

Design and Implementation of the Feedback Systems Web Laboratory

by

Gerardo Viedma Núñez

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2005

© Massachusetts Institute of Technology 2005. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
January 27, 2005

Certified by
Dr. Kent H. Lundberg
Postdoctoral Lecturer
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Design and Implementation of the Feedback Systems Web Laboratory

by

Gerardo Viedma Núñez

Submitted to the Department of Electrical Engineering and Computer Science
on January 27, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis describes the design and implementation of a remote web-based laboratory (WebLab) for MIT's 6.302 Feedback Systems course. The WebLab system proposed consists of a three-tiered architecture where client and server communicate with each other via web services. On the front end, the user interacts with the system through the Lab Client's graphical user interface implemented as a Java applet. On the back end, the Lab Server processes experiment requests from users and runs them at the laboratory site. Once the experiment has been completed successfully, the Lab Server sends the measured data to the Lab Client for display on the screen and further manipulation by the user. Furthermore, the WebLab is designed to take advantage of the iLab framework for provision of authentication and authorization services, as well as common administrative tasks, such as user management and logging of experimental results.

Thesis Supervisor: Dr. Kent H. Lundberg

Title: Postdoctoral Lecturer

Acknowledgments

I would like to express my deepest gratitude and appreciation to my supervisor, Dr. Kent Lundberg. It never crossed my mind that a thesis project could evolve into such a pleasurable and rewarding experience. Much have I learned over the past year, and I am glad to admit that I enjoyed every moment of it. I believe Dr. Lundberg's enthusiasm, leadership and vision for the WebLab project had much to do with such a successful outcome.

It was also a great pleasure working side by side with the 6.302 staff during the deploy of the WebLab in the Fall of 2004. I am very grateful to all the 6.302 students who helped test and evaluate the system. Their feedback has been indispensable in enhancing the educational value of the WebLab for future generations of feedback systems students.

I also want to thank the iLab team for their continuous support and great work on the iLab architecture. Without their efforts, this thesis work would certainly not have been possible. I am also much indebted to the Microelectronics WebLab team, and James Hardison in particular, for providing much of the source code and expertise that were leveraged in building our system. In addition, I am obliged to MIT and the Department of Electrical Engineering and Computer Science, for helping support my thesis work through their Research Assistantship and Departmental Fellowship programs.

Last and most importantly, I would like to thank my parents whose boundless love and patience have never failed to encourage me in pursuing my goals. The completion of this thesis project is as much a fruit of their effort and perseverance as it is of my own. My most heartfelt congratulations go to them.

A mis padres y mi hermana Gloria; soy quien soy gracias a vosotros.

Contents

1	Introduction	11
1.1	Motivation	11
1.2	Related Work	12
1.3	Overview of this Thesis	13
2	The iLab Shared Architecture	15
2.1	Motivation	15
2.2	The iLab Framework	16
2.2.1	iLab Design Goals	16
2.2.2	Experiment Types	17
2.3	The Batched Experiment Architecture	18
2.3.1	The Case for Web Services	20
2.3.2	The iLab Batched Experiment API	21
2.3.3	Outline of a Student Batched Experiment Session	22
3	The Feedback Systems WebLab	25
3.1	Architecture Overview	25
3.2	Describing the Experiment Domain	26
3.2.1	Experiment Routine	28
3.2.2	Lab Configuration	31
3.2.3	Experiment Specification	32
3.2.4	Experiment Result	33

4	The Feedback Systems Lab Server	35
4.1	Architecture	35
4.2	ASP.NET Web Services Architectural Overview	36
4.2.1	Security	38
4.3	Database	39
4.4	Experiment Engine	41
4.4.1	Preparing the Experiment	41
4.4.2	Running the Experiment	42
5	The Feedback Systems Lab Client	47
5.1	Web Services	49
5.2	Dynamic UI Components	50
5.3	Functionality	52
6	Field Trial for the Feedback Systems WebLab	55
6.1	Student Experience	55
6.2	Statistics	56
6.3	Student Response to the WebLab	63
6.4	Lessons Learned	64
7	Conclusion and Future Work	67
7.1	6.302 and the Feedback Systems WebLab	67
7.2	The Feedback Systems WebLab beyond 6.302	68
7.3	Future Prospects for the Feedback Systems WebLab	69
A	XML Schema Definitions	71
A.1	Experiment Routine	71
A.2	Lab Configuration	73
A.3	Experiment Specification	75
A.4	Experiment Result	75

B	Defining Experiments for the Feedback Systems WebLab	77
B.1	Experimental Setup	77
B.2	Creating an Experiment for Execution	80
B.3	Obtaining the Results	80
C	Integrating the Feedback Systems WebLab into iLab	83
C.1	Registering the Lab Server	83
C.1.1	Updating the Service Broker	84
C.1.2	Updating the Lab Server	85
C.2	Registering the Lab Client	86
C.2.1	Updating the Service Broker	86
C.2.2	Publishing the Lab Client	87
C.3	Managing Users	87
D	Configuration Information for the Feedback Systems WebLab	89
D.1	Lab Server iLab Configuration	89
D.2	Lab Client iLab Configuration	90
E	6.302 Feedback Systems Fall 2004 iLab Assignment	91
F	Fall 2004 6.302 iLab Survey	97
	Bibliography	99

Chapter 1

Introduction

Remote laboratories became a reality with the advent of distributed systems that could make use of computer networks to communicate. The Internet allows anyone who satisfies some minimal requirements (e.g. a Java-enabled browser) to conduct an experiment from anywhere and at any time. This development has provided opportunities to explore new teaching methodologies that make use of these technologies to enhance science and engineering courses. An example of this is the current project of building a web-accessible laboratory for MIT's Electrical Engineering course 6.302 Feedback Systems.

1.1 Motivation

The Feedback Systems Web Laboratory [1] (henceforth referred to as *WebLab*) provides an effective means for students to conduct experiments from a variety of locations through a web browser. Using our remote web-accessible laboratory students are able to conduct experiments at any time and any place that is convenient for them. In addition to its flexibility from the student point of view, the WebLab also helps alleviate the load on teaching assistants and professors. In a successful remote laboratory, their physical presence in the lab will no longer be required while each and every student completes their lab assignment.

Furthermore, a remote laboratory provides an excellent means to time-share ex-

pensive and scarce equipment. This consideration was one of the primary motivations in building the Feedback Systems WebLab, for which the Dynamic Signal Analyzer represents an expensive piece of lab equipment that is very difficult to share efficiently among students in a conventional lab setting.

1.2 Related Work

There have been many approaches to the design of Internet-based remote laboratories (henceforth referred to as *weblabs*) for control education. Early systems required specialized platform-dependent software running at the client computer [2, 3, 4, 5]. Later approaches moved towards browser-enabled technologies for the client, including Java applets [6], static and dynamic HTML pages [7], and CGI scripts [8]. HTML-based solutions often result in thin clients with little processing abilities and rely heavily on server-side technologies such as CGI that tightly couple client and server development [9].

Most current designs employ Java applet technology for the client environment, due to Java's processing abilities and platform independence. Many of these systems rely heavily on TCP/IP sockets for communication [10, 11]. Although an efficient means for client to server communication, sockets require client developers to grapple with a style of programming radically different from the object-oriented paradigms they are accustomed to.

Instead, the iLab architecture [12] provides a common framework for lab development and deployment. This approach differs from sockets-based solutions by hiding many of the details involved in network communication from the developer. This goal is achieved by using web-service technology, which provides an object-oriented interface to client/server communication based on traditional method calls that take place over HTTP.

In addition, the iLab architecture limits the amount of implementation work that needs to be done by weblab developers and administrators. Previous remote laboratory designs have wrestled with the provision of administrative services not specific to

the laboratory. In doing so, it has been the tendency to include this kind of functionality at the server end along with the laboratory-specific services [6, 13]. In contrast, the iLab architecture decouples laboratory-specific operations related to running experiments from the more generic administrative tasks of user authentication, user authorization, group management, and results-storage functionality.

The iLab architecture naturally extends the client/server weblab topology by incorporating an additional third tier: the *Service Broker*. The Service Broker handles all administrative tasks, thus freeing the server machine (and the weblab developers) from having to implement custom administrative solutions for each new weblab. Our approach to building a linear-systems web-based laboratory that integrates with the iLab framework is summarized in [14].

Finally, MIT's iLab project has already produced a number of functional labs for a variety of different courses among a diversity of disciplines. It is a goal of this project to contribute to the iLab initiative, by making use of much of the existent infrastructure and know-how that has already been put in place through the creation of other iLab-based weblabs. In particular, this project is indebted to the developers of the 6.012 Microelectronics Devices and Circuits WebLab [15], who gratefully provided much of the framework and tools that were leveraged in the implementation of the Feedback Systems WebLab.

1.3 Overview of this Thesis

The current chapter provides an introduction and motivation for this research, and discusses some of the previous work done in the area. Chapter 2 describes the iLab framework upon which our WebLab is based. Details for our particular WebLab implementation for the Feedback Systems course are given in Chapter 3. Chapters 4 and 5 provide a detailed account of the Lab Server and Lab Client implementations respectively. Finally, a discussion of the student experience and the lessons learned from the WebLab's deployment in a class setting is provided in Chapter 6, followed by some concluding remarks and suggestions for future work in Chapter 7. The integra-

tion of our WebLab into the overall iLab framework is documented in Appendix C.

I have applied the following typographical conventions in the remaining pages. Web service methods, SQL procedures and namespaces are provided in **true-type**. SQL tables and software classes appear in **sans-serif**, variables and parameters are *slanted*, and UI/menu options and new terms are introduced using *italics*.

Chapter 2

The iLab Shared Architecture

2.1 Motivation

The iLab project [16] developed as part of the iCampus initiative [17] to promote online laboratories at MIT. Even though there is a great educational value in hands-on laboratory experiences, conventional laboratories suffer from a number of important drawbacks. First, they tend to be costly and involve complex logistics [18]. Expensive equipment needs to be time-shared and scheduled for use, and requires lab space, lab staffing and training, as well as involving issues of safety. Secondly, conventional labs do not scale very well, making it exceedingly difficult to share equipment as the number of users increases. They also impose severe limitations on the geographical location of potential users, who must necessarily be physically present in lab in order to run experiments.

Online labs share many of the advantages of conventional labs in delivering the educational benefits of hands-on experimentation, while overcoming their biggest limitations. Moreover, these labs are not limited to providing simulations nor running “canned experiments” (although this functionality can still be easily included). Instead, online labs provide a virtual interface to real laboratory hardware that can now be accessed over the Internet from anywhere and at any time [19].

Online labs are also unique from a pedagogical perspective. For instance, they allow laboratory experiments to be introduced at the most opportune moment in

the curriculum, and are very flexible, allowing students to perform experiments in pleasant environments at the times of their choice. Moreover, by providing simple and more intuitive interfaces to laboratory equipment, they can help minimize student frustrations with hardware that can often detract from the educational effectiveness of traditional lab work. In addition, online labs greatly facilitate the collection and manipulation of experiment data, which students can now easily export to a number of different formats and applications for analysis, comparison and visualization [6].

2.2 The iLab Framework

2.2.1 iLab Design Goals

The iLab framework was designed to allow the usage of online labs to scale to a large number of users geographically dispersed throughout the world. Its goal is also to decouple the generic administrative operations from those involved in the implementation of particular labs. Consequently, the iLab framework provides a set of generic services relating to user authentication and authorization, group management, experiment specification and result storage, as well as lab access scheduling. In this way, lab operators need not develop custom solutions for individual user management and data storage, but can instead focus on lab-specific development.

Scalability of the system is also ensured by delegating the implementation of administrative and user-management policies to each of the universities or research centers participating in it. The delegation of control and authentication policies allows and encourages universities with diverse network infrastructures to interoperate and share access to lab equipment.

The long term vision of the iLab project is for the educational content of online labs to be broadly shared around the globe, enhancing science and engineering education by multiplying the lab experiences students will ultimately have access to [18].

2.2.2 Experiment Types

The iLab framework defines three broad categories of online experiments [12]:

1. *Batched experiments*: those in which the entire course of the experiment can be specified before the experiment begins. The current Feedback Systems WebLab provides an example, where students submit the set of experimental parameters that are needed to completely characterize the experiment task.
2. *Interactive experiments*: those in which the user can monitor and dynamically modify one or more inputs to the experiment during its execution.
3. *Sensor experiments*: those in which users monitor or analyze real-time data streams without influencing the phenomena being measured.

The very different characteristics of the above types of experiments result in a variety of requirements for the shared architecture. In a batched experiment, users completely specify their experiment before submitting it for execution. As a result, users need not be online during experiment execution, but can retrieve their results at a later time. For this reason, it is appropriate to satisfy experiment requests in a manner that maximizes the efficient use of the Lab Server rather than convenience for the user.

On the other hand, interactive experiments require that the user be online during execution in order to adaptively control and alter the experiment inputs. In this scenario, it becomes necessary to schedule experiments that take longer than a few minutes to execute. Scheduling thus prevents users from having to wait for long periods of time before the experiment apparatus becomes available.

Finally, sensor experiments provide no mechanism to directly control an experiment. Moreover, these experiments differ fundamentally from batched experiments and interactive experiments in which users' experiment requests are executed sequentially. Instead, they allow users to subscribe to a number of different data streams providing different resolutions or transformations of the base data. Consequently, sensor experiments offer the ability to multicast the same data to a multitude of users at

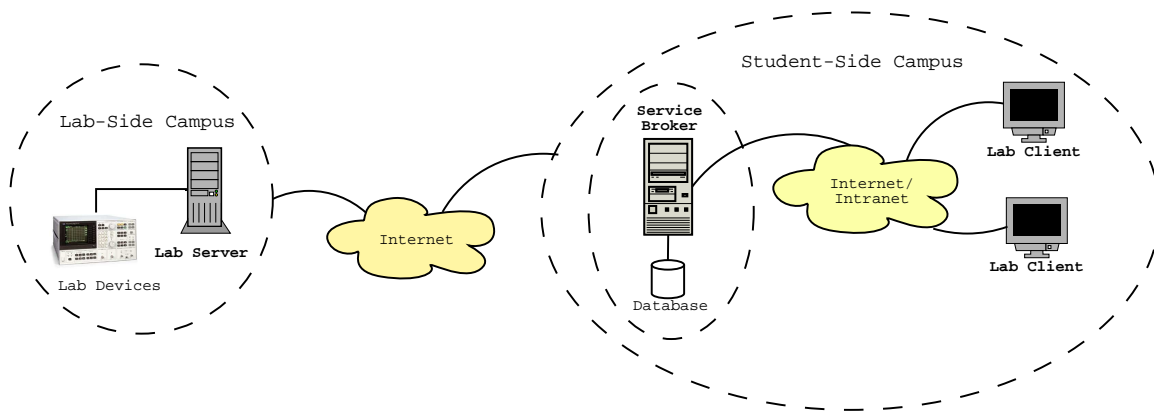


Figure 2-1: Architectural overview of the three-tiered iLab system. The Service Broker handles all administrative tasks, thus freeing the Lab Server (and its developers) from having to implement custom administrative solutions for each different weblab. The Service Broker architecture also simplifies the sharing of iLabs between universities, by alleviating the lab-side (host) university from administering guest users. The host university can grant access to the student-side (guest) university’s Service Broker, and the guest university can then administrate its own users.

once. They can also be used to generate archives of data gathered over extended periods of time. These data can then be employed at a later time for statistical analyses and searches of events of interest.

Given the relative novelty of the iLab project, only the batched experiment architecture has been tested and fully deployed at the current time. The interactive and sensor experiment architectures are still undergoing heavy development, and prototypes should become available in 2005 and 2006 respectively [12].

2.3 The Batched Experiment Architecture

The iLab framework proposes a three-tiered topology for batched experiments based on web services, as show in Figure 2-1:

1. The first tier consists of the **Lab Client** application that either runs as an applet on the user’s browser, or alternatively as a downloaded application on the user’s workstation.

2. The middle tier provides the shared common services by means of a **Service Broker**. The Lab Client communicates solely with the Service Broker, which forwards the requests to the final third tier. In the most common scenario, the Service Broker resides in the client side of the network, on a server at the student's institution. However, the architecture still allows for the Service Broker and Lab Client to reside on different networks, for example, when granting accounts for users at other collaborating institutions.
3. The third and last tier is composed of the **Lab Server**, which usually resides at a specified laboratory site on campus. The Lab Server's task is to execute the experiments specified by users, notifying the Service Broker when their experiment has been completed and results are available to be retrieved.

In this framework, the Service Broker consists of completely lab-independent generic code, and knows nothing about the domain dependent nature of the experiments. On the other hand, the Lab Client and Lab Server constitute lab-dependent tiers that must understand and speak a common protocol for describing the experiment universe (e.g. when specifying experiment parameters and results). Using this protocol, experiment requests and results are forwarded from Lab Client to Lab Server as opaque objects through the Service Broker. However, the Service Broker does not understand the contents of these opaque objects beyond the metadata description of experiment requests, such as Lab Server IDs, etc.

Furthermore, individual students are also abstracted away from the Lab Server, which does not know on which student's behalf it is executing a given experiment. Instead, the Service Broker authenticates and authorizes students to contact a particular Lab Server, and then assigns them to an *effective group* when submitting an experiment specification on their behalf at the Lab Server. This scheme enables lab implementors to grant different levels of access to different effective groups, while delegating administrative decisions regarding group membership and management to the Service Broker. It also maintains the Service Broker as the single point of contact to the Lab Server, since students never need to contact or even be directly aware of

the location of the Lab Server.

2.3.1 The Case for Web Services

The choice of network technologies on which to build the iLab shared architecture has a great impact on the capabilities of the resulting distributed application framework. The iLab shared architecture favored the use of web services based on its design requirements emphasizing interoperability, software reuse, lab discovery and licensing possibilities [12].

First of all, it is crucial that the architecture support lab-side services as well as client-side services (including authentication and authorization, and experiment data storage). These two services will often run on separate networks, and possibly under different hardware and software platforms. Furthermore, the iLab architecture must be able to transparently support lab-side institutions which enforce different networking policies (firewalls and network services) from client-side institutions. Web services built on top of the SOAP [20] standard provide a platform and language-independent protocol for exchanging information in a decentralized and distributed environment. Moreover, SOAP web services are based on XML [21] and are adaptable to the Internet, since all communication takes place over the HTTP protocol. These advantages make web services a transparent and interoperable network technology ideal for the integration of iLab's distributed online laboratory framework.

Often, labs will possess a preexisting code-base that was independently developed by lab experiment owners and course staff. Web services makes it possible to leverage that previous development effort by reusing such legacy code when deploying these labs as iLab-enabled weblabs. In addition, the loose coupling of web services allows lab developers to more easily integrate vendor specific modules (e.g. National Instruments' LabView [22]) into their weblabs, thus potentially reducing total development time.

Looking at the future of the iLab project, web services technologies herald enormous possibilities in publishing and discovery methods for online services. Employing web services WSDL [23] and UDDI [24] technology, weblab owners will be able

to publish their online labs services to the entire world, fostering the cooperation between educational institutions as more online labs become available. Moreover, WSDL-based negotiation will provide the possibility of matching Internet accessible labs with high-end visualization and data analysis tools licensed at the student-side institutions.

2.3.2 The iLab Batched Experiment API

In order for iLab-enabled weblabs to be truly distributed, the different layers must adhere to the set of standardized operations stipulated in the iLab API. These operations are implemented over web services and the SOAP protocol for message exchange.

The iLab shared architecture defines two sets of web service methods composing the iLab API: service calls from Lab Client to Service Broker [25], and service calls from Service Broker to Lab Server [26].

In fact, the bulk of the iLab API consists of *pass-through methods*, whose function is simply for the Lab Client to call a corresponding method from the Lab Server API. We summarize the most important ones here:

- **GetLabStatus**: checks on the status of the Lab Server.
- **GetEffectiveQueueLength**: checks on the effective queue length of the Lab Server.
- **GetLabInfo**: gets general information about a Lab Server.
- **GetLabConfiguration**: gets the lab configuration of a Lab Server.
- **Validate**: checks whether an experiment specification would be accepted if submitted for execution.
- **Submit**: submits an experiment specification to the Lab Server for execution.
- **Cancel**: cancels a previously submitted experiment.

- **GetExperimentStatus**: checks on the status of a previously submitted experiment.
- **RetrieveResult**: retrieves the results from a previously submitted experiment.

