iLab Interactive ~~Interactive~~ Services — Overview

Author: *Jud Harward, Ting Ting Mao, Imad Jabbour, Philip Bailey*
Last Revision: ~~*14 April 2006*~~*14 January 2010*

Introduction

This paper is intended to introduce the iLab ~~interactive~~ architecture to general readers and developers. The latter group will then need to read more detailed documentation. Readers who have some knowledge of the iLab batched architecture[1] will have an easier time grasping the interactive architecture, but knowledge of the batched is not a prerequisite for reading this document. The current reference implementation of the iLab Service Broker(ISB) supports both batch and interactive lab servers.

Related Documents:

There are three series of iLab ~~interactive~~ architecture documents. The *Overview* series provides a high-level introductory view of the architecture. It is aimed at the general technical reader, but it also provides a starting point for developers encountering the architecture for the first time. This document is the first of the Overview series. The *Operation* series describes individual web services and applications within the iLab ~~nteractive~~ architecture at a level of detail that developers will require. The *Specification* series presents the application programming interfaces and data type descriptions for each of the ~~interactive~~ web services. It is intended to serve as a programming reference for developers and is derived from comments embedded in the source code.

1. *iLab ~~Interactive~~ Services — Overview (This document)*
2. *iLab ~~Interactive~~ Ticketing and Integrated Management — Operation*
3. *iLab ~~Interactive~~ Ticketing and Integrated Management — Specification*
4. *iLab Experiment Storage Service — Operation*
5. *iLab Experiment Storage Service — Specification*
6. *iLab ~~Interactive~~ Scheduling — Operation*
7.
8. *iLab Scheduling Services — Specification*
   ~~iLab Interactive Scheduling Services — Specification~~

A Comparison of the Batched and Interactive Architectures

Executing a Batched Experiment

In the batched architecture (revisions 6.1 and lower), a user begins a session by using a standard web browser to log into her account on a particular instance on the iLab ~~Batched~~ Service Broker (ISB). Once the student has authenticated herself, the Service Broker ~~(SB)~~ can look up the groups in which the user is enrolled. Groups usually correspond to academic classes and the user's role in the class, e.g., student, teaching fellow, system

---

[1] See, for instance, V. Judson Harward et al., iLab: A Scalable Architecture for Sharing Online Experiments, ICEE 2004, available at http://icampus.mit.edu/iLabs/architecture/downloads-/downloadFile.aspx?id=3.

manager, etc. The user is asked to select one of the groups in which she is a member to be the effective group for the current session. The ~~SB~~ ISB determines which experiments can be executed by members of that effective group. The user can then ask the ~~SB~~ ISB to launch one of the experiment clients to which her group has access. Up until this point the user has been executing a web application hosted on the ~~SB~~ ISB. Once the client is launched, however, the user interacts with the client, which then communicates with the ~~SB~~ ISB via web services. Experiment clients can be implemented in several technologies, e.g., a Java applet, an ASP.NET application.

The essence of a batched experiment is that once it starts execution, the user can not interact with it. The experiment client can be thought of as an experiment editor. The user specifies all the parameters that will define the experiment by interacting with the client before the experiment starts. The client then submits the experiment to the ~~SB~~ ISB, which saves a copy as part of the experiment record, before relaying the experiment specification to the lab server. The lab server executes the experiment immediately if the experiment apparatus is available; otherwise, it enqueues the experiment specification. The client can poll the ~~SB~~ ISB, which, in turn, can poll the lab server to determine whether the experiment has completed. Once the lab server has executed the experiment, it makes a web service call to the Service Broker to notify it that the experiment has finished. The Service Broker then retrieves the results from the lab server and saves them along with any error messages as part of the experiment record. The client can then pull the results from the Service Broker. The client doesn't even need to remain online while the experiment executes. It can come back online at a later time to check on the progress of the experiment and retrieve the results.

Consequences of the Batched Architecture

Two aspects of the batched design greatly simplify the architecture. The first of these is that the client never communicates directly with the lab server. All information passes through the Service Broker. Each Service Broker may support multiple batch lab servers, but each lab server has a direct relationship with the Service Broker. Batch Lab Servers may establish relationships with multiple Service Brokers.

Figure :The topology of the Batched Experiment Architecture

This allows the Service Broker to store the authoritative record of the experiment. It first saves the experiment specification submitted by the client before it forwards it along to the lab server for execution. Later once the lab server notifies the Service Broker that the experiment has finished, it retrieves the results of the experiment and saves the results before returning them to the client. Thus, there is no need to expose an independent web service to store the log of an experiment.

