Exploring Innovation in Education and Research
Tainan, Taiwan, 1-5 March 2005

The Microelectronics WebLab 6.0 – An Implementation Using Web Services and the iLab Shared Architecture

J. Hardison, D. Zych, J. A. del Alamo, V. J. Harward, S. R. Lerman, S. M. Wang, K. Yehia, C. Varadharajan

hardison@mit.edu, dzych@mit.edu, alamo@mit.edu, jud@mit.edu, lerman@mit.edu, smwang@mit.edu, kyehia@mit.edu, charuv@mit.edu

Microsystems Technology Laboratories and Center for Educational Computing Initiatives, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America

Abstract— *We have developed and deployed a new version of the MIT Microelectronics WebLab that has been constructed around the iLab Shared Architecture. The MIT Microelectronics WebLab (or simply WebLab) is an online semiconductor characterization laboratory. While WebLab is primarily of interest in microelectronics education, it also represents a testbed for new pedagogical and technological concepts associated with online laboratories. Our latest release, WebLab 6.0, is constructed around the newly developed iLab Shared Architecture. This is a new three-tier framework designed to expedite the development and simplify the management of online laboratories. The iLab Shared Architecture introduces a piece of middleware (termed the "Service Broker") between the Client application and Lab Server. The Service Broker uses Web Services to provide a generic set of functionality which is common to all labs. At the same time, it serves as a pass through between the Client and Lab Server for lab-specific information such as experiment specifications and data. WebLab 6.0 is the first lab to be deployed within this new architecture. The new WebLab 6.0 Client incorporates Java support for digitally signed applets. This allows it to break out of the Java security "sandbox" and interact more directly with the student's computer. In addition, the use of the open source package kSOAP enables the Client to communicate via Web Services. The WebLab 6.0 Lab Server has been rebuilt as a data-driven web application that communicates with the Service Broker via Web Services. The new WebLab 6.0 Lab Server also includes a persistence layer that stores system information along with an experiment execution queue as well as an Experiment Execution Engine that governs the execution of queued experiments on the lab hardware. WebLab 6.0 was successfully tested during the Spring 2004 semester in an undergraduate course on microelectronics at MIT involving over 100 students. During the Fall of 2004, several undergraduate and graduate courses at MIT as well as at other institutions made use of WebLab 6.0 for lab assignments. Additionally, the WebLab 6.0 code was released in October of 2004 as an exemplar since it was the first online laboratory implemented on top of the iLab Shared Architecture.*

Key Words— *iLab, microelectronics, online laboratory, WebLab, web services*

INTRODUCTION

The MIT Microelectronics WebLab Project has, since 1998, worked to provide an online device characterization experience for microelectronics students. Traditionally, microelectronics courses lacked a laboratory component. This is largely due to the various costs and complicated logistics involved in deploying such a lab for classes with a substantial number of students. Ideally, such a lab should use state-of-the-art industrial characterization equipment. Unfortunately, equipment of this nature is rather expensive, especially when multiple test stations are to be set up.

In terms of logistics, adequate space must be used to house the lab. This space needs to be easily accessible and large enough for students to use the equipment effectively. Lab staff who are properly trained to use the equipment safely must be hired. The physical security of the lab and its equipment must also be insured. The cost and complexity of these logistics has meant that many microelectronics students have often lacked a device characterization laboratory experience.

The Microelectronics WebLab, or WebLab for short, was developed to remedy this situation. At its essence, WebLab is a tool which allows students to perform current voltage measurements on microelectronic devices from anywhere via the Web [1], [2]. An Agilent model 4155B Semiconductor Parameter Analyzer is used to perform the device characterization measurements. This instrument is connected to eight devices-under-test, or DUT's, via an Agilent E5250A Low Leakage Switching Mainframe. Additionally, an Agilent 34970A Data Logging unit is used to measure and report the ambient temperature of the lab. This equipment is controlled by a Windows 2000 server which hosts the lab online. A Java based Client interface is used by students to control the hardware remotely. Thus, a single hardware setup can be made available to students across the world for a small marginal cost.

Until recently, WebLab was constructed according to a rather monolithic design. That is, all components of the lab, from user authentication modules to lab instrument drivers were built specifically for the Microelectronics WebLab. In addition to this, there were no clear organizational distinctions

between functionally different pieces of code. Over time, new functionality was added in an ad-hoc manner more befitting of a prototype than a finished product. As such, code modules, both within the Lab Server application and the Client interface, became increasingly interdependent and fragile. As a result, in 2000 when the iLab Project was launched at MIT to develop a variety of online laboratories in several engineering disciplines, only a small portion of WebLab's code could be recycled [3]-[6].

To address this deficiency of these first-generation laboratories, the iLab Project at MIT targeted in 2002 the development of a generic, or shared, architecture which would define a basic communication paradigm for online laboratories. In addition, the project aimed to provide turnkey software modules capturing the most generic subset of lab administrative functionality (i.e. user authentication and data storage). The goal of this effort was to decrease the work required to deploy an online lab while not imposing undue constraints on its operation and capabilities [6], [7].