In addition, the Service Broker publishes the **Notify()** web service method, which may be called by the Lab Server to announce that an experiment has been completed successfully and its results are ready to be retrieved.

2.3.3 Outline of a Student Batched Experiment Session

The following walk-through of a student experiment session illustrates the main interactions and web service calls between Service Broker, Lab Server, and Lab Client that take place during a batched experiment run [12].

1. First, the student must log on to the Service Broker through an active server page, by supplying a user name and password for authentication.
2. The Service Broker responds by displaying a list of possible user groups for which the student is registered. Upon selecting one, the Service Broker displays a list of the available Lab Clients for the selected group.
3. Upon selection of a Lab Client, the corresponding Java applet loads on the student's browser.
4. Using the Lab Client's user interface, the student edits the description of the experiment to be run at the laboratory site. Once complete, the student directs the Lab Client to invoke the web service **Submit()** method on the Service Broker. **Submit()** takes a text encoded version of the experiment specification as an argument, which the Service Broker is not expected to understand.
5. The Service Broker then stores a copy of the experiment specification and forwards the **Submit()** call on to the Lab Server.

6. At this point, the Lab Server receives the experiment specification and validates it for correctness. If legal, the Lab Server queues the experiment for execution. It then returns a submission report that includes an error message in the case of an invalid specification.
7. The Service Broker forwards the submission report to the Lab Client, along with an experiment ID that is now used by all parties to identify the experiment.
8. Upon successful completion of the experiment, the Lab Server calls the `Notify()` web service method on the Service Broker to indicate that the experimental results are now available to be retrieved.
9. The Service Broker then requests the results from the Lab Server by means of the `RetrieveResult()` web service call.
10. The Lab Server returns the experimental results and any error messages to the Service Broker, which stores them but is unable to interpret them.
11. Finally, the Lab Client can request the cached results from the Service Broker by calling the `RetrieveResult()` web service. The Service Broker then returns the results and any error messages, which the Lab Client is now able to interpret and display to the student.

Chapter 3

The Feedback Systems WebLab

This chapter describes the implementation strategy for the Feedback Systems WebLab. First, the system architecture and its main components are briefly introduced. We then define a common syntax and semantics for describing experiments and their results between Lab Client and Lab Server.

3.1 Architecture Overview

The experiments composing the Feedback Systems WebLab are entirely specified by the user before the experiment begins. Our WebLab thus falls into the batched experiment category. Consequently, the architecture of the Feedback Systems WebLab is based on the three-tiered batched experiment architecture described in Section 2.3.

The iLab architecture for batched experiments suggests a distributed system of three layers for building the WebLab. These three tiers communicate with each other via SOAP messages exchanged through web services, as shown in Figure 3-1. The first tier is made up of the *Lab Client*, which runs on the user's machine as a Java applet that can be loaded via a Java-enabled web browser. The Lab Client is the only component in the system that is visible to the end user.

The middle tier consists of a trusted intermediary, known as the *Service Broker*, which forwards requests from the Lab Client to the Lab Server for execution. The main roles of the Service Broker are to authenticate users and grant them appropriate

permissions when forwarding their requests for experiments. The Service Broker also provides additional administrative services, such as group management, and the ability to temporarily store experiment requests from the Lab Client, and experiment results from the Lab Server.

Finally, the *Lab Server* constitutes the third tier in the architecture. Its task is to receive requests from the Lab Client via the Service Broker, and to reply to these requests also via the Service Broker intermediary. Usually, the Lab Client requests an experiment to run with a particular set of experimental parameters. In this case, the Lab Server communicates with the laboratory equipment, runs the experiment on the system under test, and finally replies with the experimental results upon successful completion of the experiment.

A detailed account of the Feedback Systems Lab Server implementation strategy is provided in Chapter 4, while Chapter 5 explains the design of our Lab Client. Both the Lab Server and Lab Client described in this thesis build upon the Feedback Systems WebLab prototype described in [27].

3.2 Describing the Experiment Domain

The iLab framework stipulates three different specifications for describing the experiment universe. The content of these specifications is unique to the Feedback Systems WebLab, and provides a common understanding of the experiment world between Lab Client and Lab Server. In addition, the Feedback Systems WebLab was designed in such a way that these specifications may reside anywhere on the World Wide Web. As a result, the experiment can be modified remotely by anyone with the appropriate permissions (for example, the appropriate teaching assistants and professors).

The three specifications defined by the iLab framework are the *Lab Configuration*, the *Experiment Specification* and the *Experiment Result*. Moreover, the Feedback Systems WebLab introduces one additional specification: the *Experiment Routine*. In order to facilitate interoperability and the transfer of information across the Web, instances of these specifications are encoded using the syntax of the Extensible Markup

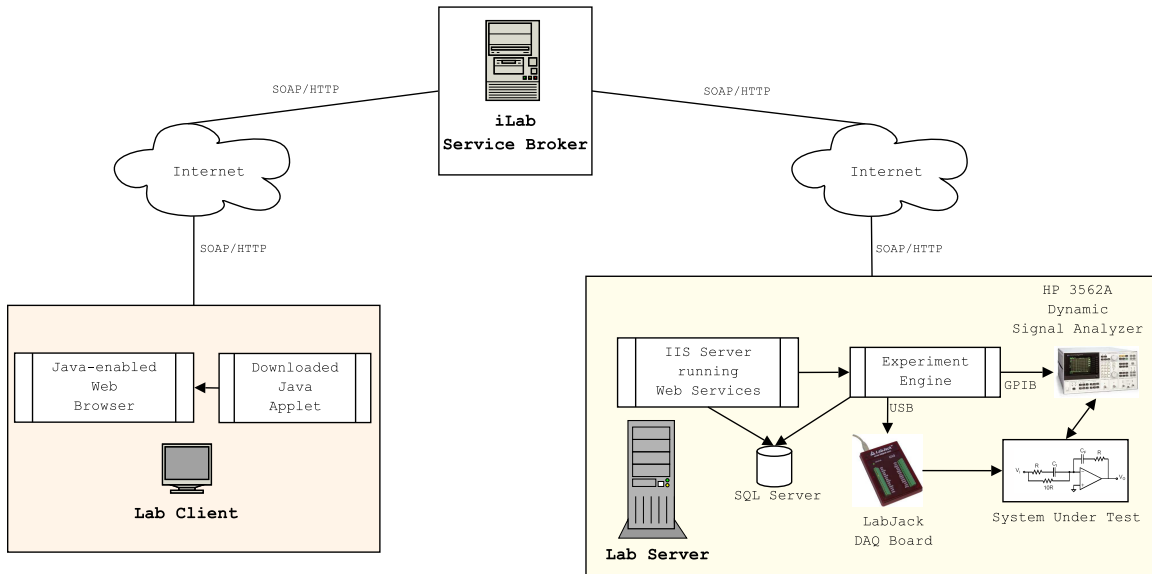


Figure 3-1: The Feedback Systems Weblab Architecture. The three-tiered architecture for the Feedback Systems Weblab consists of the Lab Client, the Service Broker, and the Lab Server. The Lab Client runs as a Java applet on the client's computer. The Service Broker provides generic weblab services for a variety of iLabs. Finally, the Lab Server communicates with the HP 3562A dynamic signal analyzer to set up frequency response measurements on the system under test. It can also set various command signals at the system under test via the LabJack™ DAQ board. All communication between tiers takes place using SOAP web service calls over HTTP.

Language (XML). XML also provides a useful way to store structured information and encapsulate it, making it easy for different computing systems to intercommunicate, as described in [21].

While actual instances of these specifications are encoded in XML, the actual structure of the specifications is defined by a corresponding XML schema written in the XML Schema language [28]. The XML schema thus defines the building blocks, consisting of elements and attributes and their underlying data types, that make up the specifications. An instance of a specification, however, contains the actual values for these elements and attributes, and must conform to the structure defined in the corresponding XML schema.

XML Schemas hold a number of advantages [29] over their predecessor DTDs [30]. For example, they are more easily maintained, support data types and name spaces, are extensible and modular, and are written entirely in XML. For these reasons, we

decided to convert our original specifications for the WebLab from the original DTD definitions to ones using XML Schema.

3.2.1 Experiment Routine

The first of the specifications defined for the Feedback Systems WebLab is the Experiment Routine. It must be noted that this specification is not defined by the iLab framework, and thus plays no role within the Lab Server-to-Service Broker or Service Broker-to-Lab Client APIs outlined in Section 2.3.2.

Why introduce yet another specification?

The reason why we decided to introduce this additional specification was to have a single file through which to completely characterize any given experiment. As described in Section 3.2.2, the Lab Configuration also serves the purpose of describing experiment configurations. However, this specification is received at the Lab Client and should thus be limited to client-specific information regarding the experiment.

On the other hand, we would also like to include information regarding other experimental parameters necessary for the Lab Server to run the experiment on the lab hardware. Such additional information does not need to and in fact should not be made available to clients via the Lab Configuration. The Feedback Systems WebLab thus introduces the notion of the Experiment Routine, containing both Lab Client and Lab Server-specific information regarding an experiment, all encapsulated in a single file.

Moreover, for convenience and so that only one file need be edited by course staff and system administrators, the Lab Configuration is dynamically generated from the Experiment Routine at the Lab Server. In this way, Lab Client-specific information can be safely handed to clients through the iLab API, while allowing the Lab Server-specific experimental parameters to be hidden away from clients that need not or should not be made aware of it.

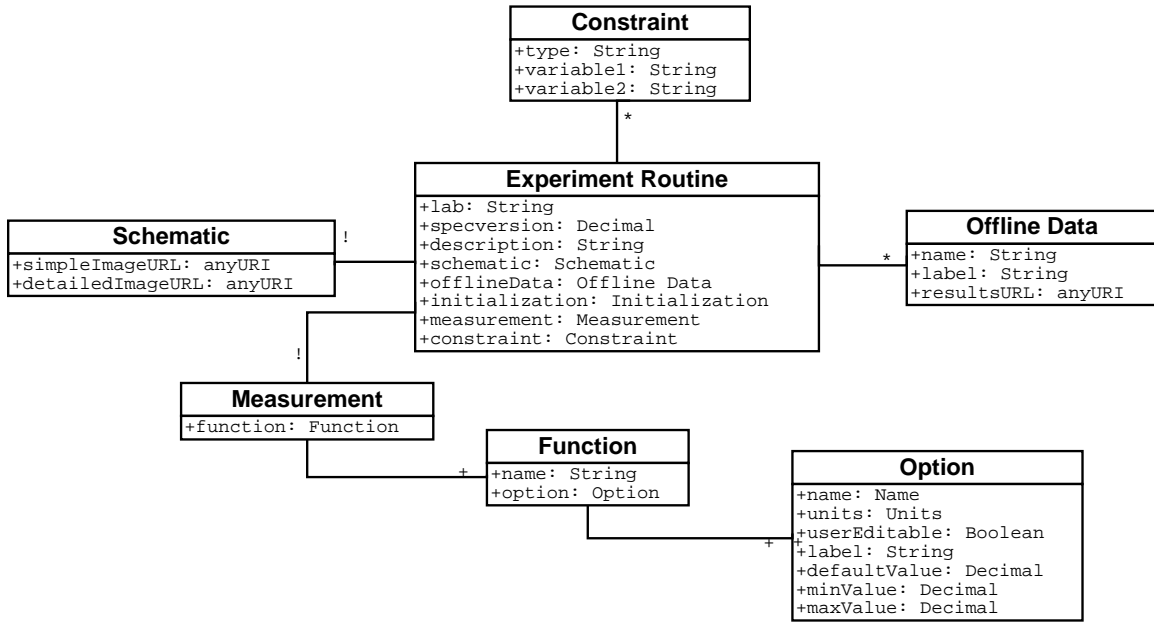


Figure 3-2: An object model representation of the Experiment Routine. The Experiment Routine encapsulates a complete experiment setup and measurement routine at the lab hardware. Thus, it defines the measurement parameters characterizing the experiment to be performed at the Lab Server. In addition, it provides links to the experiment’s schematic diagram, and other generic lab-specific information such as the lab name and its description. Lab administrators also have the ability to provide experimental data for users to download via offline data, and can specify simple constraints between measurement parameters.

Details of the Experiment Routine

As its name implies, the Experiment Routine is tightly bound to a particular experiment, and describes the set of experimental operations that need to be performed at execution time. The set of operations supported by WebLab experiments is directly determined by the lab hardware employed at the Lab Server. Currently, the WebLab incorporates an HP 3562A digital signal analyzer in order to obtain the frequency response measurements on the system under test (SUT). In addition, the Lab Server can set input voltages to the SUT by communicating with a digital acquisition board over the USB port. Actual details of our lab hardware setup are provided in Chapter 4.

Reflecting the hardware capabilities, the Experiment Routine stipulates a set of *initialization* operations, defined by a *measurement type* and a *measurement mode*.

The measurement type defines the kind of measurement operation to be performed by the dynamic signal analyzer, and can be one of frequency response or time response. Similarly, the measurement mode describes the type of response to be measured by the signal analyzer from linear response, log response, and swept sine. These experimental settings constitute the bulk of Lab Server-specific experiment information that is hidden from clients.

In addition, the Experiment Routine also specifies a set of measurements that are used to fine-tune the experiment to be run at the lab hardware. These measurements are divided into one of the following different functions:

- **Gain:** this function contains options to set command signals on the system under test.
- **Source:** sets the option for the source level at the signal analyzer.
- **Frequency:** this function contains the options to set the start and stop frequency, as well as the sweep rate for the measurement on the signal analyzer.

Moreover, each of these options can be made visible to clients via the Lab Configuration by setting the *userEditable* attribute to true. Any other options that are not set to true will be hidden from clients by default.

Finally, the Experiment Routine also contains generic information regarding the experiment, such as the lab's internal name, a descriptive label for the experiment, and URL pointers to simple and detailed experiment schematic diagrams. The Experiment Routine may also contain information regarding *offline data*, used to provide access to experimental data that have been previously collected and are available for clients to download at their convenience.

Offline data are characterized by generic experiment information, including a unique lab name, a descriptive label, as well as a URL pointer to the Web location where the experimental results can be retrieved. This generic information describing experiments is also part of the Lab Client-specific data that makes up the Lab Configuration, and described in more detail in Section 3.2.2. An object model representation of the Experiment Routine is given in Figure 3-2.

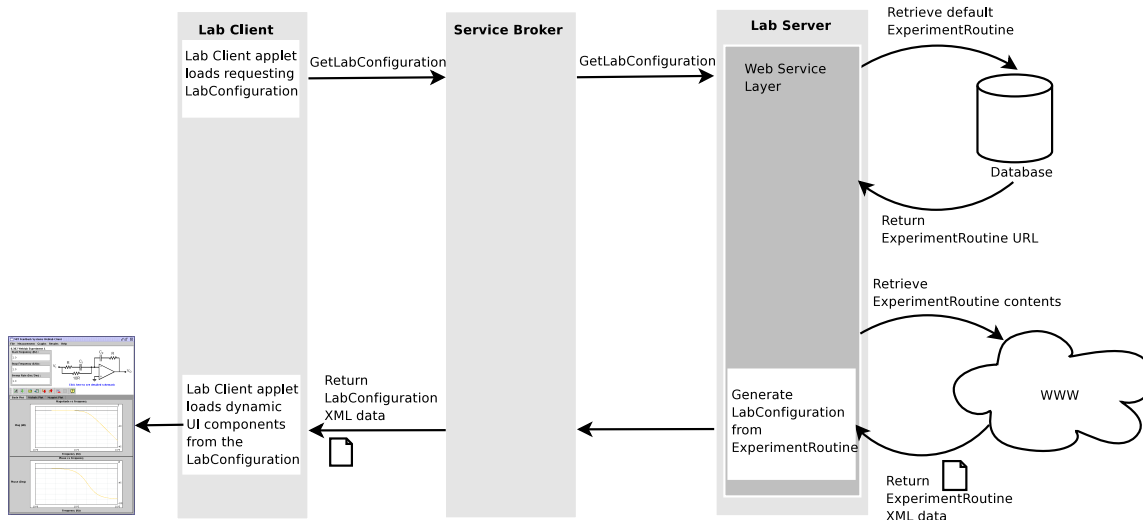


Figure 3-3: Sequence of steps involved in the dynamic generation of the Lab Configuration from the Experiment Routine. First, the Lab Client requests the Lab Configuration from the Lab Server via a `GetLabConfiguration` web service call to the Service Broker. The Lab Server’s web service logic then retrieves the URL for the default Lab Routine from the Lab Server database, and fetches its contents from the web locker. Lastly, it generates the Lab Configuration from the Experiment Routine data as outlined in Figure 3-4, and forwards it to the Lab Client via the Service Broker.

3.2.2 Lab Configuration

As mentioned in Section 3.2.1, the Lab Configuration is dynamically generated at the Lab Server based on the Experiment Routine. Figure 3-3 depicts the complete sequence of steps involved in loading the Lab Configuration at the Lab Client, while Figure 3-4 demonstrates how the Lab Configuration is populated from the Experiment Routine at runtime by the Lab Server.

The Lab Configuration consists of all the Lab Client-specific information regarding an experiment, and contains generic configuration information for the experiment requested by the Lab Client. More specifically, it lists the set of inputs to the experiment that can be modified at the Lab Client, along with their default values and valid ranges. In addition, it provides some generic information concerning the experiment, such as a text description, and URL pointers to the experiment’s simple and detailed schematic diagrams. As with the Experiment Routine, the Lab Configuration may also contain information regarding offline data. Figure 3-5 provides an object model

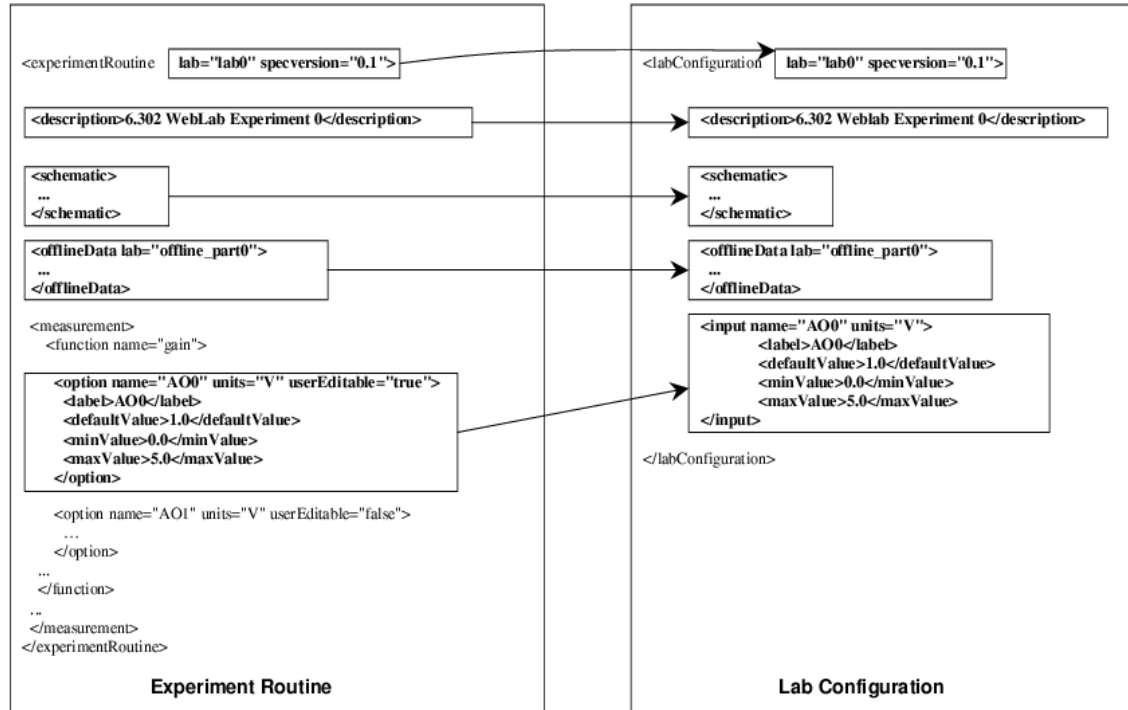


Figure 3-4: Dynamic generation of the Lab Configuration from the Experiment Routine. The Lab Configuration is not a stand-alone specification but is instead completely derived from the Experiment Routine by the Lab Server. It consists of generic lab information such as the lab description, the schematic diagram, offline data and measurement constraints. In addition, it contains all those measurement options marked as *userEditable* in the Experiment Routine.

representation of the Lab Configuration.