The interposition of the ~~SB~~ ServiceBroker between the client and lab server also simplifies authentication and authorization of users. The user authenticates himself to the ~~SB~~ SB, which then uses the standard web session technique of a session cookie to remember the identity of the user. The batch lab server permanently recognizes the ~~SB~~ SB because the ~~SB~~ SB always sends a secret token in the header of web service requests.

These requests are sent over SSL in order to protect the security of the ~~SB~~ SB's token. In the batch architecture revisions 6.1 and lower, t~~T~~he lab server administrators must initially convey the token out of band to their counterparts overseeing the ~~SB~~ SB. The transient client process has no need to authenticate itself to the lab server. In fact the lab server usually does not know the identity of the user, only the user's level of access as indicated by the effective group for the current session. In effect, the ~~SB~~ SB authenticates and then vouches for the user. The lab server has no need to maintain independent user accounts.

Finally the central role of the ~~SB~~ SB allows the user to log off while her experiment is waiting in a queue. Because the iLab architecture allows users and lab servers to be located at separate locations, potentially continents apart, the ~~SB~~ SB's automated patience can compensate for slow and unreliable internet connections.

The second simplifying aspect of the batched architecture is that the user does not interact with the experiment while it is running. Thus the execution of the experiment on the lab server is not limited by human reaction time. Many users can simultaneously prepare experiments to be run on a single lab server because the creation of the experiment specification is carried out as a separate process that usually runs on the user's own machine. As an example, a user executing a semiconductor characterization experiment at MIT (Microelectronics WebLab) may take 20 minutes to prepare the experiment specification on his client, but the experiment itself typically executes in 15-20 seconds on the lab server. If the student was actually setting up the experiment through the front panel of the semiconductor test equipment, the setup would require at least 20 minutes (because of the less intuitive front panel design). Thus, experiment throughput is potentially several orders of magnitude higher using the batched architecture than it would be using the test equipment directly. When necessary, the lab server can queue experiments submitted in close succession. As a consequence, users have no need to schedule their time on the lab equipment. They simply submit experiments on a first come, first served basis.

The Network Topology of the Interactive Architecture

At the start of the design of the interactive architecture, we had to decide whether all communication between the client and lab server should be mediated by the Interactive Service Broker (I~~ISB~~SB) as in the batched architecture. Routing all communication through the I~~ISB~~SB would allow it to save an authoritative log of the user's control of the experiment and the corresponding results. It would also simplify authentication and authorization. On the other hand, it would increase network latency between the interactive client and the lab server. Every control message from the client to the lab server and every status or result message from the lab server to the client would require two network hops instead of one.

The lab development community strongly urged the case for allowing direct communication between client and lab for a second reason. Interposing the ~~ISB~~SB would restrict lab and client developers to using an iLab defined protocol for passing control and result information. Many labs today are computer controlled even if remote access is not a requirement. Lab developers often create a virtual interface to the lab that runs on a

second system using virtual instrumentation packages like National Instruments LabVIEW~~iew~~. This (user) client and (lab) server system is often built before the decision is made to move the system to the iLab architecture. Allowing direct communication between user client and lab server gives the developers the freedom to choose their own communication protocol and to use third party packages like ~~LabView~~ LabVIEW and MatLab in their development. Thus the iLab team decided to allow direct communication although it was going to introduce added complexity to the interactive design.

Interactive Services and Applications

Creating the Interactive Experiment Log

If the interactive client and lab server can communicate directly, the ~~ISB~~SB can no longer be responsible for creating the definitive record of the experiment. In fact, all three processes, the ~~ISB~~SB, the client, and the lab server, may need to store information to record the complete experiment. This implies that the interactive architecture requires that an independent Experiment Storage Service (ESS) be exposed as a web service. The ESS stores both XML and binary data about a particular experiment, but it does not store administrative information about the experiment, e.g., the owner of the experiment log, the class the experiment was executed for, etc. The administrative information is managed by the ~~ISB~~SB. Thus, any request to access an experiment log on the ESS must be authorized by the ~~ISB~~SB since it is only the ~~ISB~~SB that can relate users to their experiments.