The manifestation of this effort is the new iLab Shared Architecture. Figure 1 depicts the topology of a lab deployed within this architecture. At the center of this topology is the iLab Service Broker, which is a server that contains the functionality identified as being generic to most online labs. As of this writing, the Service Broker supports labs which are "batched" [6]. That is, labs whose experiments can be completely defined prior to execution and run in an unattended fashion. The Microelectronics WebLab is an example of a batched-type experiment.

The Microelectronics WebLab was selected first for deployment in the iLab Shared Architecture because of its relative maturity. There were also many lessons learned from several years of successful deployment that could be shared with the rest of the project. In addition to this, it would provide an opportunity for WebLab to get a well-deserved overhaul.

This paper will describe the redesign of the MIT Microelectronics WebLab for its deployment within the iLab Shared Architecture as well as provide an outline for the development of an online lab within the iLab Shared Architecture. The components of both the Lab Server and Client interface will be dissected, discussed and justified. Following that, the deployment of this new version of WebLab, termed WebLab 6.0, and integration with the generic iLab components will be discussed. This will include accommodations that had to be made in the integration process as well as initial real-world performance metrics. Finally, an overall evaluation of WebLab 6.0, with respect to previous versions, will be presented along with future directions for the Microelectronics WebLab.

REBUILDING THE LAB

Two major goals were identified for the redesign of the Microelectronics WebLab. First, the interdependence of internal modules as well as the overall fragility of previous versions had to be addressed. In both the Client and server components of the system, new functionality and updates were often made in an ad-hoc manner. Additions would be retrofitted to previous versions with minimal low-level integration. Often changes would be made to existing code only in cases where it was required to make a new feature work. This led to a system that was monolithic and rather difficult to maintain.

Deciding to deploy WebLab 6.0 within the iLab Shared Architecture was a starting point for achieving this goal. The fact that many required functions that are generic to all labs were provided by this architecture out-of-the-box meant that there were fewer actual code modules which had to be written by the lab developer. Beyond this, our approach had two primary requirements. Both the Lab Server and Client had to be designed according to complete, lab-specific functional specifications with clear, well-defined interfaces between individual internal components. Additionally, a more

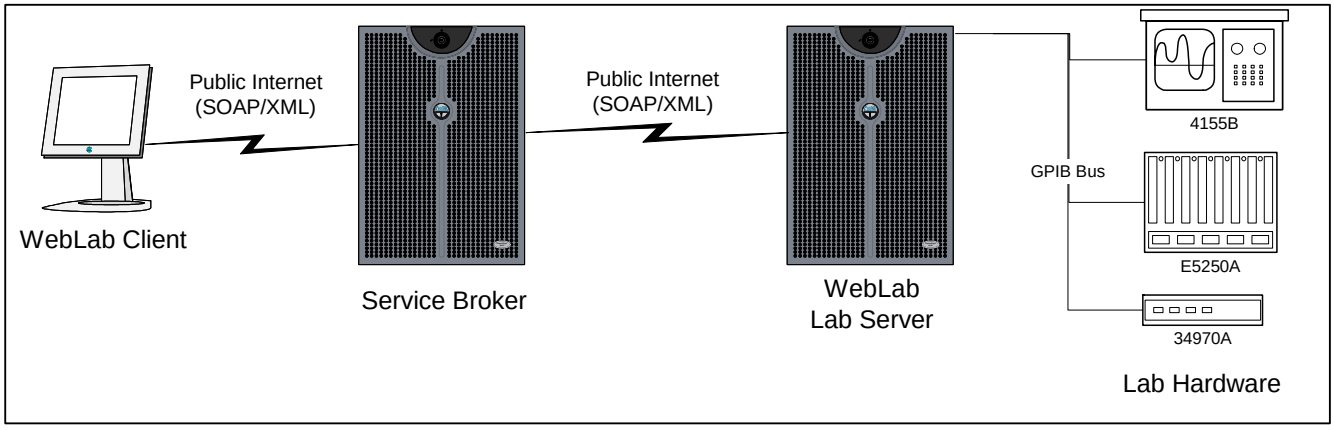

FIGURE 1
TOPOLOGY OF THE MICROELECTRONICS WEBLAB AS IMPLEMENTED WITHING THE ILAB SHARED ARCHITECTURE. THIS TOPOLOGY IS COMMON TO ALL BATCHED-TYPE LABORATORIES.

disciplined coding practice had to be established so that specified internal interfaces remain defined through future revisions.

The second major goal of the redesign effort was to enhance the educational value and technical performance over earlier generations of the system. One advantage of the monolithic design of previous WebLab versions was that everything was in the control of the lab developer. All parts of the user experience were defined by those running the lab. Furthermore, most software interfaces were internal to the same computer and, thus, would incur little overhead. Both these and the client-server interface (the only network interface in previous versions) could be tuned for a specific application.

