hands-on, simulated, and remote laboratories: A comparative literative review,” ACM Comput. Surv., vol. 38, no. 3, Sep. 2006.

[2] J. Harward,J. A. del Alamo, V. S. Choudary, K. DeLong, J. L. Hardison, *et al.*, “iLabs: A Scalable Architecture for Sharing Online Laboratories”, presented at the International Conference on Engineering Education 2004, Gainesville, Florida, October 16-21, 2004.

[3] K. Y. Yehia, “The iLab Service Broker: a Software Infrastructure Providing Common Services in Support of Internet Accessible Laboratories”, M. Sc. MIT, 2002.

[4] J. Travis and J. Kring, “LabVIEW for Everyone,” Upper Saddle River, NJ:Prentice-Hall, 2004.

[5] J. L. Hardison, D. Zych, J. A. del Alamo, V. J. Harward, S. R. Lerman, S. M. Wang, K. Yehia, and C. Varadharajan, “The microelectronics WebLab 6.0: An implementation using web services and the iLab shared architecture,” presented at the Int. Conf. Eng. Educ. Res. 2005, Tainan, Taiwan, R.O.C., Mar. 1–5, 2005.

[6] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, “Taverna: a tool for building and running workflows of services.,” *Nucleic Acids Research*, vol. 34, iss. Web Server issue, pp. 729-732, 2006

[7] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, “Taverna: lessons in creating a workflow environment for the life sciences,” *Concurrency and Computation: Practice and Experience*, vol. 18, iss. 10, pp. 1067-1100, 2006.

[8] J. Zornig, S. Chen, and O.Al Kylaney, “A REST API Architecture for Remote Labs.”, submitted to REV2011.

[9] R. Fielding, “Architectural Styles and a Design of Network-based Sofatware Architectures,” *PhD Thesis*, University of California, Irvine, USA, 2000.

[10] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” *IETF Request Form Comments: 4627*, July 2006. Available at http://tools.ietf.org/html/rfc4627.

[11] Mode.js, http://nodejs.org.

[12] M.J. Callaghan, J. Harkin, E. McColgan, T.M. McGinnity and L.P. Maguire, “Client–server architecture for collaborative remote experimentation,” Journal of Network and Computer Applications, Volume 30, Issue 4, Pages 1295-1308, November 2007.

[13] P. T. Eugster, P. A. Felber, R. Guerraoui, A. Kermarrec, “The Many Faces of Publish/Subscribe,” ACM Computing Surveys, Vol. 35, Issue 2, pp. 1-22, March 2003.

[14] Peter Millard, Peter Saint-Andre, Ralph Meijer, “XEP-0060: Publish-Subscribe,” XMPP Standards Foundation, 2010. Available at http://http://xmpp.org/extensions/xep-0060.html

[15] IBM, “MQ Telemetry Transport (MQTT) Version 3.1 Protocol Specification,” Available at http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

AUTHORS

M. F. Schulz is with The University of Queensland, where he is the Associate Director of the Centre for Educational Innovation and Technology, Brisbane, Australia (e-mail: m.schulz@ uq.edu.au).

A. Rudd is with the University of Queensland, where he is completing a Bachelor of Engineering (Software) in the School of Information Technology and Electrical Engineering (e-mail: adam.rudd@uqconnect.uq.edu.au).