Design and Implementation of a Feedback Systems Web Laboratory Prototype

Gerardo Viedma Núñez

AUP Final Report

MIT EECS

May 25, 2004

Supervisor: Dr. Kent Lundberg

Contents

Chapter 1

Introduction

The idea of remote laboratories became a reality with the advent of distributed systems that could use 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 excellent opportunities to explore new teaching methodologies that make use of these new 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 in Feedback Systems.

1.1 Overview

The Feedback Systems Web Laboratory (henceforce referred to as WebLab) provides an effective means for students to conduct experiments from a variety of a locations through a web browser. Using our remote web-accessible laboratory students can conduct experiments at any time and any place that is convenient for them. In addition to its flexibility from the student point of view, a web lab 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 expensive and scarce equipment. This 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 traditional lab setting.

1.2 Related Work

The idea of remote web-accessible labs is not a new one. In particular, 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 web labs. In particular, this project is indebted to the developers of the Microelectronics Devices and Circuits WebLab [1], who provided much of the framework and tools that were leveraged in the building of our WebLab prototype.

Chapter 2

The WebLab System

The iLab framework proposes a distributed system consisting of three main layers for building a WebLab. These three layers communicate with each other using SOAP messages exchanged through web services. The first layer is 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 of the system that is visible to the end user.

The iLab infrastructure also provides a trusted intermediary, known as the *Service Broker*, which forwards requests from the Lab Client to the Lab Server for execution. The main task of the Service Broker is to grant users appropriate permissions when forwarding their requests for experiments. The Service Broker also provides additional administrative services, such as temporarily storing experiment requests from the Lab Client, or experiment results from the Lab Server.

Finally, the *Lab Server* constitutes the third layer of the framework. The task of the Lab Server 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 to run an experiment with a particular set of experimental parameters. In this case, the Lab Server communicates with the lab equipment and replies with the experimental results upon successful completion of the experiment.