In contrast, the iLab Shared Architecture is, by design, a distributed one [6], [7]. Similar to more monolithic designs, there is a Client interface and a Lab Server. However, in the shared architecture, a Service Broker exists between the Client and Lab Server. This Service Broker is the user's initial interface to the lab, performs user authentication and data storage functions and also serves as a launch point for the Lab Client. Once the Lab Client is launched, all communication from that client to the Lab Server is routed through the Service Broker. Communication between the Client and Service Broker as well as between the Service Broker and Lab Server are performed using standard Web Services and SOAP.

At first blush, this presents itself as a lot of added machinery with many more points where non-trivial overhead can be incurred. Again, the design of the Service Broker mitigates this to a degree. The Service Broker behaves only as a communication pass-through to the Lab Server for lab-specific requests from the Client. Thus, Client requests can be performed in one conceptual round trip regardless of the nature of the request. Also, the Web Service interfaces for the Client and Lab Server are terse, well-defined and platform independent. This allows for a clean division of labor between major components as well as flexibility in implementation. The onus is on the lab developer, then, to choose appropriate tools for their application and to take advantage of them to create a Lab Server and Client that perform well.

With the well-defined interfaces at either end of the Service Broker, there is a clear division of labor in developing a new lab within the shared architecture. Both the Lab Server and Client must be constructed according to their own, lab-specific functional specifications. The Web Services provided by the Service Broker specify the mechanism by which the Client and Lab Server communicate. Thus, the Client and Lab Server must be able to communicate lab-specific information with each other within the confines of their respective, generic interfaces with the Service Broker. To this end, there are a set of XML documents that define the content of the lab-specific communication between the Client and the Lab Server. This set includes the following documents whose schemas are lab-dependent:

- **Lab Configuration:** The purpose of the Lab Configuration Document is to give the server a vector for describing the current status of the lab to the Client. For the Microelectronics WebLab, this includes information of the specific microelectronic devices connected to the lab instrumentation.
- **Experiment Specification:** In this document, the Client defines specific execution parameters for a given experiment execution to be performed by the Lab Server. In the WebLab case, this includes what device to perform the measurement on, the individual terminal settings which will define the measurement as well as what data should be returned.
- **Experiment Results:** This document provides a context-aware way for experiment data for a given execution to be relayed from the Lab Server to the Client interface.

In summary, the Lab Configuration, Experiment Specification and Experiment Results documents form the basis for lab-specific communication between the Client interface and the Lab Server via the Service Broker. As such, the initial design of WebLab 6.0 focused around their definition. The following sections describe the individual development of the Client interface and the Lab Server.

REDESIGN OF THE WEBLAB CLIENT INTERFACE

In redesigning the WebLab Client interface, the previously mentioned overarching goals needed to be addressed. While it was important that the same basic feature set present in the older Client be included in the new version, it was critical that module interdependence and other legacy issues not be propagated. To that end, WebLab 6.0 was rebuilt from scratch. On the Client side there would be effort given to refining the look and feel of the user interface components, but the primary focus would be on building a Client that was educationally valuable, modular in design, extensible and compliant with the iLab Shared Architecture.

As with previous versions, the WebLab 6.0 Client interface is based on Java technology and deployed as an applet. The primary reason for this is the relative ubiquity of Java as a client execution environment. Reasonable cross-platform compatibility in tandem with the fact that a Java runtime environment is both included with most browsers and freely available as a standalone plug-in allows WebLab a maximum reach with little or no additional development. This has worked well for previous versions of the WebLab Client [1], [2], [11].

In terms of specific design, the WebLab 6.0 Client is an assembly of three major components with well-defined interfaces as described in Figure 2. The topmost layer is the User Interface component. This includes an experiment result graphing engine along with an otherwise thin set of presentation modules which govern only the look and feel of the interface. As modularity is an important design objective, the UI layer is constructed as an interchangeable component. The advantage of this is that the WebLab 6.0 Client could be constructed, initially, as a graphically based interface as shown in Figure 3. However, at a later date, interface changes can be

