# The Challenge of Building Internet Accessible Labs

## *Introduction*

Microsoft has funded, through the iCampus Project at MIT, a three year effort to explore the potential educational value of Internet accessible laboratories (*The iLab Project*). In the past year, the iLab Project has moved from a wide sampling of potential labs and experiments to a more focused effort to develop a shared software infrastructure. This white paper lays out the thinking behind this new architectural approach and outlines progress to date as well as plans for the future.

## *Goals*

We start with the fundamental assumption that we want to make labs accessible to *students*. If that also makes them more accessible to faculty colleagues or industrial partners, so much the better. Our principal aim in the iLab project, however, has been to integrate Internet accessible labs as smoothly as possible into students' educational experience with everything that implies about context and the scalability of the endeavor.

This basic priority leads fairly directly to three goals that affect our design:

1. While students are the intended beneficiaries, the people who need the help are the faculty and lab staff who must set up and interface the labs to the Internet. Our most important goal is to provide a software infrastructure that will make it easier and cheaper to make a lab Internet accessible. To the greatest degree possible, we want to do so without making any assumptions about the nature of the lab itself, the pedagogy used by the faculty, or the academic policies of the educational institution using the lab.

   An ideal endpoint for the project would be the participation of major lab equipment and software vendors (e.g., Agilent, National Instruments) to provide *Internet-ready* equipment and software. In fact, National Instruments already provides a LabView web browser client and server-side software to connect to it. These modules, however, are better adapted for factory automation behind a firewall than the open Internet. We must reach beyond a purely academic environment to engage the active cooperation of industry, and we should focus on the part of the problem that we understand: how to use experiments to reinforce pedagogy and how to integrate online labs with students' work habits. We should avoid duplicating or competing with industry efforts to create technology that they offer to academic institutions at deep discounts.

2. We should plan for an environment that can host or offer multiple online labs semester after semester. The software should not increase the administrative load of either the course or lab staff. Simply put, Internet access opens lab equipment to a wide audience. We must make sure that the software *scales*.

There are also clear advantages to predictable interfaces and administrative procedures. If a student uses more than one Internet accessible lab, she should find using the second more natural after using the first. Domain specific aspects of lab equipment and procedures may differ, but the methods a student uses to reserve or connect to a lab, to store or review results, to generate hard copy, etc, should remain as uniform as possible.

3. The concept of Internet accessible labs encourages cross-institution cooperation. One can easily imagine students at one university using a laboratory made accessible by a second university. Schools or universities may decide to share the cost of an expensive laboratory and physically establish it at a convenient location. One can also imagine government participation that would offer limited access to national laboratories or facilities like the International Space Station. In time, as online labs proliferate, we may require a discovery process by which a faculty member (or a student) can locate an online lab that offers a particular experiment or technology.

   These potential uses require that the software architecture separates lab users from lab providers. It also suggests that the architecture should support priorities of use, and eventually distributed resource accounting. We believe the ideal scheme would be one in which laboratory staff could determine what proportion of lab time would be devoted to each category of use but delegate the mechanisms of access by category and institution to servers and policy controlled by the institutions providing the lab users (students). For example, a particular MIT laboratory might decide to offer 10% of all access to students at Stanford and 20% of night access to a consortium of universities funded under an NSF initiative. But MIT should not manage or even be cognizant of how Stanford or the NSF Consortium were allotting their proportional access to individual students.

## *Where's the Leverage?*

At this point it is reasonable to ask, given the immense variety of laboratory experiments and equipment, whether they share sufficient characteristics so that they can be supported by a common software infrastructure. How can we begin to organize the near infinite variety of ad hoc laboratory configurations and procedures so we can model them and match them to a software strategy? The fact that hardware vendors now provide low level control of lab devices via industry standard protocols such as GPIB doesn't address the issue of a high-level software architecture. Packages like National Instruments' *LabView* and its associated lower level libraries provide visualization and interface support, but they do not integrate with the types of enterprise scale software that universities are using to manage courses.

We believe the leverage lies in providing general support for the framing and maintenance of a lab session while leaving details of the lab interface and control to domain specific experts or to vendors like National Instruments. That is, we want to distinguish the parts of using an online lab that are specific to a particular piece of lab equipment or to the specific series of tasks that comprise a particular experiment from the generic tasks that precede, manage, and follow any lab session.

The student will usually need to authenticate himself to the lab software. The student's online identity may well govern the labs that are made available through subsequent menus. For experiments of some duration, the student may need to have previously reserved the online lab. Results will need to be stored, analyzed, compared, or printed. The system may encourage collaborative work using standard application sharing and communication tools. The system should allow the student to forward a log of lab activity to a staff member to enlist their help with a problem. These capabilities are generic and transcend the individual experiment. Not all online experiments will require all of them, but most online experiments will require some of them.