3.2.3 Experiment Specification

The Experiment Specification is prepared by the user at the Lab Client end, and represents the parameter values constituting the user's particular run of the experiment. An instance of the Experiment Specification contains a collection of inputs consisting of the parameter name and units, along with the value for the parameter provided by the user. An object model representation of the Experiment Specification is given in Figure 3-6.

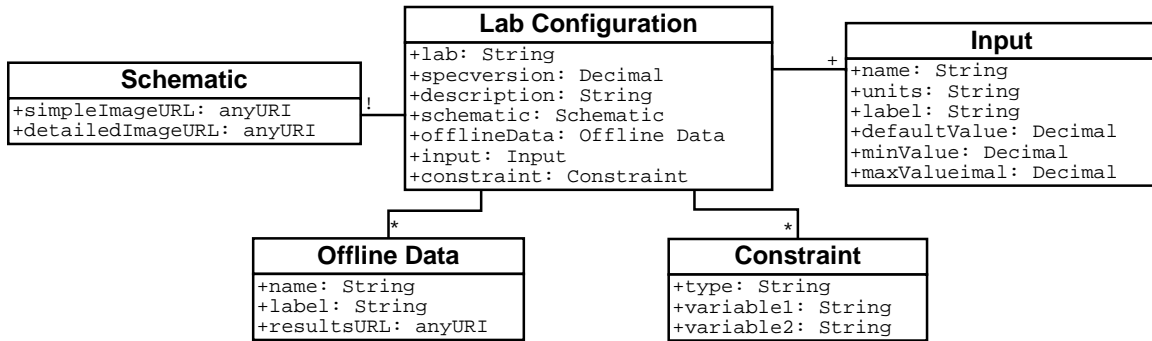


Figure 3-5: An object model representation of the Lab Configuration. The Lab Configuration encapsulates all Lab Client-specific information regarding an experiment. It consists of generic lab information such as the lab description, the schematic diagram, offline data and measurement constraints. In addition, it contains all those measurement options that lab administrators have defined to be user-settable by the Lab Client.

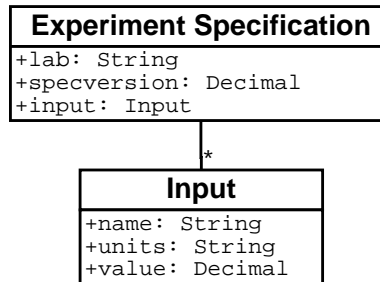


Figure 3-6: An object model representation of the Experiment Specification. The Experiment Specification consists of a number of input parameters making up a user's experiment request to be run at the lab hardware. This specification is usually generated at the client side after users submit their experiments from the Lab Client.

3.2.4 Experiment Result

Once the experiment has completed successfully, the Lab Server produces a list of data vectors containing the measured results and encapsulates them in the Experiment Result. Each data vector is characterized by its result type (usually one of frequency, phase or magnitude), the units for its data values, and the list of comma-separated numeric values measured by the experiment hardware. Figure 3-7 provides an object model representation of the Experiment Result.

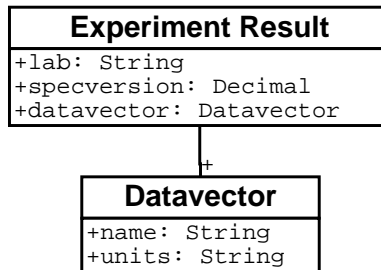


Figure 3-7: An object model representation of the Experiment Result. The Experiment Result encapsulates the experimental data retrieved after successfully completing an experiment request at the Lab Server. It consists of a list of data vectors, usually of types frequency, gain and magnitude. Each of these data vectors is further characterized by its name and units.

Chapter 4

The Feedback Systems Lab Server

4.1 Architecture

The Lab Server implements the iLab framework's Lab Server API using web services under Microsoft's ASP .NET. The web service is implemented in the Visual Basic .NET language on a web server running Internet Information Services (IIS), and can be invoked through a fixed URL reserved for the Feedback Systems WebLab. The Lab Server also runs an SQL database under Microsoft SQL Server 2000. This database is accessed from the web services module for authentication and authorization, logging calls to the web service, retrieval of hardware information relating to the experiment, etc. In addition, the database is used to enqueue experiment requests and to temporarily store any experimental results that were processed at the Lab Server.

The Lab Server also runs an experiment engine on its own thread, separate from the web services module. The experiment engine periodically checks the queue of submitted experiments, and retrieves the job with highest priority or the first one in the queue. It then processes the experiment specification provided by the user at the Lab Client, and sets up the hardware appropriately. This step is accomplished by communicating with an HP 3562A digital signal analyzer over the GPIB bus, and whose probes are attached to the system under test. Moreover, our current setup also makes use of a 20 output data acquisition and control (DAQ) board allowing users to set inputs to the system under test [31]. The DAQ board inputs are set

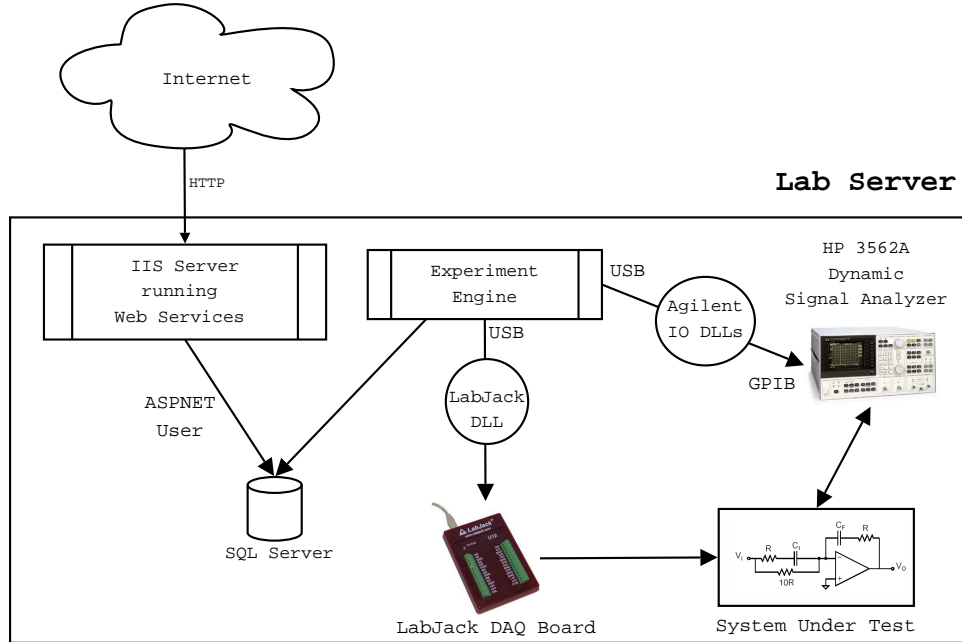


Figure 4-1: Lab Server architecture overview. The Lab Server is composed of three main modules running concurrently. First, the ASP.NET worker process listens for incoming web service method calls and dispatches them to the appropriate class and method. Second, the Lab Server maintains an SQL database storing experiment configurations and requests, user management and lab resource allocation information. Finally, the Experiment Engine executes any pending experiments on the lab hardware. It communicates with the HP 3562A dynamic signal analyzer to set up frequency response measurements, and can also issue various command signals to the system under test via the LabJack DAQ board for further experiment customization.

by the experiment engine, which makes calls to a dynamic link library providing communication with the DAQ board over the USB interface. Figure 4-1 provides a diagrammatic representation of the main Lab Server components.

4.2 ASP.NET Web Services Architectural Overview

The current section provides a brief introduction to the internals of ASP.NET, the web service implementation chosen for our Lab Server.

Like all web services implemented in ASP.NET, our Lab Server web service is accessible over the HTTP transport protocol [32]. When an incoming HTTP message reaches port 80, Internet Information Server (IIS) maps web services asmx extensions

to *Aspnet_isapi.dll* by default [33]. As a result, web service HTTP requests can be forwarded to a separate worker process named *Aspnet_wp.exe*, hosting the common language runtime and the .NET HTTP pipeline.

When a message enters the .NET HTTP pipeline requesting an asmx file, the pipeline calls the `WebServiceHandlerFactory` class to instantiate a new `WebServiceHandler` object to process the request. The `WebServiceHandler` object then opens the physical asmx file to determine the name of the class that contains the web methods composing the web service.

Moreover, the ASP.NET web services model assumes stateless service architecture, and therefore does not inherently correlate multiple calls from the same user. In fact, each time a client invokes an ASP.NET web service, a new object is created to service the request. This object is then destroyed after the method call completes.

Once the asmx handler is called by the .NET HTTP pipeline, it can begin to take care of the XML, XSD, and SOAP processing. The asmx handler carries out the tasks of message dispatching and of mapping XML to objects. In message dispatching, the asmx handler first resolves the referenced .NET class by looking at the `WebService` declaration found in the asmx file. It then reads the SOAP-encoded information in the incoming HTTP message to determine exactly which method to call in the referenced class. Through .NET reflection, it uses the value of the `SOAPAction` header to determine how to dispatch the message to the corresponding web method. Before it can actually invoke the method, however, it needs to map the incoming XML into .NET objects.

The asmx handler maps XML to .NET objects by inspecting the class via reflection in order to determine how to process the incoming XML message. It is the job of the `XmlSerializer` class to perform the automatic mapping between XML and .NET objects in the `System.Xml.Serialization` namespace, as shown in in Figure 4-2. At this point, the web method can be called and its results serialized back into an XML SOAP response by `XmlSerializer`.

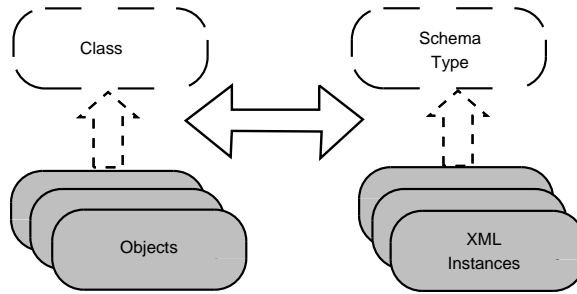


Figure 4-2: Mapping XML to .NET objects [33]. Before the asmx handler can actually invoke the method it needs to map the incoming XML into .NET objects. It is the job of the `XmlSerializer` class to perform the automatic mapping between XML and .NET objects in the `System.Xml.Serialization` namespace.

4.2.1 Security

Since ASP.NET is based on HTTP, it becomes possible to leverage the security features available in IIS to provide strong support for standard HTTP authentication schemes. More specifically, the Feedback Systems Lab Server makes use of IP filtering to deny access to the WebLab service from all hosts except the trusted Service Broker. In addition, all connections between Lab Server and Service Broker can be further secured using the Secure Sockets Layer (SSL) protocol. As a result, any data exchanged between these two hosts does not travel as cleartext but instead is always encrypted prior to being sent over the wire.

Beyond the lab-specific security considerations outlined above, the iLab framework also provides a simple mechanism based on SOAP headers for authentication. When the Lab Server becomes registered at the Service Broker, the latter assigns it a passkey that is bound to that Lab Server's unique ID (see Section C.1). Similarly, when the Lab Server registers the Service Broker, it too assigns the latter a passkey bound to the Service Broker's unique ID. From this point on, all communication between Lab Server and Service Broker includes a SOAP header parameter containing the passkey for the corresponding host. This mechanism, when used along with SSL to encrypt the payload contents, ensures that only registered hosts are able to successfully execute web service methods on either Lab Server or Service Broker.

4.3 Database

In addition to Microsoft's IIS, the Feedback Systems Lab Server runs SQL Server 2000 for experiment data and configuration management. The SQL server can be accessed by both the web services module and the experiment engine at any given time. It consists of SQL tables used to encapsulate Service Broker and Lab Server configurations, experiment records, and group management, as well as of a number of procedures used to manipulate them.

For instance, the `LSSystemConfig` table stores laboratory hardware configuration data, including the bus address for the signal analyzer and its internal VISA name. It also encapsulates all Lab Server specific information, such as the Lab Server's unique ID and the default experiment routine that it should run. In addition, it contains a boolean flag `exp_eng_is_active` which is automatically set to true whenever the experiment engine is running, but which is set to false otherwise. This mechanism allows the Lab Server to gracefully reject experiment requests at those times when the experiment engine is not running.

Similarly, the `Brokers` table is used to store data pertaining to Service Brokers registered at the Lab Server. Therefore, it contains information such as the Service Broker's ID, the Service Broker and Lab Server authentication passkeys, and the URL for the Service Broker-to-Lab Server web service interface. These parameters and their use are described in further detail in Section C.1.2.

Experiment routines are also conveniently abstracted in the SQL database, where they are stored in the `LabRoutines` table. Each experiment routine is uniquely identified by a lab name, and consists of a pointer to the routine's URL and an optional description. This scheme makes it very simple for WebLab administrators to define and support a variety of experiments. Different experimental setups can be written at any time, and registered at the `LabRoutines` table. It then becomes trivial to update the currently running experiment, by simply updating the default experiment routine reference to the new routine in the `LSSystemConfig` table described above. In the case of the 6.302 WebLab, these changes can be easily achieved through a set of

administrative GUI applications that interface to the underlying Lab Server database.

Group management is also managed through the database. Groups are characterized by a unique *group_id*, the name of the corresponding Service Broker group, and the respective Service Broker ID. In addition, a *class_id* is used to manage class-based resource permissions (through the `ClassToResourceMapping` table) and to implement a simple prioritization scheme.

Moreover, the database is the key mechanism used to handle incoming experiment requests. Each experiment job is encapsulated in the `JobRecords` table, where it is tagged with status information such as “QUEUED”, “IN PROGRESS”, “COMPLETED”, or “CANCELLED”. The status information can then be used along with priority and group membership information to implement a first-in first-out queue of jobs ordered by priorities. The group membership information also ensures that job owners have the necessary permissions for the Lab Server to satisfy their experiment requests.

The `JobRecords` table also contains all other ancillary information relating to experiment jobs, such as the elapsed and estimated execution times, times of submission and completion, and queue information. Furthermore, it stores the experiment request itself in the form of the experiment specification sent from the Lab Client, as well as the lab configuration information for the current experiment. It also contains the experiment results in the case that the experiment has been successfully completed at the Lab Server. If an error occurs and the experiment is not executed to completion, an *error_message* is stored as part of the job record.

Finally, the WebLab is dependent on the database for logging and auditing purposes. All incoming and outgoing web service calls are written to the `WebMethodRequestLog` table. These entries can then be examined at any time for security, debugging and WebLab usage evaluation purposes.

A design decision that emerged was the mechanism for communicating between the web service and the database. Initially, the web service accessed SQL Server with explicit credentials, by defining a special user on the Lab Server with permissions to log in to the database. Although this method has the advantage that the database

server need not be present at the local host, it requires that a password be generated and included as plaintext inside the source code. For this reason, the current implementation accesses the database using Windows Integrated Security [34], via the ASPNET user reserved to web applications.

4.4 Experiment Engine

4.4.1 Preparing the Experiment

The Experiment Engine executes on a separate thread from the Lab Server web service and has the task of running experiment jobs submitted to it through the iLab interface. When the web services thread receives an experiment job via the Submit web service method, the experiment request in the form of an Experiment Specification is queued at the Lab Server's database.

When the Experiment Engine is running, it periodically queries the database every five seconds to check for incoming experiment requests from users. If the Lab Server's experiment queue contains any pending experiments, an experiment request is selected from the queue in a first-in first-out basis after ordering the jobs in descending order of priority. Each time an experiment request is selected to be run at the Experiment Engine, it is labeled as "IN PROGRESS" in the Lab Server's database.

The list of experiment operations to be executed on the lab hardware for this experiment job is then loaded at the Experiment Engine. The instructions for running the experiment are obtained by parsing the Experiment Routine for the current lab, whose location is stored on the Lab Server's database. The Experiment Routine can reside on any world-readable web address (e.g. in the 6.302 Athena locker), which makes it easy to maintain and modify remotely by 6.302 course staff and WebLab administrators. Furthermore, it is now possible to easily switch from one experiment to another (e.g. when changing from one lab assignment to the next) by creating more than one Experiment Routine files and simply updating the current lab pointer

on the Lab Server's database.

Once this set of experimental instructions has been parsed and loaded into memory at the Experiment Engine, it is then personalized to execute the user's particular experiment request. The user's particular run of the experiment consists of the previously loaded list of experimental instructions, along with a number of experimental parameters whose values have been set to those provided by the user in the Experiment Specification. This personalization of the experiment instructions takes place by first parsing the Experiment Specification received when the job was submitted, and then completing the experimental instructions with these user-specified parameters and their values.

4.4.2 Running the Experiment

Upon parsing and completing the experimental instructions with the user-provided Experiment Specification, the Experiment Engine is ready to execute the experiment request on the laboratory hardware.

Running an experiment consists of the following steps executed sequentially at the Experiment Engine, and the entire process is depicted as a flowchart in Figure 4-5:

1. **Resetting the hardware:** this step resets the hardware session by pausing any current measurements and resetting the signal analyzer when necessary.
2. **Initializing the experiment:** sets the measurement type and mode on the signal analyzer. The measurement type can be one of *frequency response* or *time response*. The measurement mode is one of *linear response*, *log response*, or *swept sine*.
3. **Preparing the measurement:** in this step, the measurement options for the experiment are set in the order in which they were specified in the Experiment Routine. The experiment options are divided into three main types:
 - **Gain:** this option sets the command signals on the system under test via the LabJack™ digital acquisition (DAQ) board. The Experiment Engine commu-

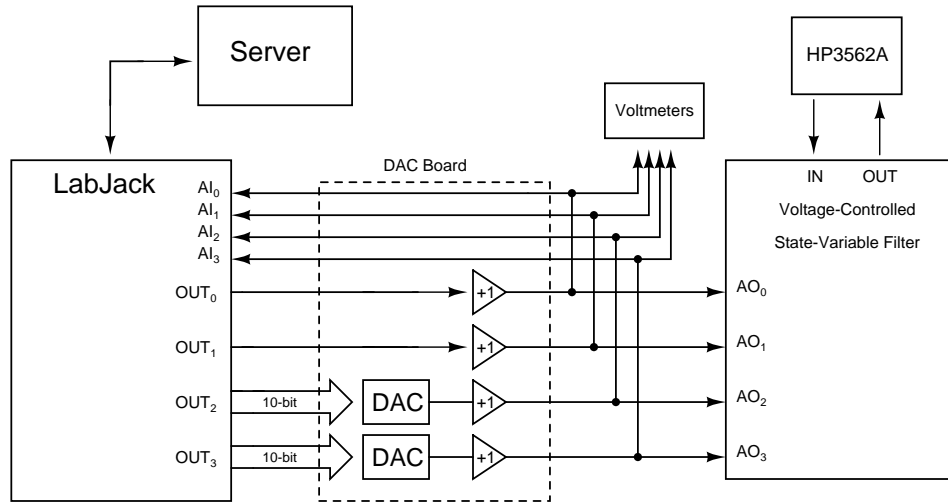


Figure 4-3: Server-side hardware configuration (taken from Isaac Dancy [31]). The four command signals (AO0–AO3) issued to the system under test can be set from the LabJack by the Lab Server via the LabJack. Voltmeters provide administrators with a view of the current command signals. The Lab Server also controls the HP 3562A measurement via the GPIB interface.

nicates with the LabJack over USB by means of a dynamic link library called through a wrapper class.

The first experiment supported by our WebLab consists of a voltage-controlled state-variable filter described in [31]. In this scenario, the LabJack is used to drive two analog 5V voltage signals and 20 lines of 5V TTL-compatible digital logic. These 20 lines are programmed into two 10-bit binary signals in the software, yielding a total of four effective command signals (AO0–AO3) which can be set from the client.

A schematic diagram of the hardware interface is provided in Figure 4-3. In addition, Figure 4-4 depicts the mapping between the LabJack output lines and the analog and digital inputs to the system under test.

- **Source:** used to set the *source level* at the signal analyzer.
- **Frequency:** sets the *start* and *stop frequency*, as well as the *sweep rate* for the measurement on the signal analyzer.