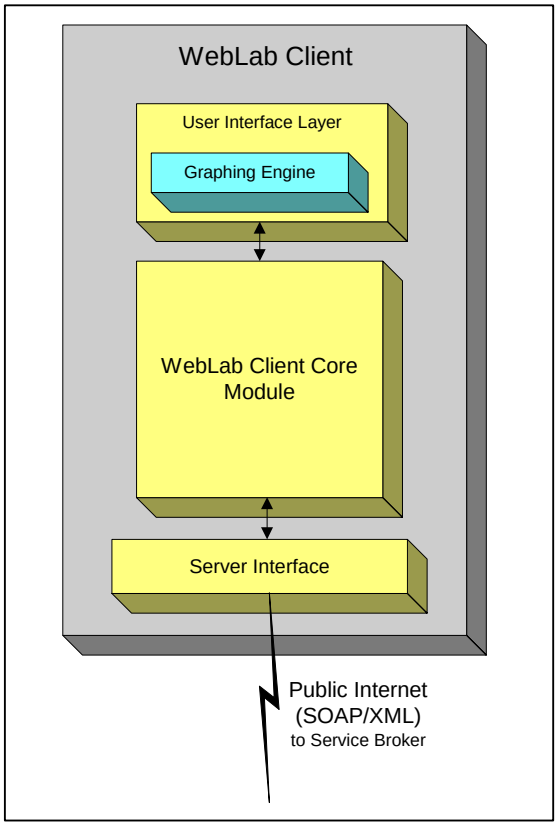

FIGURE 2
WEBLAB CLIENT INTERFACE COMPONENT DIAGRAM.

made without affecting the underlying Client behavior. This is accomplished by enforcing the use of a well-defined interface between the UI layer and the WebLab Client Core component.

The WebLab Client Core component is, essentially, the functional core of the WebLab 6.0 Client interface. This component contains all of the functional logic for the Client. This includes the back-end processing required for functionality at the UI layer as well as the construction/processing of the lab-dependent XML documents mentioned above. In particular, a significant challenge presented itself in the parsing of the Lab Configuration and Experiment Result documents. This task is relatively minor but the common XML parsing tools in the Java Web Services Developer Pack were far more robust than necessary [8]. More importantly, the inclusion of this in the Client would result in a rather prohibitive download size for the user. Fortunately, a solution was found in the SAX toolkit, which is supported in the standard Java Runtime Environment. This toolkit, while still providing more functionality than is strictly necessary, is much smaller and cleaner than its counterpart in the Web Services developer pack.

The third and final component of the WebLab 6.0 Client is the Server Interface. As its name suggests, this component governs the Client's communication with the iLab Service Broker. This layer communicates with the Service Broker via Web Services according to the Client to Service Broker API as defined by the iLab Shared Architecture [6], [7]. Once again, the requirement that this interface communicate via Web Services led to the challenge of finding a replacement for the unsuitably large Java Web Service Developer Pack. In this case, a replacement SOAP client was found in the kSOAP package [9]. This package, originally developed for use with the Java 2 Mobile Edition, provided the needed functionality while, again, keeping the overall download size reasonable.

Similar to the UI layer, the Server Interface is designed to be interchangeable. There is a well-defined interface between the WebLab Client Core component and the communication layer in order to enforce modularity. Thus, the Server Communication component present in the WebLab 6.0 Client can be cleanly replaced with another if support for a different communication protocol were necessary. This also means that the Web Service compliant component now used in the Client interface can be reused with minimal modifications in any Java based lab client constructed within the iLab Shared Architecture.

In a departure from previous WebLab Client designs, experiment validation logic has been relocated to the Lab Server. There are two main reasons for this. The first is that, with this change, the overall download size of the Client is

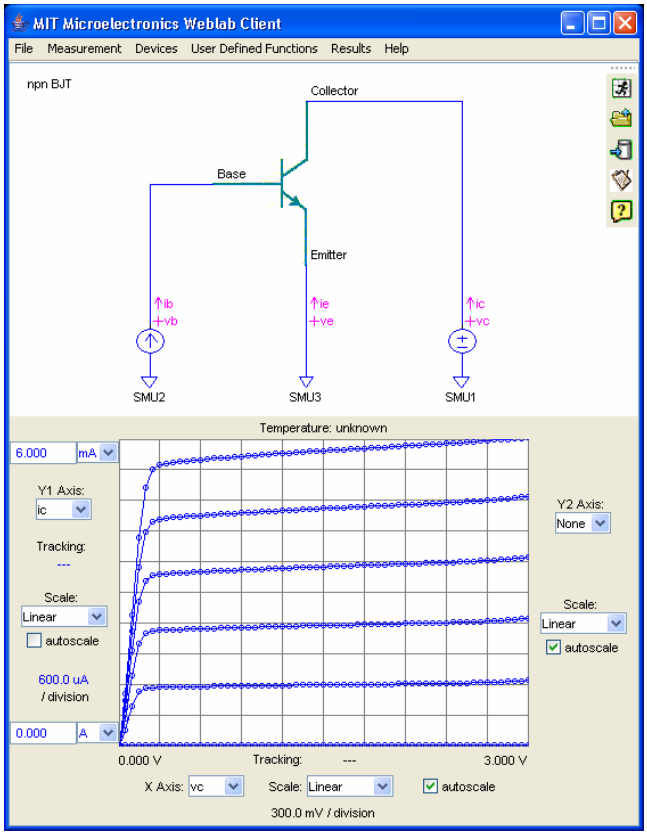

FIGURE 3
A SCREENSHOT OF THE WEBLAB 6.0 CLIENT INTERFACE.

reduced. This is an important consideration as the graphical nature of the applet as well as its need for additional components to support new functionality leads to a code base that threatens to be a formidable download. The second reason for removing validation from the Client is that its placement on the Lab Server centralizes the validation process. Where the Client may be distributed among Service Brokers in diverse locations and only loosely controlled, the Lab Server is almost always located at the lab itself. Placing validation on the Lab Server provides both an assurance that validation code will run on all submitted jobs as well as easier maintenance, with changes applying to all experiments instantaneously.