## *A Typology of Internet Accessible Labs*

Our preliminary work based on the initial online labs available at MIT suggests that a single shared architecture is too limiting. We have found it necessary to distinguish three types of laboratory experiments. Later experience may require even finer distinctions, but already we are convinced that these three categories of experiments possess different types of lab sessions and, hence, require a different software architecture.

### The Batched Experiment:

In a *batched experiment*, the student specifies all parameters that govern the execution of the experiment before the experiment starts. The lab session consists of submitting an experiment protocol, executing the experiment, and then retrieving and analyzing the results. Typically, batched experiments run quickly so that scheduling is rarely necessary. The MIT Microelectronics WebLab (weblab.mit.edu) provides an excellent example.

### The Sensor Experiment:

In a sensor experiment, the student usually can not specify any parameters although she may be able to select the particular sensor data that she wishes to receive. Running the experiment consists of subscribing to real time sensor data, usually presented in a graphical user interface such as a virtual strip chart. The system may provide options to filter the data or to transform it as well as to access archival data. Long running sensor subscriptions may benefit from implementing trigger or alarm mechanisms. Imagine an online seismometer that notifies a student of a seismic event through email or instant messaging. The detection of a seismic event that passed a specified threshold might trigger more frequent sampling or complementary data presentations.

Sensor experiments frequently have very asymmetric data flows. It takes few bits to subscribe to a sensor, but the resulting data stream from the sensor to the student's client may require a great deal of bandwidth. Some sensors may only provide best efforts to deliver continuous data with no guarantee that all samples will arrive. Other systems may provide archival quality data but perhaps with a variable lag time.

This category of experiment is currently well represented at MIT by a flagpole instrumented with accelerometers that stream continuous data (flagpole.mit.edu).

### The Interactive Experiment:

In an interactive experiment, the student typically sets a series of parameters, initiates the experiment, and then monitors the experiment's course, changing control parameters as necessary. Conceptually, an interactive experiment can be thought of as a sequence of alternating control and monitoring intervals. In general, the control intervals have many of the characteristics of a batched experiment, and the monitoring intervals resemble sensor experiments. The record of an experiment session typically includes both time-stamped control and sensor data as well as other forms of documentation that may include images or video. The Internet accessible heat exchanger at MIT (heatex.mit.edu) provides a good example of this type of experiment.

## *Are Web Services the Correct Infrastructure?*

In theory, the software framework and the network technologies we use to build our architecture are implementation issues. With the rise of middleware and distributed application frameworks, however, one's choice of a distributed computing strategy can symbiotically affect the whole design of a project.

There are design requirements in the iLab Project that immediately favor the use of web services. Students on one campus must be able to use a lab housed on a second campus. This requires an architecture that supports both lab-side services (e.g., the online lab itself, a reservation service for the lab) and client or student-side services (authentication and authorization, class management, student data storage for experiment specifications and results as well as user preferences). The lab side services may need to run on a different hardware and software platform than the client-side student software. The lab-side campus may enforce different networking policies (e.g., firewalls, directory and email services) than the client-side campus. The transparency of web services makes this technology an obvious choice to integrate our distributed application framework.

Very often existing labs have a large preexisting code-base to manage the lab equipment or to display results and control the lab equipment from one or more client machines. The loose coupling of web services makes it easier to reuse this legacy codebase as the basis of the Internet accessible implementation. It will also make it much easier to incorporate vendor supplied modules in the overall architecture.

When we turn to consider the future course of the project, the case for web services only becomes stronger. We have already mentioned that we expect the use of Internet accessible labs will foster increased cooperation between educational institutions. Some schools and colleges may be much more interested in becoming clients of such labs than in offering online labs themselves. They will need a means of discovering what online labs are available in their area of interest and of verifying that those labs are compatible both with their campus networking and software install base as well as the pedagogic goals of their courses. Web Services WSDL and UDDI provide a framework in which to conduct this discovery process. To carry this one step further, one can imagine WSDL-based negotiation that will match an Internet accessible lab with high end visualization and data analysis tools that are licensed by the client-side campus.

For all the reasons above, we have based our shared software infrastructure for Internet accessible labs on web services. But it is also important to recognize that web services

will not solve all the networking requirements of Internet accessible labs. Consider an online sensor lab that wants to multicast high bandwidth sensor data to subscribed clients. While web services may aid the subscription process, they don't provide any support for streaming the multicast data to the students' clients.