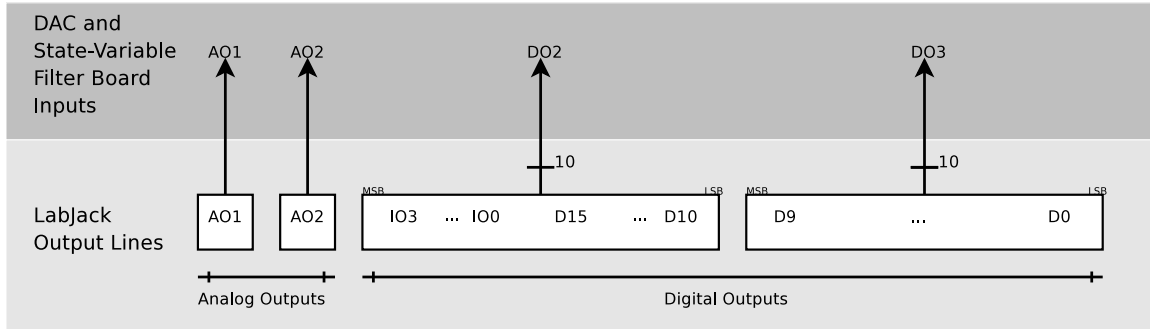


Figure 4-4: Mapping between the LabJack output lines and the analog and digital inputs to the system under test. Since the LabJack is limited to two analog 5-volt voltage signals, we must emulate the last two analog outputs using its 20 lines of 5V TTL-compatible digital logic. These 20 lines are programmed into two 10-bit binary signals in the software, yielding a total of four effective command signals (AO0–AO3) which can be set from the Lab Server.

4. **Running the measurement:** in the final step, the measurement is started at the signal analyzer. While the measurement takes place, the Experiment Engine continually polls the analyzer at one second intervals to check whether the experiment has been completed. Once the measurement is finalized, the experiment results are dumped from the signal analyzer to the Experiment Engine over the GPIB connection. The results are then written to the database and the experiment request is marked as “COMPLETE”.

Finally, event handlers are registered with the Experiment Engine to handle a *load* event when the Experiment Engine is first loaded, and a *close* event when the process is killed. When either of these events is raised, the database is dynamically notified through the event handler, and thus the Experiment Engine status can be updated accordingly. This mechanism prevents users from potentially sending experiment requests at a time when the Experiment Engine is not running at the Lab Server. Instead, they will be notified of the Experiment Engine status by means of a descriptive error message at the Lab Client.

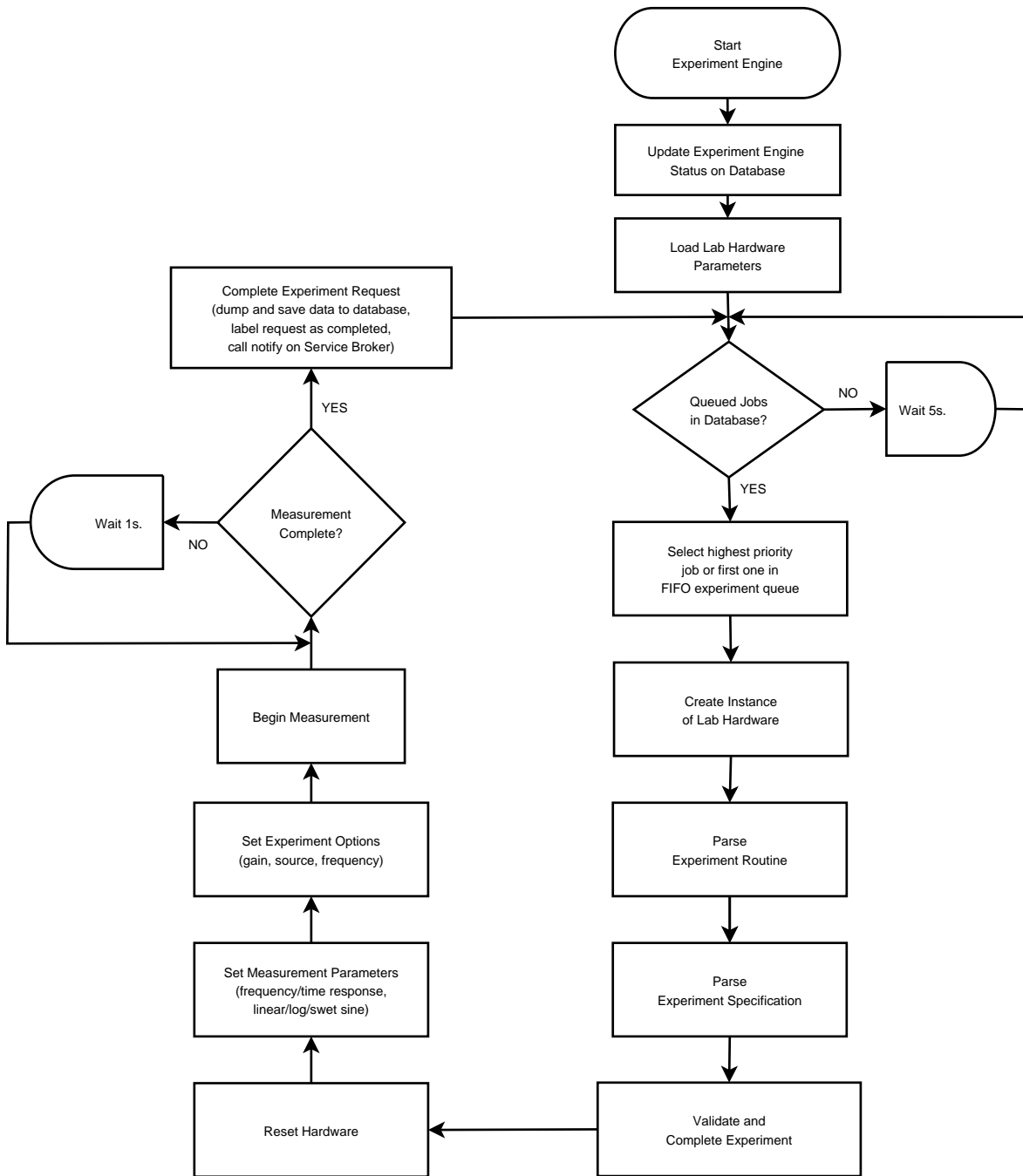


Figure 4-5: Flowchart depicting control flow at the Experiment Engine. Running an experiment at the Experiment Engine consists of four main steps: resetting the hardware, initializing the experiment, preparing the measurement, and running the measurement.

Chapter 5

The Feedback Systems Lab Client

The Lab Client provides the user interface to the Feedback Systems WebLab. First, users must log in to the iLab Service Broker host through their browsers, by supplying their respective username and password. Once they have been authenticated, they are provided with a dynamically generated web page from which to launch the Lab Client. In order to load the Lab Client applet, it is necessary for students to have installed the Java Plugin alongside Java version 1.4.x or greater. When successful, a Lab Client window similar to that in Figure 5-1 will appear before the user. If on the other hand the client system is running an older version of the Java Runtime Engine, then the user is notified of the fact through a warning message. In such cases, the Lab Client automatically closes itself and no communication with the Lab Server takes place.

The Lab Client's graphical user interface can be divided into three main components. First, the upper left side of the screen contains a number of text fields with their corresponding labels. Here, the user will be able to personalize his or her experiment by varying the value of the parameters that will be sent to the Lab Server as part of the Experiment Specification. As described in Section 5.2, the Lab Client applet parses the Lab Configuration it receives from the Lab Server to dynamically construct these editable components.

Second, on the upper right side of the screen, an image representing the schematic diagram of the experiment is displayed. The URL of this graphic is again specified

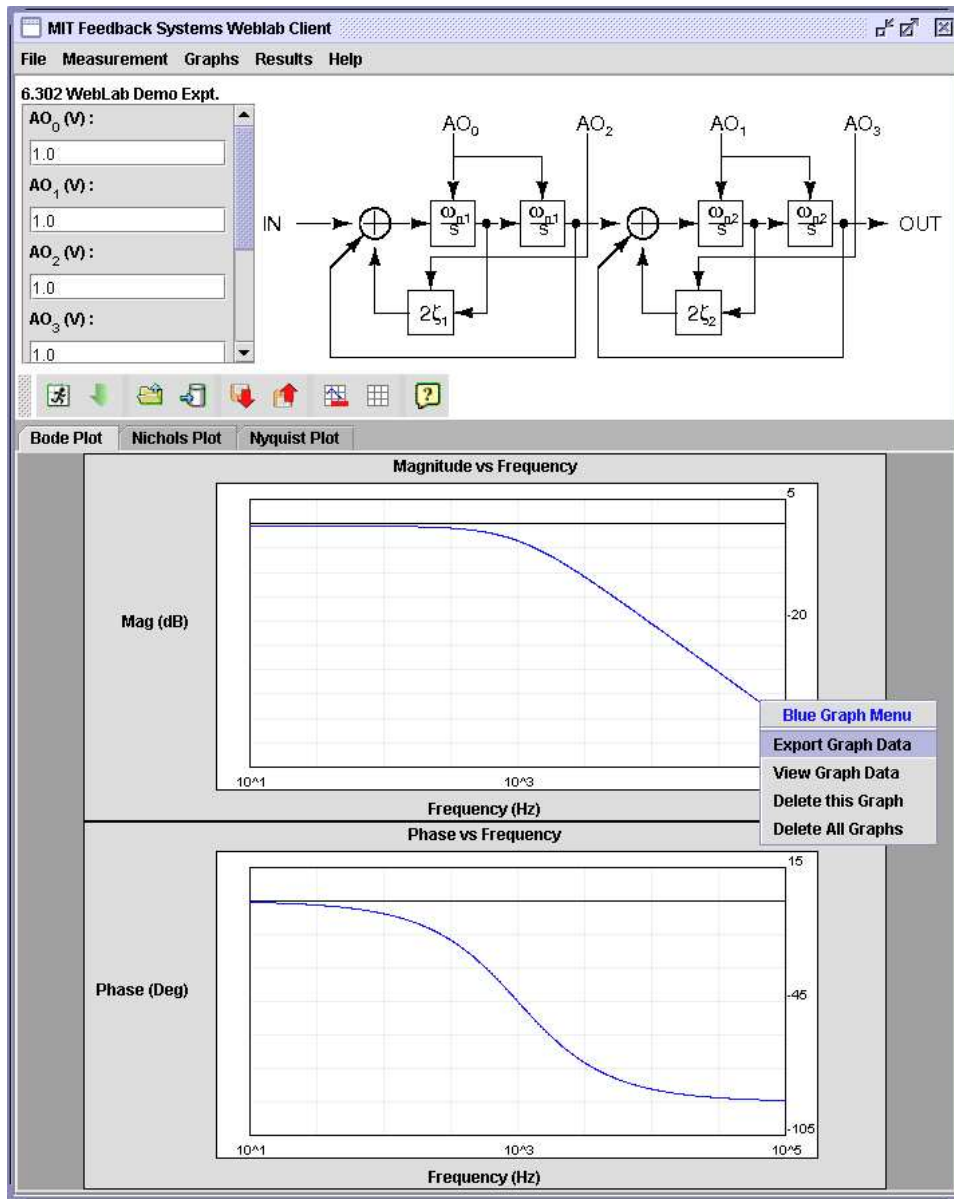


Figure 5-1: Client Applet for the Feedback Systems Weblab. The upper-right side of the applet shows the schematic diagram for the system under test, while at the upper-left corner the user is able to set a number of experiment inputs labeled with the option's name and its units. In the bottom half of the applet, the user can visualize the Bode, Nichols and Nyquist plots for any retrieved experimental data.

in the Lab Configuration and therefore will also be remotely configurable by lab administrators.

Lastly, the lower side of the screen contains a graph panel that displays Bode, Nichols and Nyquist plots of the collected data. The graph panel also allow users to interact directly with the displayed results, by allowing them to select and click on particular results to access a number of options. Some of these options include exporting the experiment results to a number of different data formats, and deleting particular sets of results. The functionality available at the Lab Client is described in further detail in Section 5.3

5.1 Web Services

The Lab Client communicates with the Service Broker by calling methods from the iLab Batched Experiment API of Section 2.3.2. In order to access the SOAP-enabled web methods on the Service Broker, the Lab Client incorporates a lightweight web services implementation known as kSOAP [35]. Although kSOAP was originally devised with embedded systems in mind, its small memory footprint makes it well-suited for building Java applets that support SOAP.

The class `KSoapSBServer` inherits from the `WeblabServer` class, and provides an implementation to call web service methods on the Service Broker host. For instance, when the Lab Client is first loaded, a call from the Lab Client GUI initialization code invokes the web method `GetLabConfiguration`. This method returns the lab configuration containing the laboratory-dependent parameters that will be used to populate the Lab Client's user interface, as described in Section 5.2.

Some of the remaining web service methods called from the Lab Client interface include `Submit`, used to submit Experiment Specifications to the Lab Server; `RetrieveResults`, which queries for experimental results corresponding to a particular run of an experiment; and `GetLabStatus`, which retrieves current status information pertaining to the Lab Server, such as queue length and estimated wait times.

Finally, the Lab Client also employs a number of client-specific API methods that

are not available on the Lab Server. These methods provide a convenient means for Lab Client users to store their Experiment Specifications and other ancillary data on the Service Broker host. These data can then be retrieved and deleted at a later time that is convenient to the user. Some examples of these methods are `SaveClientItem` and `ListClientItems`.

5.2 Dynamic UI Components

One of the design goals of the Lab Client was to provide a user interface that was generic and flexible enough to be easily reconfigurable depending on the current experimental setup in use at the Lab Server site. We were able to achieve this goal without requiring modifications to the Lab Client with each new experiment by making use of dynamic UI components.

The dynamic UI components are completely specified in the Lab Configuration, which the Lab Client obtains from the Lab Server when it first loads on the user's browser. The Lab Configuration, as explained in Section 3.2.1, is derived from the Experiment Routine at the Lab Server. By editing the Experiment Routine, WebLab administrators can easily specify the dynamic components that will form part of the Lab Configuration and which the Lab Client will then use to populate its UI components.

There are three main types of dynamic UI components that form part of the Lab Client. The first and most important type is used by users to specify the value of input parameters for their particular run of the experiment. Inputs are characterized by a label consisting of a descriptive name for the input along with some units. Since inputs always take numerical values, each input has a corresponding default value which is displayed in the text field below the input's label. Furthermore, every input is constrained by the Lab Configuration to be in the range between the specified maximum and minimum values for its input type. Providing an out-of-range value for the input and attempting to submit the corresponding Experiment Specification will result in an error message pointing to the offending input value and restating the

valid input range.

The second group of UI components encompasses ancillary information relating to the experiment. Examples of these are the experiment schematic diagram displayed on the top right area of the Lab Client. In addition, clicking on this image causes a browser window to open and load a more detailed schematic diagram for the current experiment. Another example is the location of the experiment handout. Clicking on the *Experiment Handout* menu item under *Help* will similarly open a browser pointing to the corresponding experiment handout that was provided for the given experiment. Yet another example is the descriptive label for the experiment, which appears immediately above the input panel on the top left corner of the Lab Client.

The last type of UI components is used to describe so-called *offline data*. Offline data are used to provide easy access by Lab Client users to previously obtained experimental data. This feature could prove useful when comparing measured experimental data with previously computed theoretical values derived from a mathematical formula or generated by a simulation engine. Another possible use involves the dissemination of experimental data that were obtained beforehand by the course staff, and for which it would be infeasible to run the corresponding experiment by each and every student in a large class (e.g. the temperature control experiment in feedback systems).

Offline data are described by a label and a results URL, which the Lab Client accesses to retrieve and display the corresponding experimental data in much the same way it would do for live data obtained directly from the Lab Server. In fact, from the user's perspective measured data processed at the Lab Server site and offline data retrieved from a web location are indistinguishable. Even though the manner in which these two kinds of data are obtained is very different, the operations that users can perform on them are identical (e.g. exporting the data, visualizing the plots, etc.).

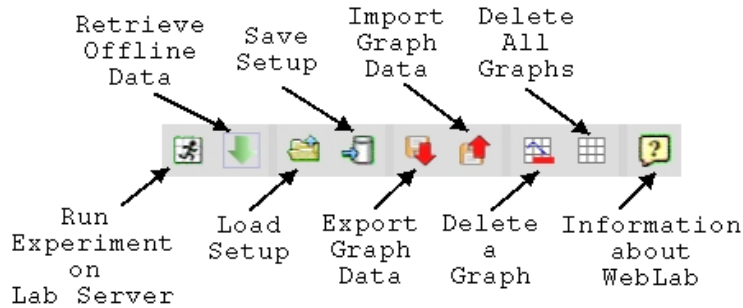


Figure 5-2: Toolbar for the Lab Client. The toolbar provides direct access to the most useful Lab Client functionality. The same functionality is also available through the applet menus.

5.3 Functionality







The most important functions available on the Lab Client are easily accessible through the toolbar found below the input area. All of these functions can also be accessed indirectly through the Lab Client menus, along with extra functionality not available via the toolbar.

As shown in Figure 5-2, the toolbar allows students to run experiments on the Lab Server, retrieve offline data, load and save setups, import and export graph data, and delete graphs, as well as access the WebLab manual.




The Lab Client can also be navigated by means of its five main menus: *File*, *Measurement*, *Graphs*, *Results*, and *Help*. A complete list of the different functions that can be accessed through these menus is provided in Table 5.1.

Finally, a detailed account explaining how to operate the Lab Client can also be found online at the Lab Client manual pages [36]. The main applet structure and web service classes of our Lab Client borrowed heavily from David Zych's Graphical Applet [37] used in the Microelectronics WebLab. Much of the functionality described in this section was developed and integrated into the Lab Client by Sriganesh Lokanathan [38]. Furthermore, the core graphing package used to generate Bode, Nyquist and Nichols plots was adapted from Brian Williams' Pole Zero applet [39].

Table 5.1: Menus and Icons used in the Lab Client

Menu Item	Icon	Description
File		
Load Setup		Loads a saved Experiment Specification locally or from the server.
Delete Setup		Deletes a saved Experiment Specification from the server.
Save Setup		Saves current Experiment Specification locally or at the server.
Exit		Exits Lab Client
Measurement		
Run Expt. on Lab Server		Submits the current Experiment Specification to the Lab Server.
Retrieve Offline Data		Retrieves any offline data available for download.
Graphs		
Always Replace		Sets the graphing mode to always replace the current plots by the newly generated ones.
Always Add		Sets the graphing mode to always add the newly generated plots to the current ones.
Delete a Graph		Deletes the selected graph (and its associated data) from the Lab Client.
Delete all Graphs		Clears all graphs and experimental data contained at the Lab Client.
Save Graph Image		Allows each of the Bode, Nyquist and Nichols plots to be saved to a png graphics file.

(continued)

Menu Item	Icon	Description
Results		
View Graph Data		Opens a new window displaying the measured experimental data in text form.
Export Graph Data		Allows the measured experimental data to be saved as either a Matlab or comma-separated value compatible file.
Import Graph Data		Loads experimental data saved to a Matlab or comma-separated value compatible file into the Lab Client. The Bode, Nichols and Nyquist plots are updated accordingly.
Help		
Experiment Handout		Opens a new browser window pointing to the laboratory handout for the current experiment.
Lab Server Status		Retrieves status information from the Lab Server, such as a wait estimate for submitted jobs and current queue length.
Lab Client Manual		Opens a new browser window pointing to the Lab Client user's manual.
About the WebLab Client		Provides general version and build information for the Lab Client.

Chapter 6

Field Trial for the Feedback Systems WebLab

6.1 Student Experience

The trial of the Feedback Systems WebLab and the iLab shared architecture was conducted in the Fall of 2004 in 6.302 Feedback Systems, a senior-level subject in the Department of Electrical Engineering and Computer Science at MIT with about 60 students. Students were issued two laboratory assignments using the WebLab. In the first assignment, students familiarized themselves with the system by running a simple predefined experiment on the Lab Server and then saving the results at the Lab Client. In the second assignment, students mathematically analyzed the response of a voltage-controlled state-variable filter and compared their calculated responses with those obtained from the actual filter via the WebLab interface. Successful completion of this assignment required at least four jobs to be submitted by each student in the class. The actual lab assignment is included in Appendix E.

During the first assignment, the system worked nearly flawlessly with no problems experienced at the Lab Server. A few minor issues emerged with students accessing the wrong Service Broker login URL, and with those who did not have the appropriate Java run-time/plugin configuration on their system, but these were all eventually resolved. In addition, an MIT AFS outage occurred at around 5 am on October 26th,

time during which a few students were unable to access the WebLab service or load the Lab Client. However, WebLab usage returned to normal once the AFS outage was eventually resolved. The second assignment was a great success, and no bug reports or problems were reported by the students.