As a finished product, the WebLab 6.0 Client, as seen in Figure 3, requires that version 1.4.2 of the Java Runtime Environment be installed on the student's local machine. This is freely available on the Internet in the form of a browser plug-in [10]. The graphical nature of the Client is one reason for this. The use of a circuit schematic-based interface has proved to have significant educational value [11] but requires a code base more advanced than that currently included with most web browsers. Additionally, having to go to some lab-specific location to get the base circuit schematic images coupled with the requirement that all functional communication from the Client go to the general Service Broker presents what is traditionally a security violation in Java environments. That is, the requirement that an applet can only communicate with the server that launched it. The 1.4.2 version of the JRE supports an adjustment to this rule where a digitally signed applet can request added privilege from the user [12]. This feature is used to enable the Client's communication with multiple servers as well as its ability to save experimental results directly to the student's local machine.

Overall, the WebLab 6.0 Client is a marked improvement over previous versions. As the Client is based on the same underlying Java technology as previous versions, it maintains its usability across multiple computing platforms. The user interface is also similar to that of WebLab 5.0, thus maintaining its intuitive look and feel. However, the WebLab 6.0 Client's elegant, modular design makes it easier to maintain. The enhanced functionality of the new Client also comes in a smaller size than previous versions. Where the WebLab 5.0 Client was a 264 KB package, cleaner design and the relocation of validation code to the Lab Server reduce the 6.0 Client to 255 KB. This is no small feat considering the additional packages required for XML parsing and SOAP communication in the WebLab 6.0 Client.

REDESIGN OF THE WEBLAB LAB SERVER

In previous versions of WebLab, the Lab Server, while functionally complete, was monolithic and, ultimately, fragile. As was the case with the Client, the WebLab 6.0 Lab Server was effectively built from scratch in order to ensure that legacy issues and previous interdependencies did not creep into the new design. In addition to the overall design goals mentioned earlier, a specific goal of the Lab Server redesign was to increase its performance scalability and reliability with respect to previous versions.

To this end, the Lab Server was designed as a data-driven web application whose primary purpose is to receive, process and execute experiment specifications and return their results. As shown in Figure 4, this application is comprised of three

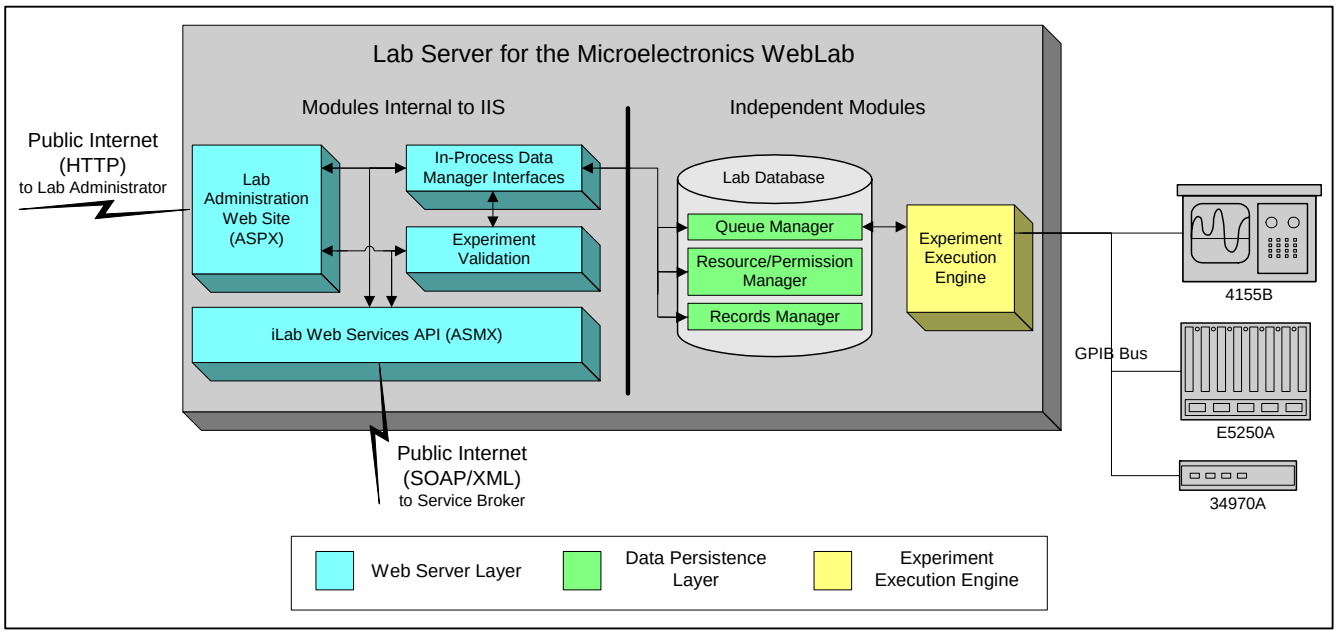

FIGURE 4
WEBLAB 6.0 LAB SERVER COMPONENT DIAGRAM.