More detailed information on the ESS can be found in the following associated documents:

1. *iLab Experiment Storage Service — Operation*
2. *iLab Experiment Storage Service — Specification*

Scheduling

The interactive architecture permits students to observe the progress of the experiment and to interact with the experiment in ways that can change the experiment's course. Such labs typically require more time to execute than batched experiments because they proceed in human not machine time. A typical interactive experiment requires 20 minutes to several hours to execute. Because users control the lab equipment, they usually require exclusive access to it. Asking users to queue for their turn to use the lab wastes their time.

Hence most interactive experiments require a scheduling application that allows the users to sign up in advance for time on a particular piece of lab equipment. Access to this scheduling application must be authorized by the I~~ISB~~SB since only the I~~ISB~~SB can authenticate a user and vouch for his identity. The scheduling application must also notify users if their reservation must be cancelled or changed. Finally certain labs have operating requirements that require actions either before or after the execution of an experiment. For instance, a heat exchanger experiment may require the apparatus to achieve thermal equilibrium before the start of an experiment. A chemical diffusion experiment may require that the diffusion apparatus be flushed at the end of the

experiment. The scheduling application must allocate time for these actions in scheduling experiment sessions.

Scheduling can be looked at from two perspectives. From the lab provider's perspective, the scheduling application coordinates reservations to use a lab from multiple campuses. The scheduling server is also the process that holds the information required to "wake up" a lab server to perform required actions before a scheduled experiment. The lab provider may want to allocate blocks or percentages of time to different groups of users according to certain policies. For instance, Lab A may want to permit use from universities B and C between 6 pm and midnight, Monday through Friday, with university B allocated 60% of the time and university C the remainder. On the other hand, the lab provider generally does not want to be aware of the details of a user's reservation. If the lab server must be taken down for maintenance, the lab provider would simply like to notify the scheduling application of the down time and have the scheduling application take care of informing the affected users and rescheduling their work.

From a teacher's and a student's perspective, the scheduling application must act as their agent in scheduling time on lab servers. The application must accept authorizations to schedule from the users' ~~ISB~~SB and must record reservations in a way that can be associated with individual users. If a reservation must be cancelled, the scheduling application must take the responsibility for informing the user. Teachers may want to stipulate policies that govern how their students may make reservations. For instance, a teacher may decide that students can only sign up for two hours of lab access per week with no single reservation lasting more than one hour. Different teachers using the same lab may want to set different policies for their students.

Given the different requirements from the lab-side and the student-side perspectives, where should the scheduling application be located? The need to coordinate reservations from multiple campuses for a single lab server argues that there should be a single scheduling application located close in network terms to the lab server. But the requirement to accommodate the different policies of individual teachers suggests the need for multiple scheduling applications, at least one on each student campus like batched service brokers. We have decided that the two perspectives require two related scheduling applications, a Lab-Side Scheduling Server (LSS) and a User-Side Scheduling Server (USS).

The two scheduling applications communicate using a very simple and restricted web service protocol. All the intelligence and complexity is housed in the two applications. Thus, while the USS will at first support a very simple set of user scheduling policies, say first-come, first served, limited by a maximum reservation allowance and a maximum reservation length, we expect new policies will be added as the system matures. Likewise, although the initial version of the LSS will be quite simple, it too will gain more complex lab-side policies over time. As an example, we would like to add a reservation policy where user reservations are prioritized by the lead time with which the users can make reservations. High priority users can reserve time a longer period in the future than low priority users, who in effect get "rush seats" for the lab. Decoupling the

LSS and USS allows separate development. A university that wants to implement an innovative USS policy can do so without touching the host's LSS.

More detailed information on scheduling can be found in the following associated documents:

1. *iLab Interactive Scheduling — Operation*
2. *iLab Interactive Scheduling Services — Specification*

Authentication and Authorization

To begin either an administrative or an experimental session with the iLab interactive architecture, the user must authenticate himself to the ISB. The reference implementation supplies a simple user name and password scheme carried out using a standard browser-based web application. The architecture permits other authentication mechanisms, e.g., authentication by certificate, to be added to the implementation.