6.2 Statistics

Statistics were collected for the WebLab usage during the two lab assignments issued in 6.302 of Fall 2004. As can be seen in Figure 6-1, the queue of the system never held more than a single job for the first assignment. For the second assignment, however, the load on the system was considerably higher and a maximum queue length of three jobs was reported on the night before the assignment was due. At that same time, the system reached its highest rate of incoming experiment requests, processing a total of 38 jobs per hour (see Figure 6-3). Overall, 85 percent of the jobs received at the Lab Server took at most 60 seconds in total to complete, and 96 percent of all jobs were executed in less than 60 seconds by the Experiment Engine (once selected from the queue of experiment jobs). A distribution of the job execution and completion times for the second assignment is shown in Figure 6-5.

Figures 6-2 and 6-4 show the distribution of experiment jobs in time. These plots show how the time taken for an average job to be executed by the Experiment Engine remained relatively constant at around 60 seconds. On the other hand, the total time required for a particular job to be processed depended heavily on the queue size at submit time, and ranged from a minimum of 55 seconds (when the queue was empty) to a maximum of 208 seconds (when the queue was at its greatest length). Additional summary statistics for the Lab Server execution records are provided in Tables 6.1 and 6.2.

Table 6.1: Descriptive statistics of the Lab Server execution records for the first WebLab assignment. *Job Elapsed* is the total time elapsed since an experiment request is first received at the Lab Server, until it has finished running on the lab hardware and the measured data have been received at the Lab Server. *Execution Elapsed* refers to the time taken for an experiment to be executed at the lab hardware, starting from the time when the job is selected to be run from the queue of experiment jobs.

Job Elapsed		Execution Elapsed	
Mean	58.3	Mean	57.3
Median	56	Median	55
Mode	55	Mode	55
Std. Dev.	11.1	Std Dev.	10.3
Variance	123.7	Variance	106.8
Range	80	Range	80
Minimum	55	Minimum	54
Maximum	135	Maximum	134
Sum	4901	Sum	4809
Count	84	Count	84

Table 6.2: Descriptive statistics of the Lab Server execution records for the second WebLab assignment.

Job Elapsed		Execution Elapsed	
Mean	63.5	Mean	56.9
Median	56	Median	56
Mode	56	Mode	56
Std. Dev.	22.0	Std Dev.	7.3
Variance	483.7	Variance	52.8
Range	153	Range	72
Minimum	55	Minimum	54
Maximum	208	Maximum	126
Sum	23116	Sum	20727
Count	364	Count	364

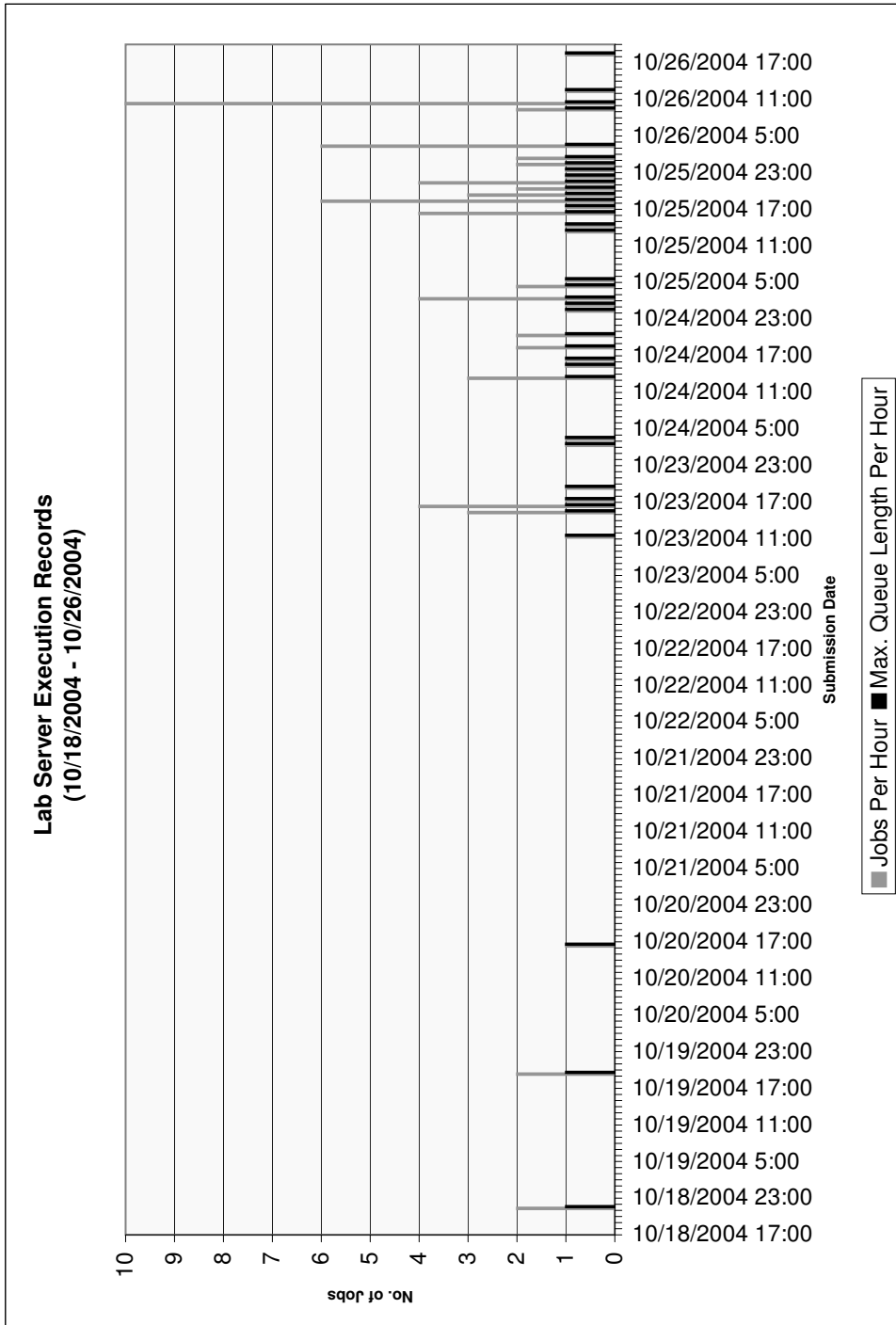


Figure 6-1-1: Histogram of experiment queuing and execution during the first WebLab assignment for 6.302, Fall 2004.

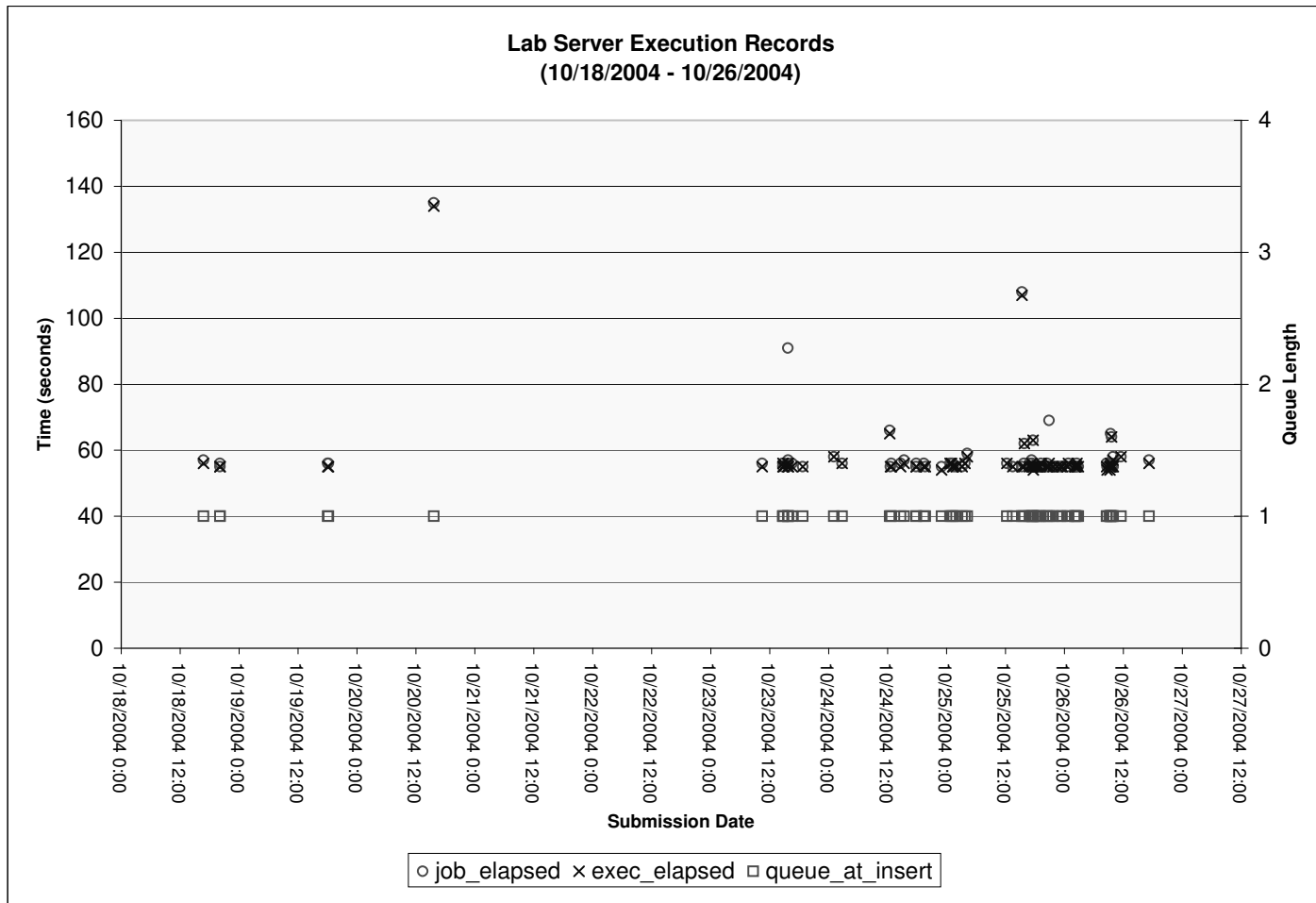


Figure 6-2: Distribution of experiment jobs for the first WebLab assignment for 6.302, Fall 2004. The scatter graph depicts the total time elapsed until job completion (*job_elapsed*), the queue length when the job was received at the Lab Server (*queue_at_insert*), and the time taken for the job to be executed by the Experiment Engine (*exec_elapsed*) for individual student experiment requests.

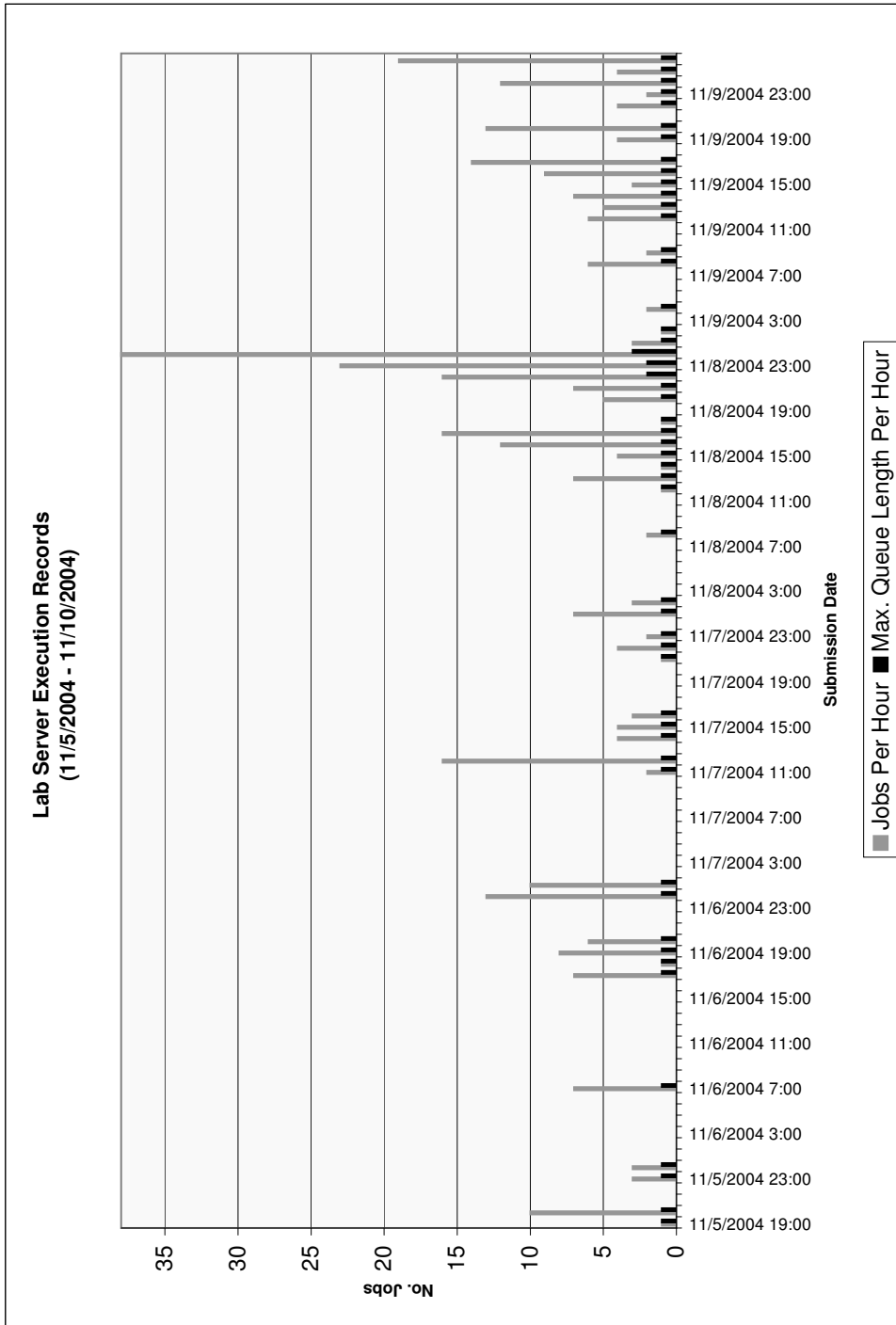


Figure 6-3: Histogram of experiment queuing and execution during the second WebLab assignment for 6.302, Fall 2004.

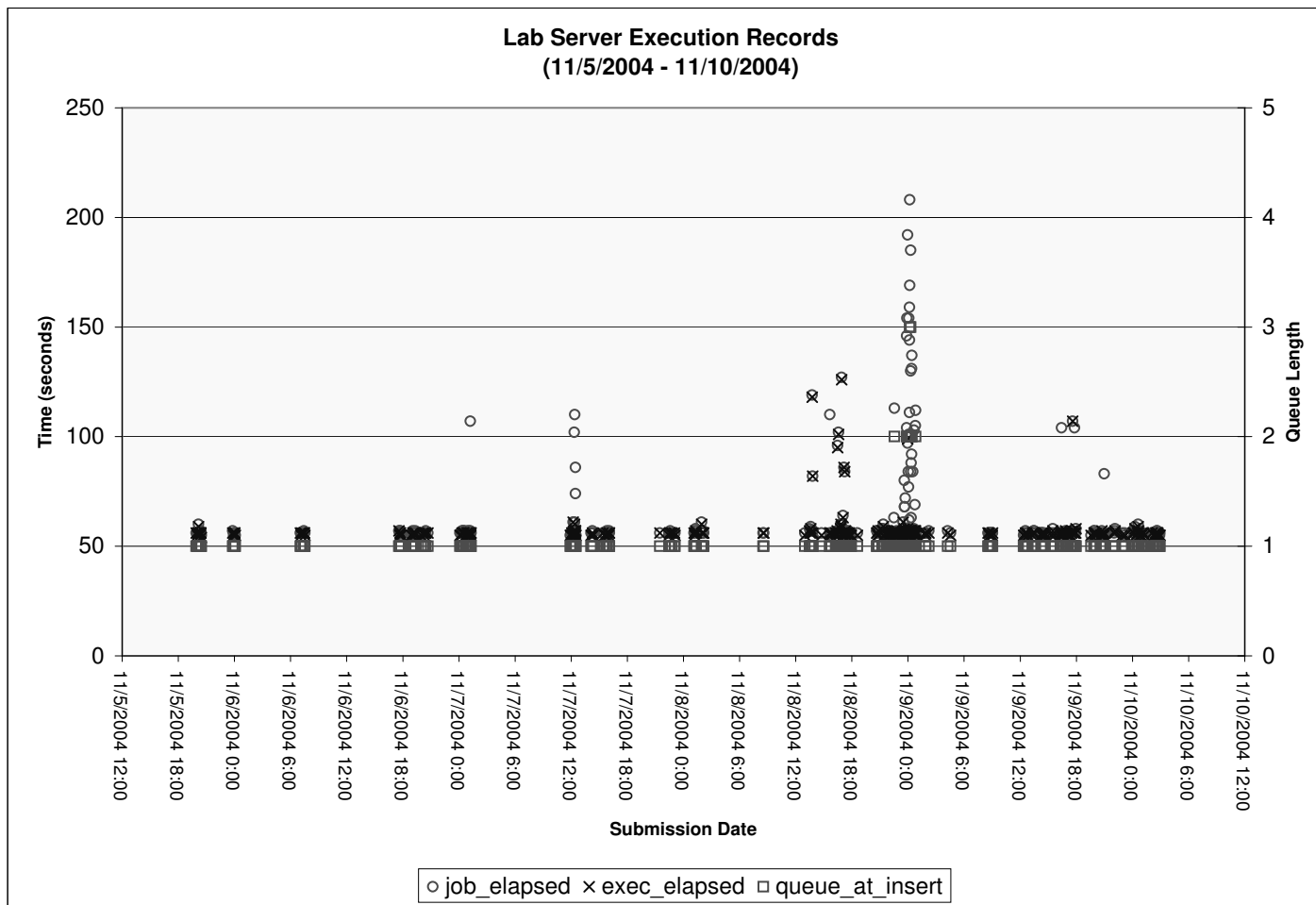


Figure 6-4: Distribution of experiment jobs for the second WebLab assignment for 6.302, Fall 2004. The scatter graph depicts the total time elapsed until job completion (*job_elapsed*), the queue length when the job was received at the Lab Server (*queue_at_insert*), and the time taken for the job to be executed by the Experiment Engine (*exec_elapsed*) for individual student experiment requests.