## *The Current MIT iLab Architecture*

Our design of a shared software infrastructure did not start from scratch. The MIT online labs already mentioned all had working implementations up on the web at the start of the 2002/2003 academic year. Although these implementations did not share a single line of code, they informed our approach by the problems they had encountered and the strategies they had evolved to solve them. One of those labs, the Microelectronics WebLab, a batched experiment laboratory, became the focus of our early design because of the sophistication of their current implementation and their willingness to devote programming resources to moving their implementation onto the new architecture.

We also had the benefit of examining and building on the work of Dave Mitchell and Eric Carlson, Microsoft engineers visiting at MIT. In 2001/2, they had led a student staffed project to build a web service prototype architecture for Internet accessible labs. This project, called the Framework Project in its first incarnation and the Edge in its more developed form, was based on a series of seven small independent services (Identity, Authorization, Reservation, Notification, Storage, Status, and Event) and a unifying Lab Service that invoked them while providing an interface to the underlying online lab. After review and some experiment, we decided that the Framework Project approach had two main drawbacks. The first was that it was in some ways too general. It took a completely agnostic approach to the location of the seven small web services. It did not distinguish between client-side and lab-side services. The second objection was that these services were too fine-grained. The overhead of marshalling and unmarshalling the SOAP calls from the Lab Service to these independent services placed an unacceptable load on the Lab Service.

Our goal in this first phase of developing a shared architecture has been to support batched experiments that cross campus boundaries. Instead of employing a number of fine-grained services we have designed an architecture that more closely resembles a three-tier web architecture.

1. The first tier is the student's client application that either runs as an applet or as a downloaded application on the student's workstation.
2. The middle tier, called the Service Broker, will usually reside on a server on the student's campus. It will be backed by a standard relational database such as SQL Server or Oracle. The student's client communicates solely with the Service Broker, which forwards experiment specifications to the final third tier.
3. This third tier is the Lab Server itself, which executes the specified experiments and notifies the Service Broker when the results are ready to be retrieved.

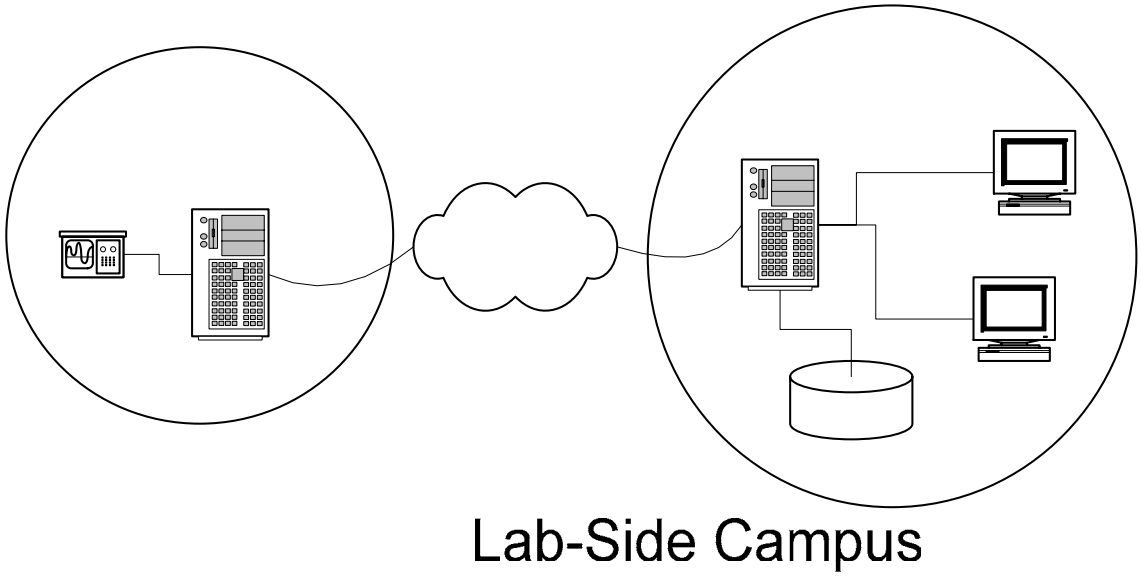


**Figure 1**

In this scheme, the student client and the lab server represent the domain- or lab-dependent software modules. We intend the Service Broker to be completely generic code. That is, we expect staff to be able to configure a fresh version of the Service Broker straight from the software distribution to register and cooperate with any Lab Server that implements the appropriate Lab Server interface expressed in terms of Web Service SOAP calls defined in WSDL. The student's client is once again a domain- or lab-dependent piece of code that must understand the protocol for specifying a particular experiment's parameters. The client communicates with the Service Broker using a second web service interface.

