JAVA Implementation of the Batched iLab Shared Architecture

L.J. Payne [1], M.F. Schulz [2]

[1] The University of Queensland, School of Information Technology and Electrical Engineering, Brisbane, Australia
[2] The University of Queensland, Centre for Educational Innovation & Technology, Brisbane, Australia

***Abstract*—The MIT iLab Shared Architecture is limited currently to running under Microsoft Windows. A JAVA implementation of the Batched iLab Shared Architecture has been developed that can be used on other operating systems and still interoperate with the existing Microsoft .NET web services of MIT's iLab ServiceBroker. The JAVA implementation provides a 3-tier code development model that allows code to be reused and to develop only the code that is specific to each experiment.**

***Index Terms*—Web Services, MIT iLab, Remote laboratories.**

I. Introduction

The iLab Shared Architecture (ISA) developed by MIT [1] uses the Microsoft .NET web services of the Microsoft Windows platform[2]. It also uses the Microsoft SQL database server for information storage by the ServiceBroker and LabServers. The Microsoft Visual Studio development tools are used to build the web applications for the ServiceBroker, LabClients and LabServers. By developing these web applications in JAVA [4] and using the PostgreSQL[6] database, it is now possible to extend the use of the iLab Shared Architecture beyond the Microsoft Windows platform.

JAVA provides the *jax-ws* framework for developing web service applications that interoperate with the Microsoft .NET web services. This allows a JAVA LabClient to communicate with a .NET ServiceBroker that in turn communicates with a JAVA LabServer.

JAVA is used to code the web service because the development tools and database software are free to download from the Internet, are free to use, and are available for a wide range of operating system platforms such as LINUX and Mac-OS as well as Microsoft Windows.

II. iLab Shared Architecture Model

A. Existing Model

The existing model for an MIT Batched experiment consists of two parts: a LabClient and a LabServer. The LabClient provides the interface through which the user creates and submits an experiment specification. The LabServer handles the validation and submission of an experiment specification from the LabClient (via the ServiceBroker) and runs the experiment on the equipment according to the experiment specification.

B. Revised Model

The model developed at the University of Queensland consists of three parts: LabClient, LabServer and LabEquipment [3]. The LabClient is the same as in the MIT model but the LabServer has been separated into two parts. Again, the LabServer handles the validation and submission of an experiment specification from the LabClient (via the ServiceBroker) but the LabEquipment separates the running of the experiment on the hardware from the LabServer.

Quite often the software used to drive the experiment hardware is very dependent on the computer platform, and in many cases is only available for the Microsoft Windows platform. So by separating out the LabEquipment from the LabServer, the LabServer can be developed in JAVA while the LabEquipment remains platform dependent.

As a result of the separation, the LabEquipment and LabServer no longer need to reside on the same computer. The LabEquipment can reside at a location suitable for running the experiment while the LabServer can reside on a system server, possibly along with the ServiceBroker and the LabClient.

An example of this occurs at the University of Queensland where the Radioactivity LabEquipment is located in the Physics department while the Radioactivity LabServer and LabClient reside on the School of Information Technology and Electrical Engineering server along with the UQ iLab ServiceBroker.

III. LabServer Web Service

The JAVA web service for the LabServer is generated from the WSDL file obtained from the MIT's .NET LabServer web service. An example implementation of a Batch LabServer was provided with the MIT's 6.1 version of the Batch ServiceBroker as the *Time-Of-Day* experiment. This implementation was originally used to obtain the WSDL file from the .NET web service. Since then, an abstract class of the .NET web service has been written for the LabServer to obtain the WSDL file.

A. SOAP Header

The ServiceBroker passes information in the SOAP header of the web service call to the LabServer. This includes the *Identifier* which is the ServiceBroker's GUID and the outgoing *PassKey* and both are used to determine the authenticity of the ServiceBroker making the request.

SOAP header processing is carried out in the message handler that is attached to the web service for the incoming requests. Since each request is independent of any other request, the information in the SOAP header has to be passed between the message handler and the web service application by means of the message context. The LabServer may receive two consecutive requests from two different ServiceBrokers meaning the information in the SOAP header will be different for each request.

The *Identifier* and *PassKey* are contained in an *AuthHeader* object that is extracted from the SOAP header and passed to the LabServer's web service through the message context for authentication. Should authentication fail, an exception is thrown back to the ServiceBroker denying access to the LabServer.