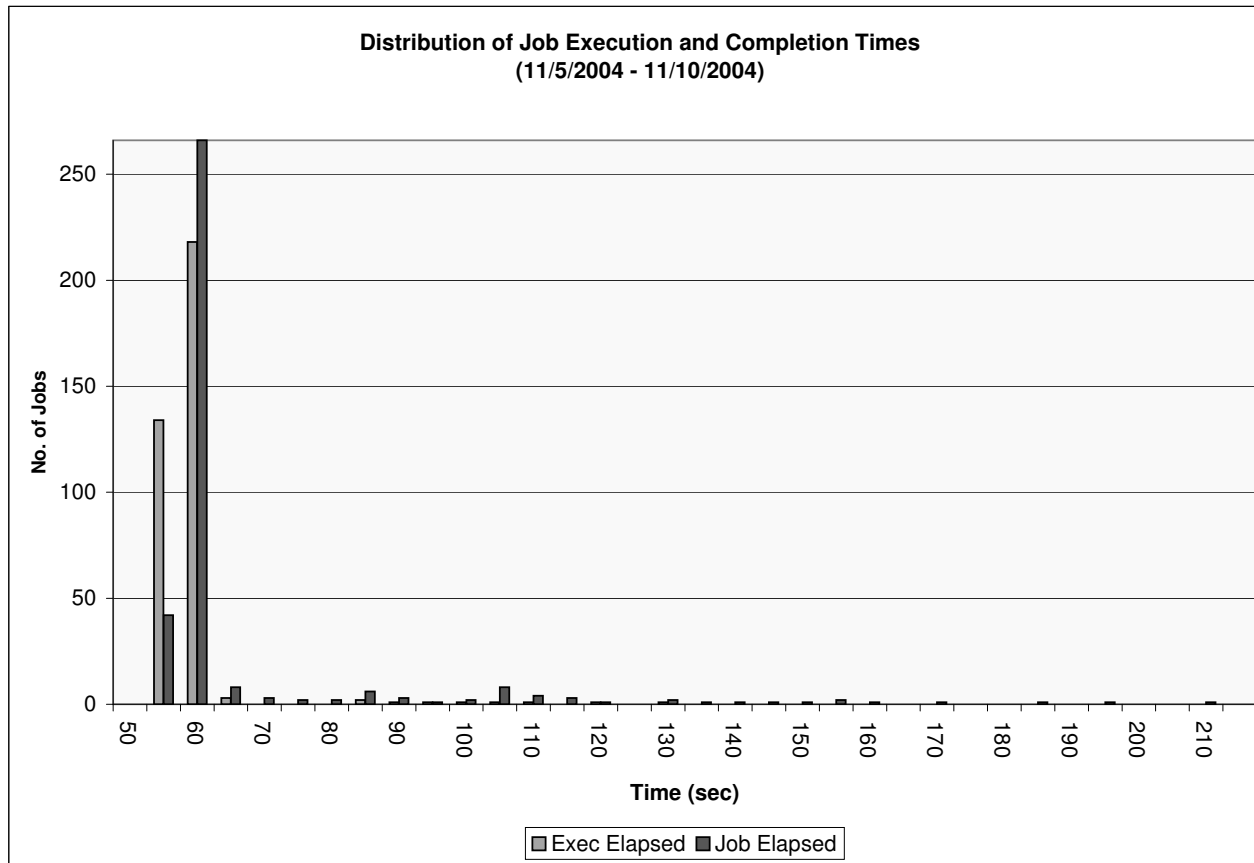


Figure 6-5: Distribution of execution and completion times for the second WebLab assignment for 6.302, Fall 2004. A number of jobs required more than 140 seconds in total to complete, corresponding to those times when prior experiment requests had already been queued at the Lab Server. More interestingly, about 3 percent of jobs took between 70 and 130 seconds to *execute* at the Experiment Engine, possibly due to networking and hardware limitations. Unresponsiveness of the MIT AFS course locker can result in longer wait times when retrieving experiment routines over HTTP prior to execution. Moreover, the signal analyzer periodically undergoes calibration, thus becoming unable to process experiment requests.

6.3 Student Response to the WebLab

Shortly after completion of the two WebLab assignments, students were asked to complete a five minute written survey regarding their experiences with the system. A total of 29 students completed the survey, out of a class size of approximately 60 students. A copy of this survey can be found in Appendix F.

The first set of questions required students to rank the WebLab's ease of use, responsiveness, effectiveness in teaching the course material, and effectiveness experimenting on a real system. The results provided in Figure 6-6 indicate that students had a very positive experience with the ease of use and responsiveness of the WebLab. 90 percent of the students surveyed ranked the ease of use and responsiveness of the system as acceptable, good or very good. However, the results also show the students' discontent with the pedagogical effectiveness of the WebLab. For example, only 60 percent of those surveyed regarded the WebLab as an effective tool for experimenting with a real system, and a disappointing 50 percent considered it to be an effective aid in teaching the course material.

The second set of questions gave students free reign to comment on the best and worst aspects of the WebLab. Among the wide variety of responses, the student consensus was that the system was very easy to use, convenient, provided an intuitive interface to a real system, and responded quickly to user interaction. In students' words, these attributes made it easy to "quickly try out different things and see the result without the headache of setting up and troubleshooting lab equipment". On the other hand, a large percentage of the students felt disappointed with the lab assignment, which they described as "pointless" and "badly parametrized". In addition, a few students mentioned that the WebLab felt somewhat artificial, with "an interface basically indistinguishable from a good Matlab script". It is interesting to note that, in general, students who realized they were experimenting on a real system tended to rate the WebLab assignment high on pedagogical effectiveness. On the other hand, those students who were not directly aware of this fact or who commented on its "artificial feel" were much more likely to rate it low on effectiveness.

Finally, students were asked their thoughts on incorporating WebLab assignments in future versions of 6.302. About 55 percent of those surveyed were in favor of using a combination of WebLab and traditional in-lab assignments, while an additional 10 percent would like to see all lab assignments replaced by WebLab assignments. The remaining 33 percent of students surveyed would prefer to have only traditional in-lab assignments in future versions of the course.

6.4 Lessons Learned

From our experience in the course setting and judging from the students' response, we are able to make two main observations. First, we found that the students' perception of the WebLab's pedagogical effectiveness is strongly related to the degree to which the system under test feels like a real system to students interacting with it. Second, as WebLab assignments become an integral part of the Feedback Systems curriculum, their success will be partly determined by how well they are integrated with the rest of the course material and problem sets.

The realization that the WebLab assignments may not appear very real to some students was highlighted after some discussion with the 6.302 teaching assistants [40]. As a result, a number of students believed they were interacting with a Matlab script instead of a real system. Consequently, these students were more likely to describe the WebLab assignments as pointless and tedious than those who were conscious they were interfacing to real hardware.

The second type of criticism referred to the actual contents of the WebLab assignment itself. A number of students felt that the exercise was badly parametrized, making it cumbersome to calculate inputs to the system. Moreover, some felt that the lab assignment consisted of tedious and mechanical steps that detracted from the learning experience and often lead to frustration. Some examples included the conversion of frequency response data from Hertz to radians per second when comparing real and experimental data, as well as exporting experimental data at the Lab Client in order to import and manipulate it in Matlab [40]. Still other students felt

that they were simply testing out or debugging the WebLab, as opposed to carrying out meaningful lab work. As the first real deploy and trial of the Feedback Systems WebLab, we recognize the validity of these criticisms and acknowledge that much further work remains to be done in improving both the content and presentation of future WebLab assignments.

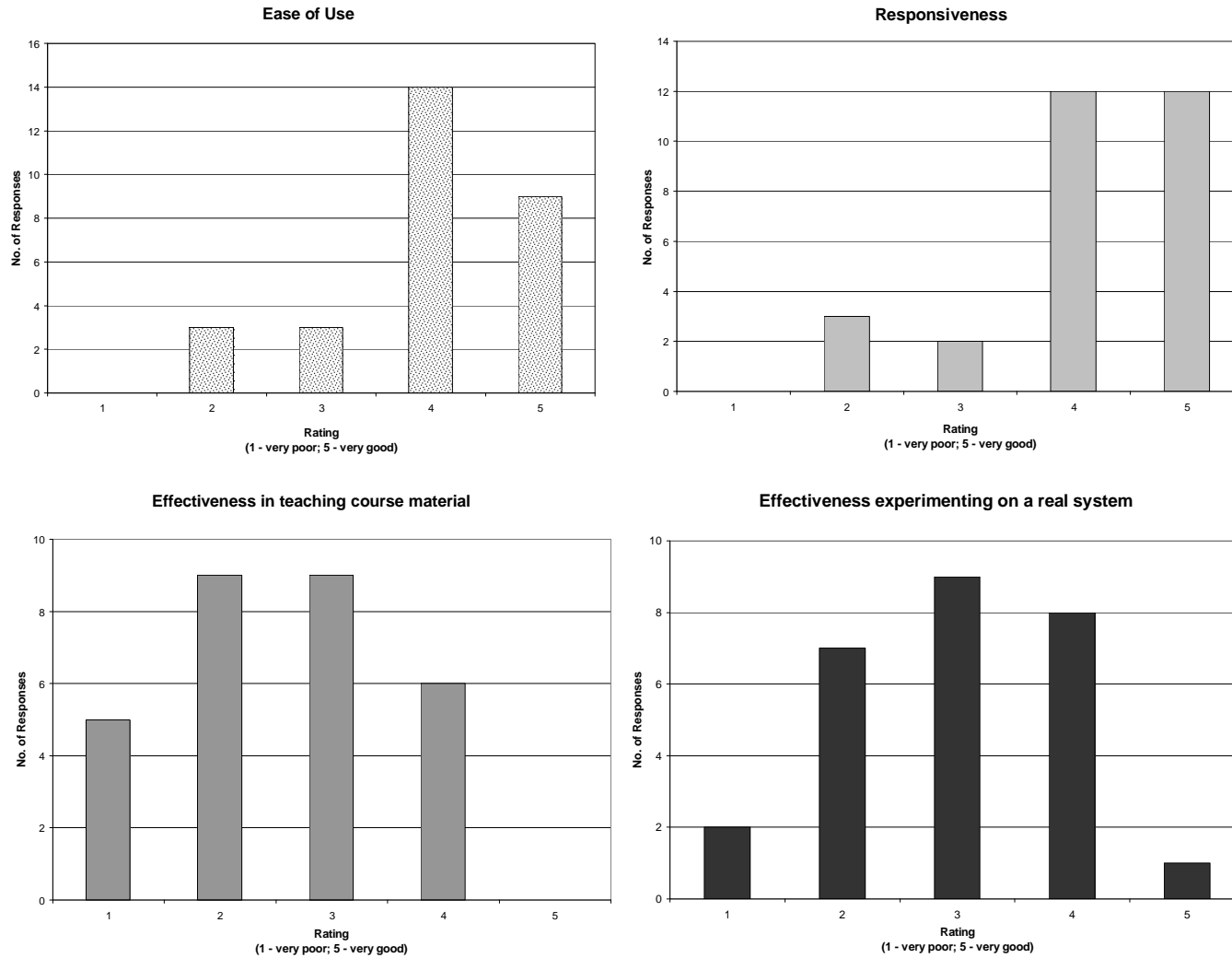


Figure 6-6: Student response to the 6.302 WebLab assignment (Fall 2004). Students were asked to rank the WebLab’s ease of use, responsiveness, effectiveness in teaching the course material, and effectiveness experimenting on a real system.

Chapter 7

Conclusion and Future Work

7.1 6.302 and the Feedback Systems WebLab

In the light of our experience with the WebLab in 6.302 of Fall 2004, we are able to draw three major conclusions.

First, the Feedback Systems WebLab has been shown to reliably and efficiently support a middle-sized class such as 6.302 for web-based laboratory assignments. A total of 60 students submitted some 360 jobs over the course of their weekly lab assignment. Out of these jobs, 96 percent were executed in less than 60 seconds at the Experiment Engine, and 85 percent of the experiment requests were completed in at most 60 seconds (see Section 6.2). Moreover, the student response to the WebLab was also very positive, with 90 percent of students surveyed ranking the ease of use and responsiveness of the system as acceptable, good or very good (Section 6.3).

Second, the students' perception of the WebLab's pedagogical effectiveness is strongly related to the degree to which the system under test feels like a real system to students interacting with it. Our survey found that students who were not conscious they were interfacing with real hardware were more likely to describe the WebLab assignments as pointless and tedious (see Section 6.4).

Upon discussion with the teaching assistants, we believe that students should be made aware of the physical reality of the underlying system in order to further motivate the WebLab assignments [40]. Students could be made cognizant that they

are interacting with a real system, and not simply a contrived Matlab script, by forcing the system into saturation. They could also be provided with opportunities to explore how the real system analyzed via the WebLab compares to an ideal system treated mathematically. Yet another possibility would involve incorporating a real-time view of the system under test, perhaps via a webcam whose live image updates every few seconds at the Lab Client.

Lastly, the pedagogical effectiveness of the WebLab is only as good as the assignments that are supported by it. The current WebLab assignment served its purpose well as a proof of concept, yet as acknowledged by the students is in need of further refinement to bring home the full range of pedagogical opportunities available to remote web-based experimentation. For example, only 60 percent of those surveyed regarded the WebLab as an effective tool for experimenting with a real system, and a disappointing 50 percent considered it to be an effective aid in teaching the course material (Section 6.3).

In particular, future assignments could involve less tedious and mechanical manipulation of data, and instead focus more on analyzing and comparing the real data with theoretical models of the system under test. In addition, they could incorporate some additional Lab Client functionality, such as the possibility of plotting more than one sets of data (e.g. experimental and theoretical data). Future labs could also involve Nichols and Nyquist plots, thus allowing students to explore how these plots can be generated from real data and how they can be used to analyze a real system.

7.2 The Feedback Systems WebLab beyond 6.302

The Feedback Systems WebLab is capable of measuring the response of a number of different systems. This characteristic makes it suitable to explore and study system responses in a wide variety of engineering disciplines. Beyond the coursework of 6.302, our WebLab could also be incorporated into introductory signals and systems courses (e.g. MIT's 6.003, and Unified Engineering in the Aeronautics and Astronautics department).

In such a setting, it could be used by students to investigate the effects of pole and zero locations on system response. It could also allow them to explore second-order system concepts, such as the significance of the damping ratio ζ , and how it affects the system's behavior in the frequency domain. Moreover, the HP 3562A dynamic signal analyzer is also capable of measuring the time response of a system. Such functionality thus opens the door to web-based analysis of time-domain behavior, and could be used in the studies of step responses for a variety of second-order systems.

Lastly, a number of physical systems can be modeled by linear constant-coefficient differential equations. Examples of these include models of heat flow, as well as a number of mechanical systems. Moreover, these can often be represented and implemented by combining first-order and second-order systems in cascade or parallel arrangements. Thus, lab hardware that implements higher-order systems for electrical engineering courses can in principle be leveraged in the study of different fields. In the particular case of feedback systems, adding a third and higher order stages to the system would enable students to experiment with compensation and how different compensation schemes affect system behavior.

7.3 Future Prospects for the Feedback Systems WebLab

A number of improvements have been planned for the next version of the Feedback Systems WebLab. In the short term, we would like the Lab Client to incorporate a live image of the system under test obtained from a webcam at the laboratory site. We are also contemplating converting portions of the current 6.302 thermal-control system lab into a WebLab assignment.

In the longer term, much work remains to be done on creating WebLab assignments of additional pedagogical value that also integrate well with the rest of the curriculum. As they become available, the WebLab may begin to make the transition from being a curious and exciting piece of technology to becoming an indispensable

tool in the learning and applying of engineering principles.

To this end, we also envision incorporating a Feedback Systems Simulation WebLab with much of the same flavor as Adrian Solis' Device Simulation WebLab for Microelectronic Devices [41]. The simulation functionality could then be further integrated into our current WebLab, thus providing students with opportunities to simultaneously compare theoretically derived and measured experimental data for the same lab assignment. We believe this key enhancement would be of great pedagogical value to the teaching of control systems.

Finally, as noted by a wise 6.302 student, it is “hands-on experiments that matter most when you sit down in a lab on the job”. The intended role of the Feedback Systems WebLab as a *complement*, not as a substitute, of traditional laboratory work acknowledges this reality. The WebLab can thus help students save time and effort on routine hardware setup and configuration, allowing them to focus on the more important areas of problem analysis and system design that constitute the high value-added skills of the modern engineer.

Appendix A

XML Schema Definitions

A.1 Experiment Routine

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns="http://i-lab.mit.edu" elementFormDefault="qualified"
  targetNamespace="http://i-lab.mit.edu"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple elements -->
  <xs:element name="description" type="xs:string" />
  <xs:element name="selectMeas" type="xs:string" />
  <xs:element name="measMode" type="xs:string" />
  <xs:element name="label" type="xs:string" />
  <xs:element name="defaultValue" type="xs:decimal" />
  <xs:element name="minValue" type="xs:decimal" />
  <xs:element name="maxValue" type="xs:decimal" />
  <xs:element name="simpleImageURL" type="xs:anyURI" />
  <xs:element name="detailedImageURL" type="xs:anyURI" />
  <xs:element name="resultsURL" type="xs:anyURI" />
  <xs:element name="variable1" type="xs:string" />
  <xs:element name="variable2" type="xs:string" />

  <!-- definition of complex elements -->

  <!-- schematic element -->
  <xs:element name="schematic">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="simpleImageURL" />
        <xs:element ref="detailedImageURL" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```

<!-- offlineData element -->
<xs:element name="offlineData">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="label" />
      <xs:element ref="resultsURL" />
    </xs:sequence>
    <xs:attribute name="lab" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<!-- initialization element -->
<xs:element name="initialization">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="selectMeas" minOccurs="1" maxOccurs="1" />
      <xs:element name="measMode" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- option element -->
<xs:element name="option">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="label" minOccurs="1" maxOccurs="1" />
      <xs:element ref="defaultValue" minOccurs="1" maxOccurs="1" />
      <xs:element ref="minValue" minOccurs="1" maxOccurs="1" />
      <xs:element ref="maxValue" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="units" type="xs:string" use="required" />
    <xs:attribute name="userEditable" type="xs:boolean" use="required" />
  </xs:complexType>
</xs:element>

<!-- function element -->
<xs:element name="function" >
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="option" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<!-- measurement element -->
<xs:element name="measurement">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="function" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```



```

<!-- constraint element -->
<xs:element name="constraint">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="variable1" minOccurs="1" maxOccurs="1" />
      <xs:element ref="variable2" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<!-- experimentRoutine element -->
<xs:element name="experimentRoutine">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="description" />
      <xs:element ref="schematic" minOccurs="1" maxOccurs="1" />
      <xs:element ref="offlineData" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="initialization" minOccurs="1" maxOccurs="1" />
      <xs:element ref="measurement" minOccurs="1" maxOccurs="1" />
      <xs:element ref="constraint" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="lab" type="xs:string" use="required" />
    <xs:attribute name="specversion" type="xs:decimal" use="required" />
  </xs:complexType>
</xs:element>

</xs:schema>

```

A.2 Lab Configuration

```

<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns="http://i-lab.mit.edu" elementFormDefault="qualified"
  targetNamespace="http://i-lab.mit.edu"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple elements -->
  <xs:element name="description" type="xs:string" />
  <xs:element name="label" type="xs:string" />
  <xs:element name="defaultValue" type="xs:decimal" />
  <xs:element name="minValue" type="xs:decimal" />
  <xs:element name="maxValue" type="xs:decimal" />
  <xs:element name="simpleImageURL" type="xs:anyURI" />
  <xs:element name="detailedImageURL" type="xs:anyURI" />
  <xs:element name="resultsURL" type="xs:anyURI" />

```

```

<!-- definition of complex elements -->

<!-- schematic element -->
<xs:element name="schematic">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="simpleImageURL" />
      <xs:element ref="detailedImageURL" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- offlineData element -->
<xs:element name="offlineData">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="label" />
      <xs:element ref="resultsURL" />
    </xs:sequence>
    <xs:attribute name="lab" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<!-- input element -->
<xs:element name="input">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="label" minOccurs="1" maxOccurs="1" />
      <xs:element ref="defaultValue" minOccurs="1" maxOccurs="1" />
      <xs:element ref="minValue" minOccurs="1" maxOccurs="1" />
      <xs:element ref="maxValue" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="units" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<!-- labConfiguration element -->
<xs:element name="labConfiguration">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="description" />
      <xs:element ref="schematic" minOccurs="1" maxOccurs="1" />
      <xs:element ref="offlineData" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="input" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="lab" type="xs:string" use="required" />
    <xs:attribute name="specversion" type="xs:decimal" use="required" />
  </xs:complexType>
</xs:element>

</xs:schema>

```

A.3 Experiment Specification

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns="http://i-lab.mit.edu" elementFormDefault="qualified"
  targetNamespace="http://i-lab.mit.edu"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="experimentSpecification">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="input" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="value" type="xs:decimal" />
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="units" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="lab" type="xs:string" use="required" />
      <xs:attribute name="specversion" type="xs:decimal" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.4 Experiment Result

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns="http://i-lab.mit.edu" elementFormDefault="qualified"
  targetNamespace="http://i-lab.mit.edu"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="experimentResult">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="datavector" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string"
                  use="required" />
                <xs:attribute name="units" type="xs:string"
                  use="required" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="lab" type="xs:string" use="required" />
      <xs:attribute name="specversion" type="xs:decimal"
        use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Appendix B

Defining Experiments for the Feedback Systems WebLab

In order to clarify the WebLab architecture described in previous chapters, this appendix works through all the steps needed to run a sample experiment. It begins with a discussion of an experimental setup specified by the Experiment Routine. It then describes a possible interaction with the user to produce a personalized experiment encapsulated by the Lab Client in an Experiment Specification. It concludes with an account of how to run the experiment at the Lab Server, and the kind of results that would be obtained upon successful completion of the experiment.

B.1 Experimental Setup

The following Experiment Routine implements a simple experiment with four user-configurable input parameters: AO_0 through AO_3 . Note how each of these parameters is described by a unique name, a label, units, a default value and a range of valid values. The Experiment Routine also specifies the location of the simple and detailed schematic diagrams corresponding to the experiment.