major components. The core of the Lab Server is the Data Persistence Layer which serves as the information hub of the system. This layer is built with a Microsoft SQL Server 2000 database which fulfills three roles. The first is to store the operational data of the Lab Server. This includes device models and permission information which is used to define the current configuration of the lab with respect to the agent trying to use it.

The second role of the Data Persistence Layer is to maintain a queue of any experiment specifications awaiting execution as well as a record of past executions. A single database table is used for this purpose. The status of a given experiment specification or job (queued, in progress or complete) is stored as a state variable in the job's record. The job queue, then, is comprised of all records that are marked as "queued," which can be ordered by a Lab Server assigned priority and the time of submission. Alternately, the set of records marked as "complete" comprise the Lab Server's execution log. A persistent data store was used for the job execution queue, as opposed to using a construct within the memory of some Lab Server process, so that jobs would not be lost once submitted regardless of what happened to the other processes involved.

Finally, the Data Persistence Layer also defines the low-level mechanisms for the access of the data stored within it. This is achieved by the creation of a set of Data Manager Methods which provide an interface for common operations, such as the submission of a job or the creation of a device model, on the Data Persistence Layer. These Data Manager Methods are implemented as a series of stored procedures and user defined functions within the SQL database. The purpose of this is twofold. First, this ensures that informational requirements and dependencies of the data model can be accounted for within the database and contained within the interface defined by the Data Manager Methods. Second, these common yet, sometimes, complex operations can be performed within the database where there is machinery to make such operations more efficient.

On one side of the Data Persistence component is the Web Server Layer. This layer is governed by a Microsoft Internet Information Services 5.0 web server process which joins the Data Persistence component to the outside world. The interface defined by the Data Manager Methods discussed earlier are implemented by a series of VisualBasic.NET libraries for use by this layer. In addition to this, a similar code library provides access to the Experiment Validation Engine. The validation engine is the software component which parses a given Experiment Specification document, checks it against known hardware error conditions as well as server imposed access controls. Every job submitted to the Lab Server is analyzed by the validation engine before being entered into the execution queue. While this serves the functional requirements of checking the validity of jobs before execution, this also improves server performance as well as the user's experience. Most errant jobs are caught before further machine cycles or hardware resources are spent on them and, when they are caught, an error message that is more context-aware than that provided by the lab instrumentation can be generated.

At the exterior of the Web Server Layer are the web interfaces themselves. The Web Service Interface implements the Lab Server portion of the Service Broker to Lab Server API as defined by the iLab Shared Architecture [6], [7]. This interface is defined as a VisualBasic.NET Web Service class which delegates work to the in-process Data Manager and Validation Engine libraries. The security of this interface is guaranteed by two mechanisms. First, every call from the Service Broker to the Lab Server contains a previously agreed upon Server ID value and Passkey in the header of the SOAP envelope. These credentials are defined out of band when a Lab Server and Service Broker register with each other. At each call on the Lab Server Web Service Interface, the caller's level of access to the interface is determined based on these credentials. In order to secure these credentials during transport, a Secure Sockets Layer connection is required for communication with the Lab Server Web Service Interface.

Additionally, an Administrative Web Interface is defined within the Web Server Layer. This gives administrators of the Microelectronics WebLab a suite of online lab management tools. These tools are comprised of a series of ASP.NET pages which interface to the in-process Data Manager libraries as well as Data Persistence Layer when direct access is required for reading large datasets. This administrative site, while available on the public Internet, is only usable by lab administrators who can authenticate themselves to the system.

On the other side of the Data Persistence Layer is the Experiment Execution Engine. This third component of the Microelectronics WebLab Lab Server governs the execution of individual experiment specifications. This component is a VisualBasic.NET executable which runs in a process completely separate from the other components of the Lab Server. This process is launched when the host server boots and remains active while the machine is on. When idling, the execution engine checks for jobs in the experiment queue in the Data Persistence Layer via its Data Manager interface. If a job is found, it is loaded into the execution engine where it is parsed and prepared for execution. The top level of the execution engine configures the lab instrumentation with the specified parameters by calling methods defined by a set of drivers which have been written specifically to control the interface to the lab instrumentation. These drivers are the only legacy objects from previous versions of WebLab. They are Microsoft COM libraries which communicate with the lab instrumentation via an API provided by the manufacturer of the hardware interface between the computer and the GPIB bus connecting the lab equipment.