The LabServer uses an Enterprise Bean [5] to do the work of the web service. The web service simply processes the *AuthHeader* object information that it received through the message context before passing the request on to the bean to do the work.

B. Initialization

The first point of contact with the web service is its message handler. When a ServiceBroker sends a request to the LabServer, the message handler processes the request before passing the message to the LabServer's web service. This means that the first phase of initialization of the LabServer has to be carried out in the message handler and not in the web service. This is fine because a message context exists in the message handler allowing configuration information to be read from the *web.xml* file. This information includes the location of the configuration properties XML file that is read and an instance of a *ConfigProperties* object created.

But how does the LabServer's web service Enterprise Bean get to see the configuration information? The message handler places the newly created *ConfigProperties* object into a static variable in the LabServer's web service and sets an *initialized* Boolean flag. When the web service bean's constructor executes, it gets the *ConfigProperties* object from the static variable in the LabServer's web service and carries out the remainder of the initialization required by the LabServer.

Why can't the LabServer's web service bean get the configuration information from the *web.xml* file itself? A web service context does not exist outside of a web service call to enable that to occur.

IV. LabEquipment Web Service

The LabEquipment web service application is responsible for running an experiment on the equipment according to the specification provided to it by the LabServer. In certain cases, it also provides a mechanism for powering down the equipment after a period of inactivity. Generally, there is a burst of activity when experiments are submitted followed by long periods of inactivity. It makes sense then to power down the equipment during these periods of inactivity to reduce component wear as well as reducing overall power usage.

As mentioned earlier, the LabEquipment software is dependent on the computer platform used. If the LabEquipment only used the network to communicate with the equipment or carry out simulations then the LabEquipment could be developed in JAVA.

The JAVA web service reference for the LabEquipment is generated from the WSDL file obtained from the .NET LabEquipment web service abstract class.

A. SOAP Header

In a similar fashion to the LabServer web service, the LabServer passes information in the SOAP header of the web service call to the LabEquipment. This includes the *Identifier* which is the LabServer's GUID and the outgoing *PassKey* and both are used to determine the authenticity of the LabServer making the request.

SOAP header processing is carried out in the message handler that is attached to the web service for incoming requests. The *AuthHeader* object containing the *Identifier* and *PassKey* is extracted from the SOAP header and passed to the LabEquipment's web service through the message context for authentication.

The LabEquipment uses an Enterprise Bean to do the work of the web service. The web service simply processes the *AuthHeader* object information that it received through the message context before passing the request on to the bean to do the work.

B. Initialisation

Initialization of the LabEquipment web service occurs in the same way as the LabServer web service. The first phase of the initialization occurs in the message handler attached to the web service. The web service bean's constructor then carries out the remainder of the initialization required by the LabEquipment.

C. Web Methods

The LabEquipment service provides a number of web methods that can be called by the LabServer to run experiments on the LabEquipment. These include:

- GetLabEquipmentStatus – Determines the status of the LabEquipment and if it is offline, provides a status message.
- Validate – Takes an experiment specification and determines the estimated execution time. This time is dependent on the experiment specification and the type of equipment used.
- StartLabExecution - Takes an experiment specification and starts the experiment running on the equipment.
- GetLabExecutionStatus – Determines the status of the currently running experiment.
- GetLabExecutionResults – Retrieves the results of the experiment after it has finished running.
- CancelLabExecution – Cancels a currently running experiment.

These web methods do not depend on the type of experiment that is being run or the equipment used.

V. LabClient Web Application

The LabClient uses the JAVA ServerFaces framework to provide an interface for the user to submit experiments and retrieve results. A *Loader Script* is used by the ServiceBroker to launch the LabClient. The loader script passes the LabServer's GUID and the ServiceBroker's web service URL to the LabClient, by way of the URL request parameters. This allows the one deployment of the LabClient to be used by multiple ServiceBrokers and LabServers.

The JAVA web service reference is generated from the WSDL file obtained from the .NET ServiceBroker web service abstract class that includes only the web service methods for batch experiments.

The ServiceBroker generates a *CouponId* and *CouponPasskey* when the LabClient is launched and passes these to the LabClient also by the way of the URL request parameters. The *CouponId* and *CouponPasskey*

are then passed back to the ServiceBroker in the SOAP header with each web service call and used by the ServiceBroker to authenticate the LabClient.