Once the ISB knows the identity of the user, it supplies authorizations for actions that the user wants to perform on other distributed applications and servers. Two examples follow:

1. After authentication, the user may indicate that he wishes to schedule a future lab session and chooses one of the labs to which he has access. The ISB would then redirect him to the web application of the USS that handles the reservations for that lab. The redirection must be accompanied by credentials sufficient to identify the user and to convince the USS to allow him to schedule a future experiment session.
2. When the time has come for the student to execute the experiment, the ISB must launch the client with credentials that the lab server will recognize. The client will usually try to contact the lab server directly, and the lab server should only accept the connection if it trusts the credentials originally furnished by the ISB. These credentials will usually include the period the user has reserved and the group (or class) for which the experiment is being executed. Different groups may be allocated different levels of access. A graduate class may be able to perform more sophisticated functions than an introductory class.

This latter case introduces an additional requirement because the lab server will probably need to use the same credentials to invoke services on behalf of the user. When the lab server needs to store experiment data, it must contact the user's ESS and present the forwarded credentials that will allow the ESS to recognize who owns the data that is being stored.

The interactive architecture will eventually use the emerging standard known as WS-Security to manage authorization and credentials. Until this standard has stable cross-vendor implementations, credential management will be managed by an iLab-specific mechanism called *general ticketing*. This technology is further described in the following documents:

1. *iLab Interactive Ticketing and Remote Management — Overview*
2. *iLab Interactive Ticketing and Remote Management — Operation*
3. *iLab Interactive Ticketing and Remote Management — Specification*

Remote Management

Over the past five~~two~~ and a half years the iLab team has managed batched service brokers at MIT and assisted other universities in installing their own service brokers that could access MIT labs. Despite the relative simplicity of the batched architecture, it has proved difficult to configure and maintain multiple service brokers and lab servers. Each such system possesses its own administrative interface, and staff must log in to a privileged account on each system and manually enter information in order to configure it. Our experience has convinced us that the far more complex interactive architecture will have to be centrally managed to the greatest degree possible.

Typically a campus will possess one or more service brokers and associated iLab servers. Each service broker forms a *domain*, that is, a set of applications and servers whose resources are managed by tickets issued by that domain's service broker. An interactive lab server will execute experiments for users from multiple domains, but it will always possess a home domain managed by a service broker. The same principle applies for other iLab servers and applications like the ESS, USS, and LSS. The~~:~~ interactive services are collectively known as Process Agents, and implement a small set of Web Service methods that standardize the exchange of data between the services.

In order to simplify the management of intra-domain and inter-domain services, I~~ISB~~SBs will provide administrative groups for each of the associated servers. Administrators for each iLab server will login to the domain's I~~ISB~~SB to manage their own server and configure access to it. There may be a separate administrative interface that runs on the server itself, and the server administrators always have the option of directly configuring their own server. Best practices, however, advise that all iLab configuration be performed through the interface on the domain's interactive service broker.

An example will clarify the division of responsibilities. The owners of a particular interactive lab, IL1, may want to share it with students from multiple universities. Reservations to use the lab must be confirmed through a lab-side scheduling server (LSS1) in the same domain as the lab server. This domain is managed by an interactive service broker, I~~ISB~~SB1. Let us say that the IL1 owners want students from U1 university to have permission to use IL1 from 9 am until 5 pm every week day, and students from U2 university to have permission to use IL1 from 6 pm until midnight every evening.

An IL1 administrator, Mary, must first grant access to students from U1 and U2 to access IL1. To do this, Mary will login in to I~~ISB~~SB1 as a member of the IL1 administrators group and will grant access to groups of students from U1 and U2 on an I~~ISB~~SB1 active server page. Once this is done, Mary must also set lab-side scheduling policy on LSS1, but she can only gain access to the policy page on LSS1 dedicated to LS1 through the service broker I~~ISB~~SB1. That is, from the same IL1 privileged login, Mary must request access to IL1's scheduling policy page on LSS1. I~~ISB~~SB1 will redirect her to LSS1 with credentials that tell LSS1 that Mary has permission to specify scheduling policy only for LS1, but not for any other lab server.

One advantage of centralized management is that it will allow lab providers to use standard reusable code to perform most of the tasks involved in managing their systems. Lab specific management will require custom programming, but it can still take advantage of iLab credential management.

For example, Mary may need to monitor the level of consumable reagents used by the lab. She would carry out this lab-specific operation through a custom page of the IL1 administrative interface. We recommend that she gain access to this page using the same credential system that gave her access to LSS1 rather than through a server specific login.

The remote management system is further described in the following three documents:

1. *iLab Interactive Ticketing and Remote Management — Overview*
2. *iLab Interactive Ticketing and Remote Management — Operation*
3. *iLab Interactive Ticketing and Remote Management — Specification*