In addition, the sample experiment provides links to offline data for theoretical results of single and four pole systems. These data can then be downloaded and manipulated by the user from the Lab Client.

Finally, the following Experiment Routine defines the current experiment as a frequency response measurement under swept sine mode. It also sets the source level, start and stop frequencies, as well as the sweep rate parameters for the experiment.

Unlike the gains AO_0 – AO_3 , these experimental parameters do not form part of the Lab Configuration, and are thus unknown (and cannot be modified) by the user.

```
<?xml version="1.0" encoding="utf-8" ?>
<experimentRoutine lab="labjack" specversion="0.1"
  xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
  http://web.mit.edu/6.302/www/weblab/xml/experimentRoutine.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <description>6.302 WebLab Experiment 1</description>

  <schematic>
    <simpleImageURL>
      http://web.mit.edu/6.302/www/weblab/images/labs/filter-b.png
    </simpleImageURL>
    <detailedImageURL>
      http://web.mit.edu/6.302/www/weblab/images/labs/filter-s.png
    </detailedImageURL>
  </schematic>

  <offlineData lab="single pole system">
    <label>Theory data for 10 Hz to 100 kHz</label>
    <resultsURL>
      http://web.mit.edu/6.302/www/weblab/xml/dump/theory-single-pole.xml
    </resultsURL>
  </offlineData>
  <offlineData lab="four pole system">
    <label>Theory data for 10 Hz to 100 kHz</label>
    <resultsURL>
      http://web.mit.edu/6.302/www/weblab/xml/dump/theory-four-pole.xml
    </resultsURL>
  </offlineData>

  <initialization>
    <selectMeas>freq resp</selectMeas>
    <measMode>swept sine</measMode>
  </initialization>

  <measurement>
    <function name="gain">
      <option name="A00" units="V" userEditable="true">
        <label>A0<sub>>0</sub></label>
        <defaultValue>1</defaultValue>
        <minValue>0</minValue>
        <maxValue>5</maxValue>
      </option>
      <option name="A01" units="V" userEditable="true">
        <label>A0<sub>>1</sub></label>
        <defaultValue>1</defaultValue>
      </option>
    </function>
  </measurement>
</experimentRoutine>
```

```

        <minValue>0</minValue>
        <maxValue>5</maxValue>
    </option>
    <option name="A02" units="V" userEditable="true">
        <label>A0<sup>2</sup></label>
        <defaultValue>1</defaultValue>
        <minValue>0</minValue>
        <maxValue>5</maxValue>
    </option>
    <option name="A03" units="V" userEditable="true">
        <label>A0<sup>3</sup></label>
        <defaultValue>1</defaultValue>
        <minValue>0</minValue>
        <maxValue>5</maxValue>
    </option>
</function>

<function name="source">
    <option name="source level" units="V" userEditable="false">
        <label>Source Level</label>
        <defaultValue>1.0</defaultValue>
        <minValue>1.0</minValue>
        <maxValue>2.0</maxValue>
    </option>
</function>

<function name="frequency">
    <option name="start frequency" units="Hz" userEditable="false">
        <label>Start Frequency</label>
        <defaultValue>100</defaultValue>
        <minValue>1</minValue>
        <maxValue>10000</maxValue>
    </option>
    <option name="stop frequency" units="kHz" userEditable="false">
        <label>Stop Frequency</label>
        <defaultValue>10</defaultValue>
        <minValue>1</minValue>
        <maxValue>150</maxValue>
    </option>
    <option name="sweep rate" units="Sec/Dec" userEditable="false">
        <label>Sweep Rate</label>
        <defaultValue>5</defaultValue>
        <minValue>1</minValue>
        <maxValue>100</maxValue>
    </option>
</function>
</measurement>

</experimentRoutine>

```

B.2 Creating an Experiment for Execution

Once the experiment described above has been loaded at the Lab Client, the user is now able to set the values of the gains AO_0 through AO_3 defined in the Lab Configuration.

For the sake of this example, the user chooses a value for all gains equal to 1.0 V. The user then requests her experiment to be executed at the lab site, generating the following Experiment Specification which is sent as part of her request to the Lab Server:

```
<?xml version="1.0" encoding="utf-8" ?>
<experimentSpecification lab="labjack" specversion="0.1"
  xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
  http://web.mit.edu/6.302/www/weblab/xml/experimentSpecification.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <input name="A00" units="V">
    <value>1.0</value>
  </input>
  <input name="A01" units="V">
    <value>1.0</value>
  </input>
  <input name="A02" units="V">
    <value>1.0</value>
  </input>
  <input name="A03" units="V">
    <value>1.0</value>
  </input>

</experimentSpecification>
```

B.3 Obtaining the Results

Finally, once the user's experiment request has been successfully processed at the Lab Server, the Lab Client receives a set of experimental results embedded in the Experiment Result XML data type. In the following example, the results consist of three vectors of data, corresponding to the frequency, magnitude and phase measurements determining the frequency response of the system. Note that only the first two measurements of each vector are included for brevity.


```

<?xml version="1.0" encoding="utf-8" ?>
<experimentResult lab="labjack" specversion="0.1"
  xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
  http://web.mit.edu/6.302/www/weblab/xml/experimentResult.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <datavector name="frequency" units="Hz">
    100.0,100.577306300174,...
  </datavector>
  <datavector name="magnitude" units="dB">
    0.836208992373013,0.836530439529939,...
  </datavector>
  <datavector name="phase" units="deg">
    -11.4084340175456,-11.4635319519532
  </datavector>

</experimentResult>

```

Once the Experiment Result has been parsed at the Lab Client, the frequency response of the experiment can be plotted and the data points optionally be saved to a file. Figure B-1 provides a screenshot of the resulting response obtained at the Lab Client.

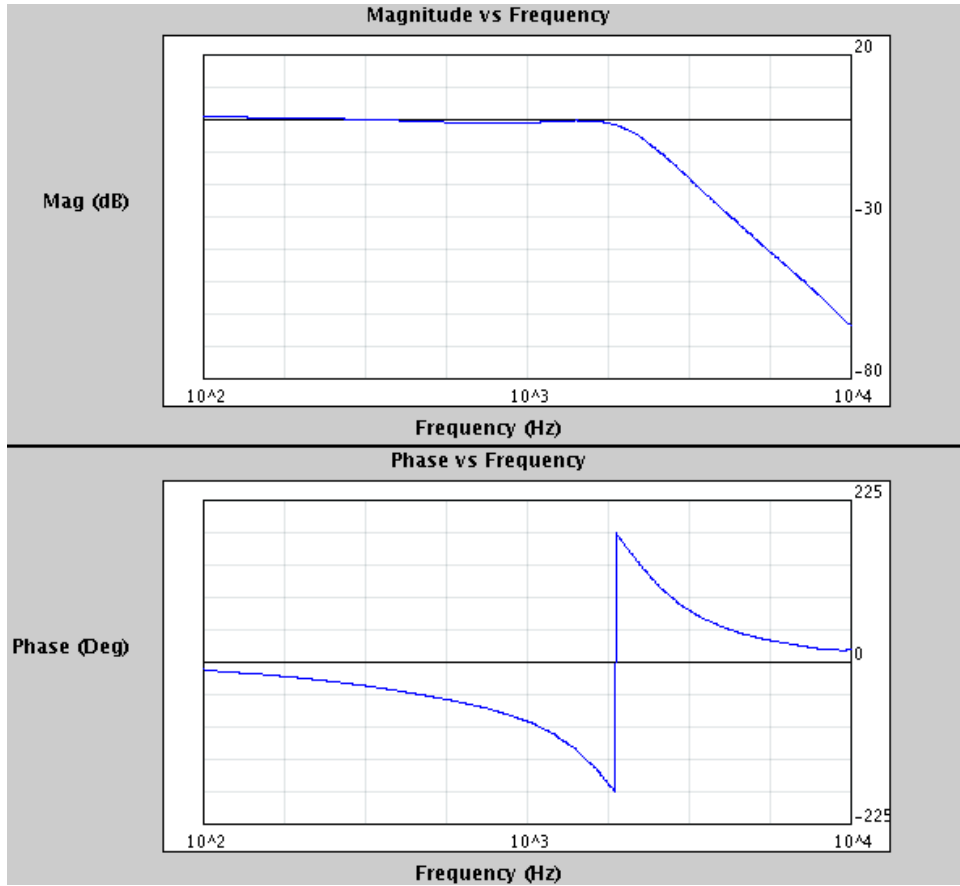


Figure B-1: Screenshot of Lab Client's Bode plot panel after executing the sample experiment at the Lab Server.

Appendix C

Integrating the Feedback Systems WebLab into iLab

Before the Lab Server can receive experiment requests originating from a Lab Client, it must first be registered at the Service Broker. The steps required to publish a Lab Server at a given Service Broker are specified in Section C.1.

Similarly, in order for Lab Clients to communicate with a particular Lab Server it is also necessary for them to be registered with the Service Broker. Once registered, iLab administrators can grant them the appropriate permissions to access any of the Lab Servers registered on that Service Broker, as described in Section C.2.

Ultimately, from the user's point of view, he or she must be able to login to the Service Broker to access the WebLab. To this end, we must be able to create user accounts and grant them the necessary resource permissions to execute experiments. In order to simplify user management, we rely on the creation of groups consisting of collections of users who have identical permissions. The management of users and groups is explained in Section C.3.

C.1 Registering the Lab Server

The steps required to register a Lab Server can be divided into two groups: those steps that take place at the Service Broker, and those that occur at the Lab Server.

C.1.1 Updating the Service Broker

In order to register the Lab Server, the WebLab administrator must first login to the Service Broker's active server pages with superuser privileges. The Lab Server can then be registered by clicking on the *Manage Lab Servers* option under the *Lab Servers and Clients* tab. At this stage, a successful registration of a Lab Server requires the completion of the following entries:

- **Lab Server Name:** the internal name for the Lab Server.
- **Lab Server ID:** a unique GUID-based ID that will be used to identify the Lab Server. This ID is generated at the Lab Server by invoking the `CreateServerID` method from the `ResourcePermissionManager` VB class. Calling this method stores the GUID as the Lab Server ID in the *LSSystemConfig* table of the Lab Server database.
- **Web Service URL:** the URL for the Lab Server – Service Broker web service.
- **Description:** a description of the Lab Server.
- **Lab Info URL:** the URL for the WebLab information pages.
- **Contact First Name**
- **Contact Last Name**
- **Contact E-mail**
- **Outgoing Passkey:** the passkey used to authenticate requests to the Lab Server originating at the current Service Broker. This passkey is included in the SOAP header of all method calls from the Service Broker to the Lab Server web service interface. It is generated by the Lab Server administrator(s).
- **Incoming Passkey:** the passkey used to authenticate incoming requests into the Service Broker originating at the Lab Server. This passkey may be included in the SOAP header of method calls from the Lab Server to the Service Broker web

service interface. It is generated at the Service Broker by clicking on the “Generate Incoming Passkey” button, once all the above items have been successfully completed and saved.

The actual values for the 6.302 Lab Server configuration parameters are given in appendix D.1.

C.1.2 Updating the Lab Server

Before we can communicate between the Service Broker and our Lab Server, however, it is necessary for the Service Broker to be correctly registered at the Lab Server. For this purpose, we can invoke the SQL stored procedure `rpm_AddBroker`. Calling this procedure has the effect of adding a new Broker entity to the *Brokers* table in the WebLab service’s database. When registering a new Service Broker, we must provide the following parameters:

- ***broker_server_id***: the GUID-based ID for the Service Broker.
- ***broker_passkey***: the Lab Server-assigned passkey used to authenticate the Service Broker at the Lab Server. This passkey can be generated by invoking the `CreateForeignPasskey` function from the `ResourcePermissionManager` VB class, which has the effect of updating the Service Broker configuration data stored in the *Brokers* table. Also, note that this passkey corresponds to the Outgoing Passkey of Section C.1.1, and should thus be identical.
- ***server_passkey***: the Service Broker-assigned passkey used to authenticate the Lab Server at the Service Broker. This passkey is generated when registering the Lab Server at the Service Broker through its active server pages, and corresponds to the Incoming Passkey of Section C.1.1.
- ***notify_location***: the URL for the Service Broker-to-Lab Server web service interface.

After updating the Service Broker configuration, it is necessary to define usage classes, groups and their resource permissions at the Lab Server. The SQL stored

procedure `rpm_AddClass` adds a Usage Class record to the `UsageClasses` table. We can then optionally modify the `ClassToResourceMappings` table to fine tune resource access, including granting permissions to view or edit a resource, and establishing its access priority level.

Having established a usage class, the stored procedure `rpm_AddGroup` can then be invoked to add a particular user group into the Lab Server database. These groups will usually correspond to those user groups defined at the Service Broker through its active server pages (e.g. the *6.302_students* or *6.302-devel* groups), as described in Section C.3. In addition, each user group belongs to exactly one usage class, which establishes the resource permissions for that particular group. Thus, updating the usage class a group belongs to, or modifying the attributes of its current usage class, provide the main mechanisms for resource permission management at the Lab Server.

Finally, we must update our firewall configuration to allow for incoming connections from the Service Broker host. In the case of IP filtering, we must add the IP address of the Service Broker to the list of authorized incoming connections.

C.2 Registering the Lab Client

C.2.1 Updating the Service Broker

As with the Lab Server registration, the WebLab administrator(s) must first login to the Service Broker active pages with superuser privileges. The Lab Client can then be registered by clicking on the *Manage Lab Clients* option under the *Lab Servers and Clients* tab. A successful registration of a Lab Client then requires the completion of the following fields:

- **Lab Client ID:** the internal name for the Lab Client.
- **Name:** a descriptive name for the Lab Client.
- **Version:** version information for this Lab Client.
- **Description:** an extended description for the Lab Client.
- **Contact First Name**

- **Contact Last Name**
- **Contact E-mail**
- **Information 01:** the label for the dynamic button component pointing to URL 01.
- **URL 01:** the URL for the dynamic button labeled with “Information 01”.
- **Loader Script:** the HTML Applet tag to be used when loading the client. This tag is used to specify such parameters as the *serviceURL* for the Service Broker web service interface, and the *labServerID* for the Lab Server. These parameters are later read by the Lab Client applet at load time. Also, note that the Service Broker hostname used to access the WebLab (e.g. *i-lab.mit.edu*) must match the one used as part of the *labServerID*. If this is not the case, attempts at Lab Client to Service Broker communication will fail. In addition, the applet tag also specifies the location from which the Lab Client applet jar should be loaded.

After saving the changes, we must associate one or more Lab Servers with our Lab Client. In this way, we can grant permissions for the Lab Client to access and submit experiment requests to a particular Lab Server.

The actual values for the 6.302 Lab Client configuration parameters are given in appendix D.2.

C.2.2 Publishing the Lab Client

Every time the Lab Client is updated, we must replace the compiled jar bundle with the newer version. The jar file for the Lab Client resides at some predefined web location whose URL should match that specified in the Loader Script of Section C.2.1.

C.3 Managing Users

To simplify user management, the Feedback Systems WebLab defines four different user groups: *6.302-devel*, *6.302-tas*, *6.302_students*, and *6.302_students-request*. The first three of these groups have been granted permission to access the Feedback

Systems Lab Client and Lab Server; thus, they allow group members to submit experiment requests through the iLab interface. The latter request group, however, is conceived as a waiting list for newly registered users whose requests have yet to be verified. Since these users cannot yet be trusted, members of this group have no permissions to execute experiments via the interface. Moreover, users with superuser privileges can easily modify the permissions granted to each of the groups by selecting the *Grants* tab in the Service Broker active pages.

New users wishing to be registered must follow the steps below:

1. Point their web browsers to `http://i-lab.mit.edu`.
2. Click on the “Go” button under *New User Registration*.
3. Fill out the form, choosing the desired request-group they wish to join from the list provided (e.g. *6.302_students-request*).
4. Press “Submit”, and await a confirmation email from WebLab administrators.

Once a user has submitted his or her registration request, WebLab administrators automatically receive an email notifying of the incoming request to join the specified group. Logging into the Service Broker with superuser privileges, and accessing the *Group Membership* option under the *Users and Groups* tab displays all of the different groups. The new user whose request has just been received should be visible after expanding the request-group’s node. The administrator can then proceed to manually verify the request, and move the user to the requested group (e.g. *6.302_students*) if the verification is successful.

A confirmation e-mail can then optionally be sent by the administrator once the verification stage has been successfully completed. At this point, the registered user may login to iLab with the username and password they provided along with their request. The user will then be able to access all of the functionality granted to the particular group(s) to which he or she belongs.

Appendix D

Configuration Information for the Feedback Systems WebLab

D.1 Lab Server iLab Configuration

Lab Server: 6.302 .NET Lab Server
Lab Server Name: 6.302 .NET Lab Server
Lab Server ID: e2017fa7470844ba8ad63f7a1eece182
Web Service URL: [http://serv-6302.mit.edu/labserver/
services/WebLabService.asmx](http://serv-6302.mit.edu/labserver/services/WebLabService.asmx)
Description: 6.302 .NET Lab Server
Lab info URL: <http://web.mit.edu/6.302/www/weblab/>
Contact First Name: Gerardo
Contact Last Name: Viedma
Contact E-mail: gviedma@mit.edu
Outgoing Passkey: (SECRET)
Incoming Passkey: (SECRET)

D.2 Lab Client iLab Configuration

Lab Client ID: 6.302 WebLab Client
Name: Transfer Function Lab Client
Version: 1
Description: Measures the Transfer Function
via the HP Signal Analyzer
Contact First Name: Gerardo
Contact Last Name: Viedma
Contact E-Mail: 6.302-weblab@mit.edu
Information 01: Lab Client User's Manual
URL 01: [http://web.mit.edu/6.302/www/weblab/
client/index.html](http://web.mit.edu/6.302/www/weblab/client/index.html)
Information 02: 6.302 iLab FAQs
URL 02: [http://web.mit.edu/6.302/www/weblab/
faq.html](http://web.mit.edu/6.302/www/weblab/faq.html)
Information 03: 6.302 Homepage
URL 03: <http://web.mit.edu/6.302/www/>
Information 04:
URL 04:
Loader Script:

```
<APPLET height=1 width=1
  archive="http://web.mit.edu/6.302/www/weblab/
jar/signed_Weblab_Graphical.jar"
  code="weblab.client.graphicalUI.GraphicalApplet">
  <PARAM NAME="serviceURL"
    VALUE="http://i-lab.mit.edu/Services/
ServiceBrokerService.asmx">
  <PARAM NAME="labServerID"
    VALUE="e2017fa7470844ba8ad63f7a1eece182">
</APPLET>
```


Associated Lab Servers: 6.302 .NET Lab Server

Appendix E

6.302 Feedback Systems Fall 2004 iLab Assignment

Note: Problems 2 and 3 of the 6.302 Fall 2004 iLab assignment were written by Isaac Dancy. For additional information regarding these problems, the reader is referred to Isaac Dancy's master's thesis [31].

Problem 0:

In Problems 1, 2 and 3 we will make use of the 6.302 iLab, an online laboratory that allows you to conduct experiments from any Java-capable web browser. In order to complete these online projects, you must have a 6.302 iLab account. Since it may take several days to process your request, do this part now:

- a. Point your web browser at `http://i-lab.mit.edu`.
- b. Click on the “Go” button under *New User Registration*.
- c. Fill out the form, choosing group *6.302-students-request*.
- d. You will receive email when you press *Submit*.
- e. You will receive another email when your account is fully activated.

Problem 1:

In the next two problems, you will make use of the 6.302 iLab. In this problem, you will run a short experiment to familiarize yourself with the system.