The LabClients developed at the University of Queensland are considered to be an engineering approach. They do not provide any fancy graphical interface but only provide sufficient information to submit an experiment to the LabServer. For example, the LabClient for the Radioactivity experiment simply provides standard web page controls to specify the experiment setup, distance of the Geiger tube from the radioactive source and the duration of exposure to the radioactive source.

Northwestern University has developed an Adobe Flash LabClient [7] for use by high school students. It provides a graphical simulation of the Radioactivity experiment and then a step-by-step procedure for preparing, running and completing the experiment. The students are asked questions and are required to provide answers to those questions before continuing to the next step.

VI. Dummy ServiceBroker

A Dummy ServiceBroker has been developed to enable the development of the LabServer and its LabClient without the complexities of having to log into an iLab ServiceBroker. The Dummy ServiceBroker simply provides pass-through methods to allow the LabClient to communicate directly with the LabServer. Only one web method is not entirely pass-through and that is the *Submit* web method where an experiment number needs to be generated.

It is then possible, while debugging, to step through the code from the LabClient into the Dummy ServiceBroker then into the LabServer and LabEquipment and all the way back again to the LabClient.

The Dummy ServiceBroker can also communicate with more than one LabServer during development. This may be useful when one LabServer is being developed with the JAVA *jax-ws* framework and while another LabServer is being developed with the Microsoft .NET framework.

VII. 3-Tier Code Development

Development of the LabServer and LabEquipment web service applications occurs at three levels. The bottom level is the *Engine* which is a library containing the code common to all LabServer applications and similarly, another library containing the code common to all LabEquipment applications. For the LabServer, it contains the routines to access the database, the web service reference routines to call the LabEquipment and ServiceBroker web services, the base classes for processing experiment specifications and experiment results as well as the experiment engine threads that run the experiment drivers. For the LabEquipment, it contains the code that powers up and powers down the equipment and the equipment engine thread that runs the equipment drivers.

The next level up is the library containing the code that processes the experiment specification and experiment results for a specific experiment, for example, the Radioactivity experiment or the Time-Of-Day experiment. For the LabServer, this level also contains the drivers that execute the experiment setups for the specific experiments. For the LabEquipment, this level also contains the drivers that run the equipment.

The top level of the model is the web service application and its message handler. The code at this level cannot be placed in a library because it is the application that is deployed to the web server. The web service applications for each LabServer are almost identical. Similarly, the web service applications for each LabEquipment are almost identical.

Using this model for the LabServer and LabEquipment allows speedy creation of new applications by focusing on the development of experiment specific code at the second level and reusing the code of the other two levels.

VIII. Conclusion

The development of a JAVA implementation of the Batched iLab Shared Architecture has enabled platforms other than Microsoft Windows to host iLab experiments.

The use of the JAVA jax-ws framework has allowed the LabServer web service applications and LabClient web applications to interoperate with existing Microsoft .NET iLab ServiceBrokers.

By using the 3-tier code development approach, the time and effort required to create new iLab experiments is reduced.

Further Work

Development of the JAVA implementation of the iLab ServiceBroker and Experiment Storage Server (ESS) is currently underway to support Batched experiments. Work will then continue with the development of the User Scheduling Server (USS) and Lab Scheduling Server (LSS) to support the development of interactive experiments.

References

[1] J. Harward, et. al., "The iLab Shared Architecture: A web Services Infrastructure to Build Communities of Internet Accessible Laboratories", *Proceedings of the IEEE*, Vol96(6), pp. 931-950, June 2008.

[2] *iLab Downloads – iLabs Dev – MIT Wiki Service* https://wikis.mit.edu/confluence/display/ILAB2/iLab+Downloads

[3] *UQ-iLab-BatchLabServer-Java Repository* https://github.com/uqlpayne/UQ-iLab-BatchLabServer-Java

[4] *Java Platform (JDK) 7u7* http://www.oracle.com/technetwork/java/javase/downloads/

[5] *NetBeans IDE 7.2 + Glassfish Development Server 3.1.2* http://netbeans.org/downloads

[6] *PostgreSQL 9.1* http://www.postgresql.org/download/

[7] iLabCentral – The place to share remote online laboratories http://ilabcentral.org

L. J. Payne, is with the School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia

(Email: uqlpayne@uq.edu.au)

M. F. Schulz, is with the Centre for Educational Innovation & Technology, The University of Queensland, Brisbane, Australia

(Email: m.schulz@uq.edu.au).