Once the job has been executed by the lab instrumentation and the results returned, the Experiment Execution Engine writes them to an XML Experiment Results document. This document, along with any hardware generated error messages or comments, is written to the appropriate experiment record in the Data Persistence Layer. Finally, the execution engine

attempts to prompt the Service Broker, by way of the Web Server Layer, to retrieve the experiment results as per the Lab Server to Service Broker API [6], [7].

This overall architecture achieves the additional goals of increased scalability and reliability, in no small measure, because of the strong separation of components. Primarily, the inclusion of a proper execution queue within the Data Persistence Layer means that job execution is asynchronous with job submission. This is a departure from previous WebLab Lab Server designs. This frees up operational bandwidth for both the web server process, which no longer has to wait for job execution to complete before moving on to the next web request, and the execution engine, which no longer has to share space with the web server. Furthermore, the separation of the Web Server and Experiment Execution Engine components means that if either of these components fail, the other can continue to operate. Thus, by enforcing modularity in design, overall reliability and performance benefit.

DEPLOYING WEBLAB 6.0

In January of 2004, the WebLab 6.0 Lab Server and Client were integrated with the iLab Service Broker in preparation for use in MIT's undergraduate level introductory microelectronics course that Spring term. This course typically has an enrolment of around 100 students and uses WebLab for two or three device characterization projects. Figure 5 details the hourly usage of WebLab 6.0 during a two-week assignment that term. In particular, the heavy usage towards the end of the assignment is typical and is where we would expect to experience problems. However, the system, as a whole, experienced no serious failures during this time while usage was at near-record levels. As of this writing, WebLab 6.0 is being used in MIT's graduate and undergraduate courses as well as courses offered in universities in Taiwan and Italy.

Additionally, in October of 2004 the WebLab 6.0 source code was released as part of the iLab Shared Architecture Dissemination Effort [13]. The goal of this effort is to prompt groups outside of MIT to develop online laboratories using the iLab Shared Architecture. As the Microelectronics WebLab was the first lab to be deployed using this architecture, it has been released as an exemplar along with the source code of the iLab Service Broker. This release is intended primarily for the purpose of commentary.

CONCLUSIONS & FUTURE WORK

In conclusion, a major architectural revision of the MIT Microelectronics WebLab has taken place in order to deploy