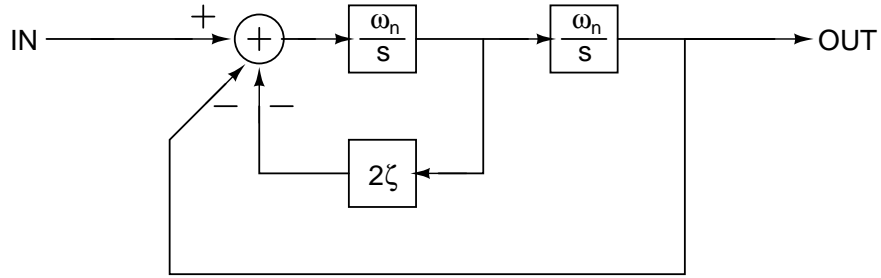
Submitting an experiment to the 6.302 iLab:

- a. Log in to <http://i-lab.mit.edu> with the username and password you specified in Problem 0.
- b. Click on the “Launch Client” button under the 6.302 WebLab Client heading.
- c. Once the client has finished loading (this may take a few minutes depending on your machine), you should see the Lab Client’s Java Applet window appear. Note that you must have Java version 1.4.x or greater to run the client.
- d. On the top left side of the Lab Client window, you will see a number of text inputs labeled AO_0 through AO_3 . Enter a value of 2.0 for all of these inputs.
- e. Now run the experiment by clicking on the leftmost icon in the toolbar below the input area. Upon submitting your experiment, you should see an estimate of your progress appear in a new window.
- f. Once your experiment has been completed, the measured data will be automatically used to plot the corresponding Bode, Nichols and Nyquist plots on the graph area in the bottom of the Lab Client window.
- g. At this point, you can export the measured experimental data by right clicking on the corresponding plot and selecting *Export Graph Data* from the popup menu. You will then be prompted to enter the file name to which to save your data. Any exported graph data will be in Matlab compatible format by default, although it is also possible to export your data to a comma-separated value format via the dialog.
- h. Upon saving your results, you may log out from the WebLab by clicking on the *Log out* link at the top of the i-lab.mit.edu browser window.
- i. Using your saved experimental data, run Matlab and plot your measured results.
- j. Turn in a copy of the Matlab graph.

Problem 2:

Pre-lab calculations for the 6.302 iLab assignment.

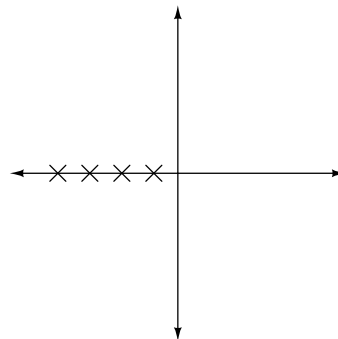
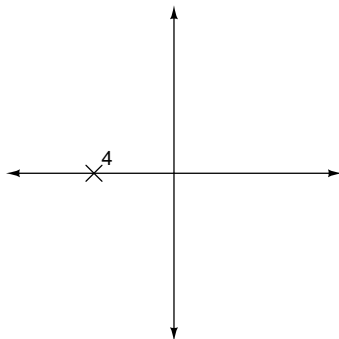
(a) Consider the following system:



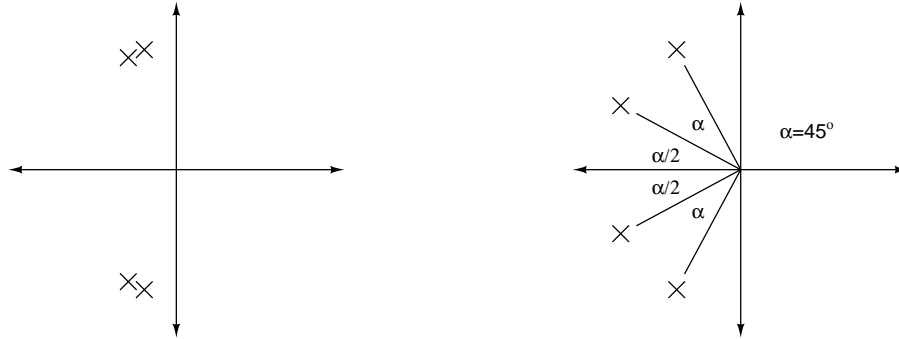
What is the closed-loop transfer function of this system? Have you seen this transfer function before? What does the closed-form of this feedback system implement?

(b) Now suppose that we cascade two copies of this system and have the freedom to control the parameters ζ and ω_n for each system. For each of the following pole-zero plots specify the parameters ζ_1 , ζ_2 , ω_{n1} and ω_{n2} which will generate a closed-loop transfer function with the corresponding pole-zero plot. (Note: Multiple solutions may exist. Limit $\zeta \leq 1.9$ and $2\pi \cdot 100 \text{ rad/s} \leq \omega_n \leq 2\pi \cdot 10^4 \text{ rad/s}$.)

- (i) Four concurrent poles at $s = -2\pi \cdot 10^3 \text{ rad/s}$. (ii) Four poles on the negative real axis.



- (iii) Two conjugate pairs of poles with $\zeta \leq 0.15$ for each system. (iv) 4th-Order Butterworth filter. All poles a distance $2\pi \cdot 10^3$ rad/s from origin.



- (c) Use `Matlab` to generate the corresponding Bode plot and step response for each system in part (b).
- (d) Butterworth filters are systems that exhibit no magnitude peaking and roll off with some slope depending on the order of the filter. Generate a plot in `Matlab` that compares the magnitude response of systems (i) and (iv). What do you notice?
- (e) Figure E-1 is a simplified block diagram of the circuit running at the server of this lab. It uses voltage inputs $AO_0 \rightarrow AO_3$ and multiplier chips (Analog Devices' AD532J) to enable you to tune the closed-loop response to match specific values of ζ and ω_n .

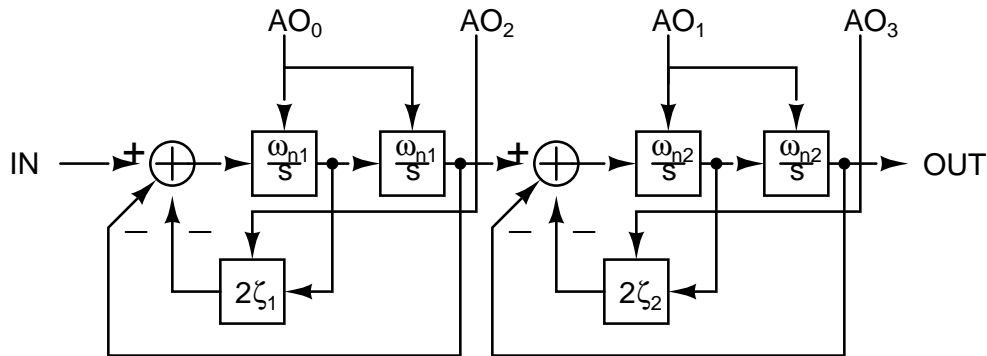


Figure E-1: Simplified block diagram of WebLab circuit.

Figure E-2 is the actual circuit, shown here at half size — inputs $AO_{0,2}$ for the first system correspond to inputs $AO_{1,3}$ for the second system. Generate a block diagram for this circuit (in the general form of Figure E-1) and determine the following relationships. Assume the OP27 is ideal and the AD532J function is

$$Z = \frac{V_1 \times V_2}{10}$$

where V_1 and V_2 are the two input voltages.

- (a) AO_0 and ω_{n1}
- (b) AO_1 and ω_{n2}
- (c) AO_2 and ζ_1
- (d) AO_3 and ζ_2

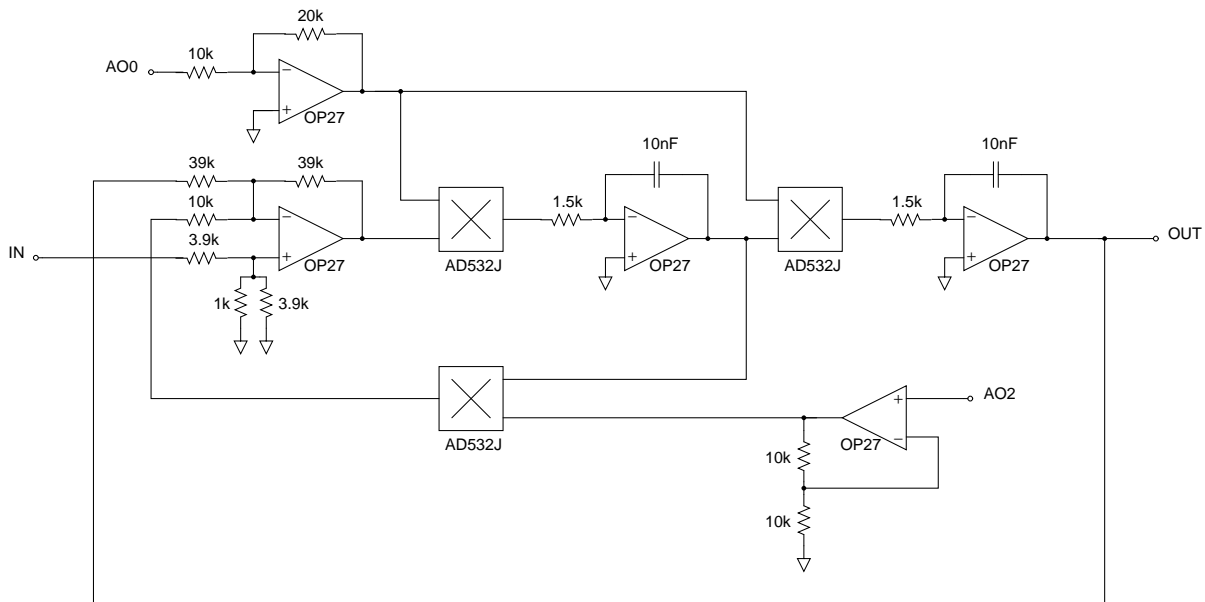


Figure E-2: Actual circuit.

- (e) Parts (i)–(iv). Calculate the input voltages that will result in a system that matches your solutions to part (b), parts (i)–(iv). Note that the input voltages are constrained to $0 < AO_n \leq 5$ V.

Problem 3:

With your pre-lab results from Problem 2, you are now ready to use the 6.302 iLab for your experiments. Log in at <http://i-lab.mit.edu> and click the “Launch Client” button to begin the lab.

- (a) Run a frequency sweep for each set of voltages in part (e) of the prelab. Sweep between 2 and 3 decades in frequency, insuring that you measure all vital parts of the response while keeping your server request manageable. After each sweep make sure you download the data.

It is possible that some valid combinations for the prelab actually saturate the circuit in this part. You **will** know if this has happened, and should reduce the magnitude peaking of your solution until it no longer saturates the server circuit.

- (b) Use `Matlab` to plot and print out your measured responses. Ambitious students can include the theoretical response on the same plots.

(`Matlab` Hint: This can be achieved by using the form of the `bode` command that returns magnitude, phase and frequency information. Use the `db` command so that the theoretical magnitude data matches the downloaded data.)

- (c) The write up for this lab should be short, simple and informal. Do your best to conserve paper when printing from `Matlab`, but do avoid cluttering any one plot excessively.

Appendix F

Fall 2004 6.302 iLab Survey

The following page includes the survey that was handed out to students in the Fall 2004 version of 6.302 Feedback Systems.

Fall 2004 6.302 iLab Survey

In order to gauge the effectiveness of the 6.302 WebLab assignments, we need your thoughts on how the online lab assignments contributed to your learning of the course material. Please take a few moments to complete the survey and hand it in at the end of class.

1). Overall, how would you rate the ease of use of the 6.302 WebLab?

1 (very difficult) 2 3 4 **5 (very easy)**

2). How would you rate the overall responsiveness when running experiments on the 6.302 WebLab?

1 (very poor) 2 3 4 **5 (very good)**

3). How useful did you find the WebLab assignment for learning/applying the course material?

1 (not useful) 2 3 4 **5 (very useful)**

4). How useful did you find the 6.302 WebLab assignment for analyzing/testing your solution on a real system?

1 (not useful) 2 3 4 **5 (very useful)**

5). When completing your WebLab assignment, where did you spend the most time? (Rank from 1 to 4 in order of importance).

Analyzing/solving the problem. [1 2 3 4]

Learning how to use the WebLab. [1 2 3 4]

Troubleshooting the WebLab. [1 2 3 4]

Waiting for experiment requests to complete. [1 2 3 4]

6). What was the best thing about the 6.302 WebLab?

7). What was the worst thing about the 6.302 WebLab?

8). In future versions of 6.302, what use of the WebLab system would you consider best for teaching the course material? Please mark one of the following options.

Only conventional in-lab assignments.

Some conventional in-lab assignments and some WebLab assignments.

Only WebLab assignments.

9). Are there any changes to the Lab Client that you think would make the WebLab easier to use?

10). Please write any other feedback or comments here.

Bibliography

- [1] K. H. Lundberg *et al.*, “6.302 iLab homepage: A WebLab for signals, systems, circuits, and control,” Massachusetts Institute of Technology, (Date downloaded: December 15, 2004). [Online]. Available: <http://web.mit.edu/6.302/www/weblab/>
- [2] M. Exel, S. Gentil, and D. Rey, “Simulation workshop and remote laboratory: Two web-based training approaches for control,” in *Proceedings of the American Control Conference*, vol. 5, Chicago, IL, June 2000, pp. 3468–3472.
- [3] D. A. Miele, B. Potsaid, and J. T. Wen, “An Internet-based remote laboratory for control education,” in *Proceedings of the American Control Conference*, vol. 2, Arlington, VA, June 2001, pp. 1151–1152.
- [4] M. Casini, D. Prattichizzo, and A. Vicino, “The automatic control telelab,” *IEEE Control Systems Magazine*, vol. 24, no. 3, pp. 36–44, June 2004.
- [5] D. Z. Deniz, A. Bulancak, and G. Özcan, “A novel approach to remote laboratories,” in *ASEE/IEEE Frontiers in Education Conference*, vol. 1, Boulder, CO, Nov. 2003, pp. T3E-8–T3E-12.
- [6] J. Sánchez, S. Dormido, R. Pastor, and F. Morilla, “A Java/Matlab-based environment for remote control system laboratories: Illustrated with an inverted pendulum,” *IEEE Transactions on Education*, vol. 47, no. 3, pp. 321–329, Aug. 2004.

- [7] A. Ferrero, S. Salicone, C. Bonora, and M. Parmigiani, "ReMLab: A Java-based remote, didactic measurement laboratory," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 3, pp. 710–715, June 2003.
- [8] M. L. Corradini, G. Ippoliti, T. Leo, and S. Longhi, "An Internet based laboratory for control education," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 3, Orlando, FL, Dec. 2001, pp. 2833–2838.
- [9] Q. Yu, B. Cheng, and H. H. Cheng, "Web-based control system design and analysis," *IEEE Control Systems Magazine*, vol. 24, no. 3, pp. 45–57, June 2004.
- [10] C. C. Ko, B. M. Chen, J. Chen, Y. Zhuang, and K. C. Tan, "Development of a web-based laboratory for control experiments on a coupled tank apparatus," *IEEE Transactions on Education*, vol. 44, no. 1, pp. 76–86, Feb. 2001.
- [11] H. H. Hahn and M. W. Spong, "Remote laboratories for control education," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 1, Sydney, Australia, Dec. 2000, pp. 895–900.
- [12] J. Harward *et al.*, "iLab: A scalable architecture for sharing online experiments," in *International Conference on Engineering Education*, Gainesville, FL, Oct. 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://icampus.mit.edu/iLabs/Architecture/Downloads/default.aspx>
- [13] B. Aktan, C. A. Bohus, L. A. Crowl, and M. H. Shor, "Distance learning applied to control engineering laboratories," *IEEE Transactions on Education*, vol. 39, no. 3, pp. 320–326, Aug. 1996.
- [14] G. Viedma, I. Dancy, and K. Lundberg, "A web-based linear-systems iLab," in *American Control Conference*, Portland, OR, June 2005, accepted for publication.
- [15] J. del Alamo *et al.*, "Microelectronics devices and circuits WebLab," Massachusetts Institute of Technology, (Date downloaded: December 15, 2004). [Online]. Available: <http://weblab.mit.edu>

- [16] S. Lerman and J. del Alamo, “iLab: Remote online laboratories,” (Date downloaded: August 19, 2004). [Online]. Available: <http://icampus.mit.edu/projects/iLab.shtml>
- [17] “iCampus,” MIT-Microsoft Alliance, (Date downloaded: August 19, 2004). [Online]. Available: <http://icampus.mit.edu>
- [18] J. del Alamo, J. Harward, S. Lerman, and K. Amaratunga, “A web-service architecture to bring labs online,” in *Microsoft Faculty Summit 2004*, August 2004, (Date downloaded: December 15, 2004). [Online]. Available: <https://faculty.university.microsoft.com/2004/>
- [19] S. Dormido, “Control learning: Present and future,” in *Proceedings of the IFAC 15th Triennial World Congress*, Barcelona, Spain, July 2002, pp. 81–103.
- [20] “SOAP Specifications,” The World Wide Web Consortium (W3C), (Date downloaded: August 27, 2004). [Online]. Available: <http://www.w3.org/TR/soap/>
- [21] “Extensible Markup Language (XML),” The World Wide Web Consortium (W3C), (Date downloaded: May 15, 2004). [Online]. Available: <http://www.w3.org/XML/>
- [22] “LabView: Visual instrumentation software from NI,” National Instruments, (Date downloaded: August 27, 2004). [Online]. Available: <http://www.ni.com/labview/>
- [23] “Web Service Definition Language (WSDL),” The World Wide Web Consortium (W3C), (Date downloaded: August 27, 2004). [Online]. Available: <http://www.w3.org/TR/wsdl>
- [24] “Universal Description, Discovery and Integration (UDDI),” OASIS, (Date downloaded: August 27, 2004). [Online]. Available: <http://www.uddi.org/>

- [25] D. Zych, "Lab client to service broker API," MIT iCampus, Tech. Rep., Oct. 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://icampus.mit.edu/iLabs/Architecture/Downloads/default.aspx>
- [26] J. Harward, "Service broker to lab server API," MIT iCampus, Tech. Rep., Oct. 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://icampus.mit.edu/iLabs/Architecture/Downloads/default.aspx>
- [27] G. Viedma, "Design and implementation of a feedback systems web laboratory prototype," Advanced Undergraduate Project, Massachusetts Institute of Technology, May 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://web.mit.edu/6.302/www/weblab/>
- [28] "W3C XML Schema," XMLSchema, (Date downloaded: May 15, 2004). [Online]. Available: <http://www.w3.org/XML/Schema>
- [29] "Introduction to XML Schemas," W3Schools, (Date downloaded: August 19, 2004). [Online]. Available: http://www.w3schools.com/schema/schema_intro.asp
- [30] "Guide to the W3C XML Specification DTD, Version 2.1," The World Wide Web Consortium (W3C), (Date downloaded: May 15, 2004). [Online]. Available: <http://www.w3.org/XML/1998/06/xmlspec-report.htm>
- [31] I. Dancy, "Educational hardware for feedback systems," Master's thesis, Massachusetts Institute of Technology, Aug. 2004.
- [32] P. Dhawan and T. Ewald, "Building distributed applications with Microsoft .NET," Microsoft Developer Network, (Date downloaded: September 02, 2004). [Online]. Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/%html/bdadotnetarch16.asp>
- [33] A. Skonnard, "How ASP.NET web services work," Microsoft Developer Network, (Date downloaded: September 02, 2004). [Online]. Available: <http://msdn.microsoft.com/library/en-us/dnwebsrv/html/howwebmeth.asp>

- [34] “Accessing SQL Server from a web application,” Microsoft Developer Network, (Date downloaded: May 15, 2004). [Online]. Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/%html/vbconaccessingsqlserverfromwebapplication.asp>
- [35] “kSOAP Project,” Enhydra, (Date downloaded: May 15, 2004). [Online]. Available: <http://ksoap.enhydra.org>
- [36] G. Viedma, “6.302 WebLab lab client manual,” Massachusetts Institute of Technology, (Date downloaded: December 15, 2004). [Online]. Available: <http://web.mit.edu/6.302/www/weblab/client/client-howto.html>
- [37] D. Zych, “Microelectronics devices and circuits WebLab client,” Massachusetts Institute of Technology, (Date downloaded: December 15, 2004). [Online]. Available: <http://weblab.mit.edu>
- [38] S. Lokanathan, “Extension to the feedback systems web laboratory client prototype,” Advanced Undergraduate Project, Massachusetts Institute of Technology, July 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://web.mit.edu/6.302/www/weblab/>
- [39] B. Williams, “A graphical package for Bode, Nichols and Nyquist plots,” Advanced Undergraduate Project, Massachusetts Institute of Technology, May 2004, (Date downloaded: December 15, 2004). [Online]. Available: <http://web.mit.edu/6.302/www/pz/>
- [40] P. Missiuro, Personal communication, December 2004.
- [41] A. Solis, “MIT device simulation WebLab: An online simulator for microelectronic devices,” Master’s thesis, Massachusetts Institute of Technology, Sept. 2004.