The lab server knows nothing about the students using the system, and it only stores experiment specifications and results temporarily. The Service Broker authenticates students, checks on their authorization to contact a particular lab server, accepts an experiment specification from the student's client, and waits to retrieve the result once the experiment completes. The experiment specification and results are stored on the Service Broker, which also maintains the link between a student and his experiments. Thus all the resources consumed by a student except for the runtime resources required to execute the experiment can be drawn from a Service Broker located on the student's campus.

There must be a degree of trust between the Lab Server and the Service Broker, first and foremost because the Service Broker authenticates and vouches for student users. The Service Broker also indicates the student's level of access to the Lab Server by forwarding a string key known as the *effective group* when it submits an experiment specification. The Lab Server does not know on which student's behalf it is executing an experiment. It only knows the requesting Service Broker and the effective group associated with the request. This allows lab suppliers to grant different levels of access to different effective groups on multiple Service Brokers, but it delegates to the Service Brokers all decisions about which students or staff can request experiment execution

under the various effective groups. Conversely, the Service Broker knows nothing about the domain dependent nature of the experiments. It forwards an opaque object from the Lab Server to the student's client describing the current lab configuration. When the student submits an experiment specification, it is forwarded to the Lab Server as another opaque object, and the results are returned as a third. The only part of an experiment that the Service Broker understands is a metadata description of the experiment that can be used to search for and retrieve old experiments. This metadata is implemented as a class that contains fields common to all experiments (e.g., the Lab Server ID, the effective group, etc) as well as an XML-based extension mechanism. We assume that the Service Broker on one campus may give its students access to Lab Servers on multiple campuses, and a Lab Server may receive experiment specifications from Service Brokers on multiple campuses.

In this architecture for the batched experiment the student's workstation never contacts the Lab Server directly. We can maintain this strict discipline because the batched experiment requires so little communication between the client and the Lab Server. Conceptually, the execution of an experiment requires a single round trip although the actual Web Service protocol is more complicated. This simplicity obviously does not extend to the cases of a streaming sensor experiment or an interactive experiment in which there are strong arguments for having the Service Broker authenticate the student's client before allowing it to connect to the Lab Server directly. We plan to explore these variant architectures in 2003/2004.

## *The Batched Experiment Application Programming Interfaces (APIs)*

We have designed the batched experiment Service Broker in a way that maximizes the modules that can be reused in a more general architecture (see Figure 2). For instance, we expect that a future interactive experiment lab server will require web service access to what is currently an internal experiment storage API on the student-side Service Broker. For batched experiments, all calls to manage experiment data are internal to the Service Broker and do not need to be exposed as web services. But by segregating this code in a reusable library (DLL), we are laying the foundations for the future expansion of the Service Broker to cover all three types of experiments. Some of these architectures will probably require dissecting the unified Service Broker into finer grained services. The following documents describes the current APIs:

- The ***Client to Service Broker API*** describes the external web service API that specifies the connection between the domain-dependent client (e.g., an applet) and the Service Broker.
- The ***Service Broker to Lab Server*** is an external web service API that specifies the connection between the Service Broker and the domain-dependent Lab Server.
- The ***Service Broker Administrative API*** describes the internal .NET API to the library (DLL) that supports the Service Broker's administrative functionality, e.g. user and group management, adding and deleting lab servers, etc.

- The ***Service Broker Authentication API*** describes the internal .NET API to the functionality for identifying (authenticating) a user. The batched experiment release will supply a reference implementation based on a standard SSL-secured log name and password scheme. The interface will allow institutions to customize the Service Broker to use any standard authentication schemes supported on individual campuses (e.g., Kerberos, X509 certificates).
- The ***Service Broker Authorization API*** describes the internal .NET API to the library (DLL) that the Service Broker uses to check an authenticated user's permissions. The Authorization API distinguishes a student's from a professor's access to resources.
- The ***Service Broker Experiment Storage API*** describes the internal .NET API to the library (DLL) that supports the Service Broker's ability to store, search, and administer experiment records.