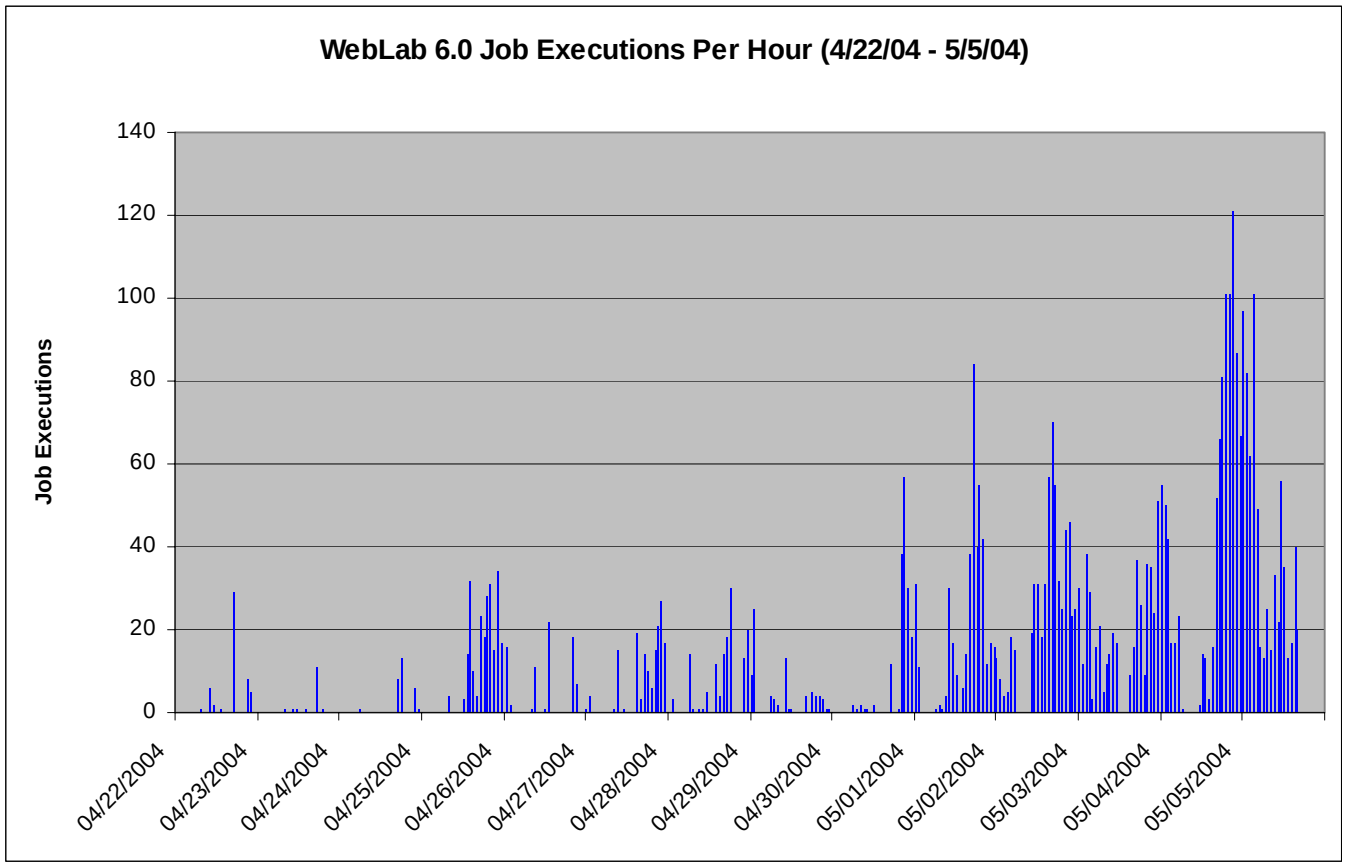

FIGURE 5
WEBLAB 6.0 JOB EXECUTIONS PER HOUR DURING A TWO-WEEK MIT COURSE ASSIGNMENT IN THE SPRING OF 2004.

WebLab within the new iLab Shared Architecture. Additionally, as WebLab 6.0 is the first lab deployed under the iLab Shared Architecture, it represents a complete, functional prototype of this new software framework for the development of new online labs. WebLab 6.0 marks a drastic departure, in terms of design, from previous versions. These changes, though, have resulted in a system that is both more robust and more scalable while maintaining its educational value. To that end, WebLab 6.0 has been and is currently deployed in courses both at MIT and abroad.

Looking forward, the WebLab 6.0 Client is in the process of being fitted with a tabular interface and being made compatible with older Java environments such as those included with most browsers for use in low-bandwidth areas. As a result of this development, further revisions of the Client's SOAP and XML modules have been made and will be applied to the current version. In the short term, both the WebLab 6.0 Client and Lab Server code will be released as exemplars as part of the iLab Shared Architecture Dissemination Effort. As a follow-up to the "for comment" release in October of 2004, this release will feature an install package for the iLab Service Broker as well as updated versions of the WebLab 6.0 source code.

ACKNOWLEDGEMENTS

This project is funded by Microsoft through iCampus, the MIT-Microsoft Alliance. The instruments used in WebLab were donated by Agilent Technologies.

REFERENCES

[1] del Alamo, J. A., L. Brooks, C. McClean, J. Hardison, G. Mishuris, et al., "MIT Microelectronics WebLab", chapter in T. A. Fjeldy and M. Shur, eds., *Lab on the Web: Running Real Electronics Experiments via the Internet*, John Wiley & Sons - IEEE, 2003. pp. 49-87.

[2] del Alamo, J. A., L. Brooks, C. McLean, J. Hardison, G. Mishuris, et al., "The MIT Microelectronics WebLab: a Web-Enabled Remote Laboratory for Microelectronic Device Characterization", World Congress on Networked Learning in a Global Environment, Berlin (Germany), 2002.

[3] Amaratunga, K. and R. Sudarshan, "A Virtual Laboratory for Real-Time Monitoring of Civil Engineering Infrastructure", ICEE, Manchester (UK), 2002.

[4] Colton, C. K., "iLab Heat Exchanger", http://heatex.mit.edu/.

[5] "Shake Table WebLab", http://flagpole.mit.edu:8000/shaketable/.

[6] Harward, V. J., J. A. del Alamo, V. S. Choudhary, K. deLong, J. L. Hardison, et al., "iLab: A Scalable Architecture for Sharing Online Experiments", ICEE, Gainesville, Florida (USA), 2004.

[7] Yehia, K., "The iLab Service Broker: a Software Infrastructure Providing Common Services in Support of Internet Accessible Laboratories", MIT Master of Science thesis, May, 2004.

[8] "Java Web Services Developer Pack", http://java.sun.com/webservices/jwsdp/index.jsp.

[9] http://ksoap.objectweb.org/.

[10] http://java.sun.com/j2se/1.4.2/download.html.

[11] del Alamo, J. A., V. Chang, J. Hardison, D. Zych, L. Hui, "An Online Microelectronics Device Characterization Laboratory with a Circuit-like User Interface", ICEE, Valencia (Spain), 2003.

[12] "Java Plug-in 1.4.2 Developer Guide", http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/contents.html, Ch. 16-19.

[13] "MIT iCampus: iLabs Architecture", http://icampus.mit.edu/ilabs/architecture/content/?ilabsdownload.