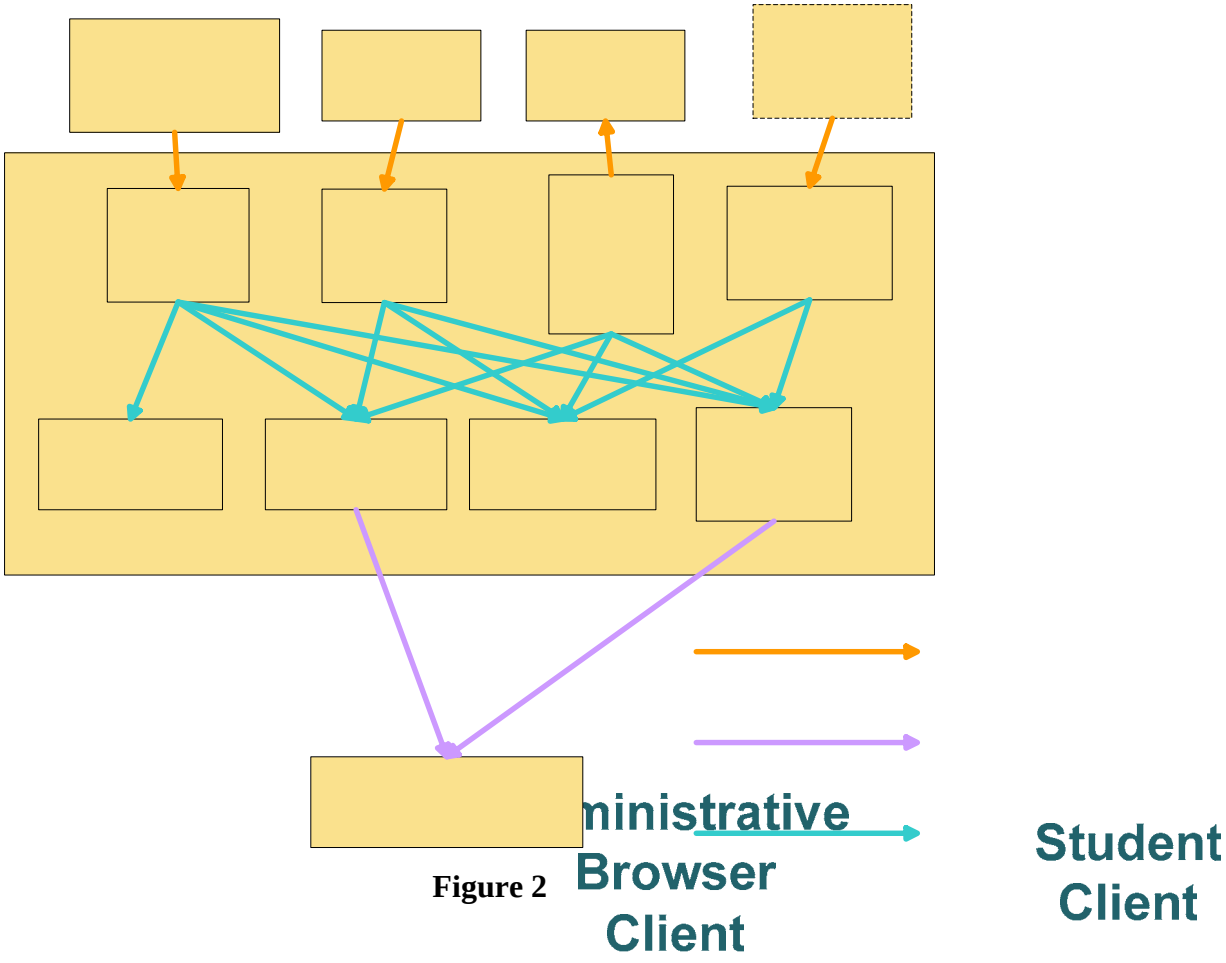


**Figure 2**

## *The Vision Beyond 2003*

During the summer of 2003, we implemented a prototype of the new Microelectronics Weblab on top of the web service architecture sketched above. This prototype, however, lacked all administrative functionality and supported only the Microelectronics WebLab and its associated client. We expect to have a fully functional Service Broker for batched

experiments with all administrative functions and support for multiple clients and lab servers by November 2003.

After the implementation of the batched experiment model, we plan to proceed in a number of directions:

1. We will extend the Service Broker architecture to support some sensor and interactive experiments with communication topologies that allow the student client to contact lab servers directly.
2. We will design and introduce a resource accounting scheme that is policy free, but will allow educational partners to implement multiple resource sharing models.
3. We will prepare a software release of the resulting code so that we can solicit feedback on its design and implementation. We intend this initial release to be taken up by colleagues on multiple campuses to implement their own online labs, but we do not expect it will represent a long term endpoint for the code. Looking back at MIT's history, we would be happy if this initial release played the role the X10 Window System did in shaping the far more influential and widely adopted X11.
4. We will actively engage industry, starting with National Instruments, to encourage their participation and to discover if the iLab architecture may open new marketing possibilities for such commercial partners.
5. We intend that steps 3 and 4 will lay the groundwork for a self-sustaining consortium under academic or joint academic-industry leadership.