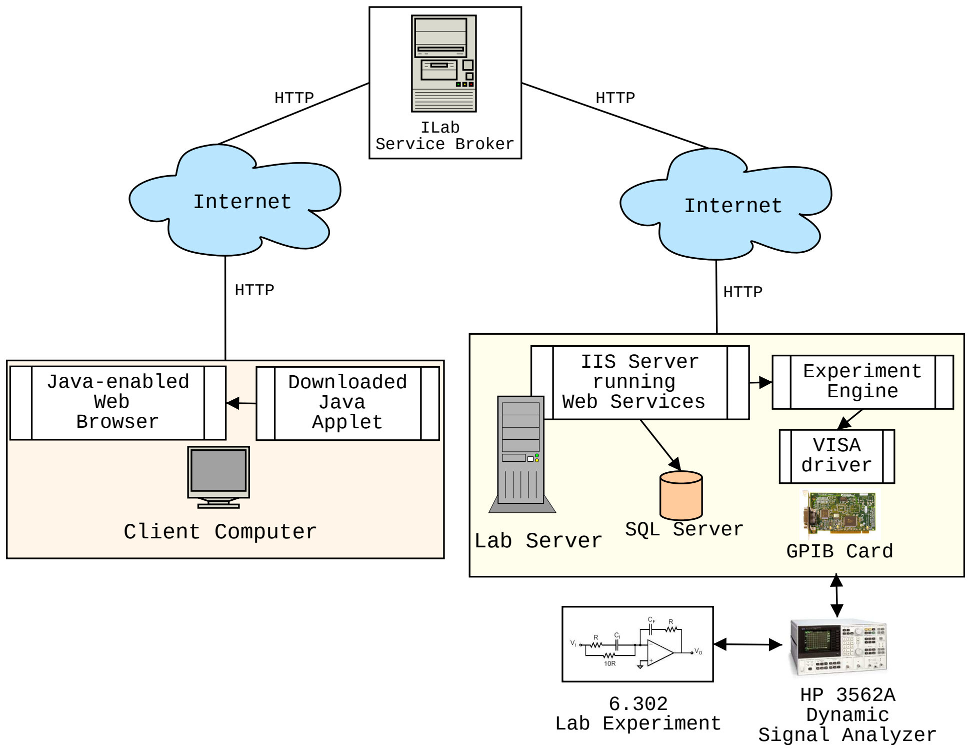


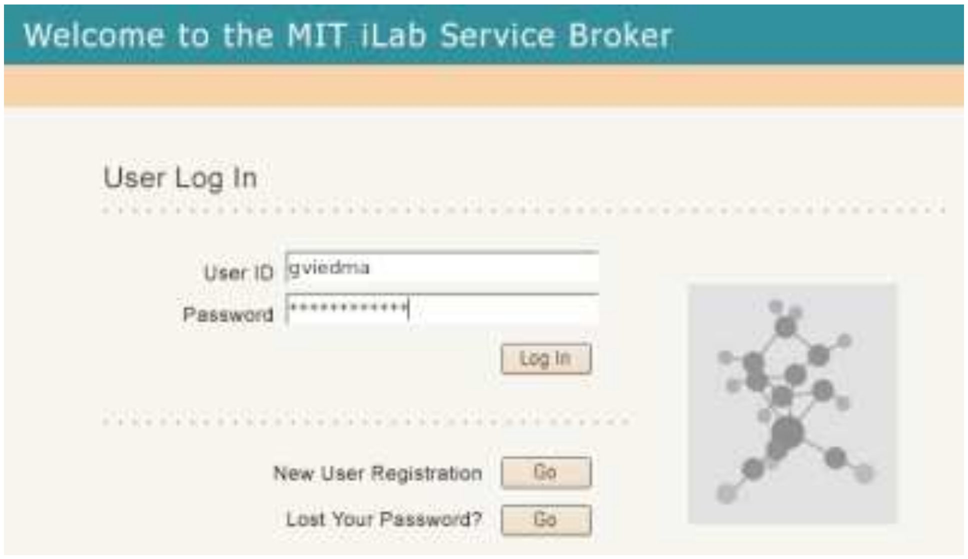
Figure 2-1: Architectural overview of the Feedback Systems WebLab Prototype

2.1 The Feedback Systems WebLab Prototype

In the Feedback Systems WebLab prototype, the user loads a Java applet through an active server page hosted at the Service Broker machine. Once the Java-based Lab Client is started, the user can make requests to the Lab Server via the user interface. These requests are received as SOAP messages at the Service Broker, which then verifies the user's credentials and passes on the request to the Lab Server if successful. In order to be truly web-accessible, all the communications between Lab Client, Service Broker and Lab Server occur over HTTP, as seen in Figure 2-1. The following sections describe the task of the three main layers in additional detail.

2.1.1 Registering with the Service Broker

In order to take advantage of the iLab infrastructure, it is first necessary to register both the Lab Client Java applet and the Lab Server with the Service Broker. For the



User ID: gviedma

Figure 2-2: Login page for Feedback Systems WebLab

Lab Client applet registration, the Service Broker must be notified of the URL where the Lab Client applet resides. An active server page can then be created at the Service Broker for running the Lab Client applet. This ASP page contains embedded applet parameter tags specifying the Lab Server ID and the Service Broker web service URL for the WebLab. These parameters are needed for the Lab Client applet to successfully communicate with the Lab Server via the Service Broker.

In addition, the Lab Server must also be registered at the Service Broker. The Lab Server must generate a unique Lab Server ID (currently implemented using the Windows GUID) along with a passkey, and communicate them to the Service Broker. Similarly, the Service Broker must communicate its ID and passkey to the Lab Server, in order for the Lab Server to successfully register the Service Broker. The passkey/ID pairs currently serve as a simple authentication scheme by encoding them in the SOAP header of any messages sent from Service Broker to Lab Server. Finally, the Service Broker must also be notified of the URL for the web service running at the Lab Server.

2.1.2 The Lab Client

When the user first enters the system, they are prompted for a username and password, as in Figure 2-2. Once their credentials have been verified, they may load the applet for the WebLab. Since the Lab Client applet is implemented in Java, it can be run from any operating system with a Java-enabled web browser and support for Java script. A screenshot of the Lab Client is provided in Figure 2-3.

Once the applet is launched, the user is presented on the left side of the screen with a list of parameters that can be modified. The user can also perform generic operations such as running an experiment, obtaining information about the Lab Server, and plotting and saving the experimental results to a file. These operations can be performed through the applet menus, or directly through the toolbar on the right side of the applet. At the center of the screen, the user is provided with a schematic diagram of the experiment that is being modified. Finally, on the lower half of the screen, the user is presented with a number of plots of the experimental results, including both phase and magnitude responses for the experiment.

2.1.3 The Lab Server

The Lab Server for the Feedback Systems WebLab is a Pentium III 500MHz machine with 500K of RAM. It runs Microsoft's Internet Information Services (IIS) on top of the Windows 2000 Server operating system. The Lab Server implements the WebLab web service by means of Microsoft's ASP.NET runtime engine. The WebLab web service running at the Lab Server can be accessed through the following URL: http://serv-6302.mit.edu/labserver/services/WebLabService.asmx.

The Lab Server also runs Microsoft's SQL Server for accessing database information regarding Service Brokers and laboratory hardware, as well as for queueing experiment requests. Finally, the Lab Server programs the experiments it receives on the HP 3562A dynamic signal analyzer. All communications between the Lab Server and the signal analyzer take place over the GPIB interface through a USB-to-GPIB controller connected to the Lab Server.

2.2 The iLab API

In order for the WebLab system to be truly distributed, the different layers must adhere to a set of standardized operations as provided in the iLab API. There are two sets of service calls specified by the iLab API: service calls from Lab Client to Service Broker [9], and service calls from Service Broker to Lab Server [3].

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.

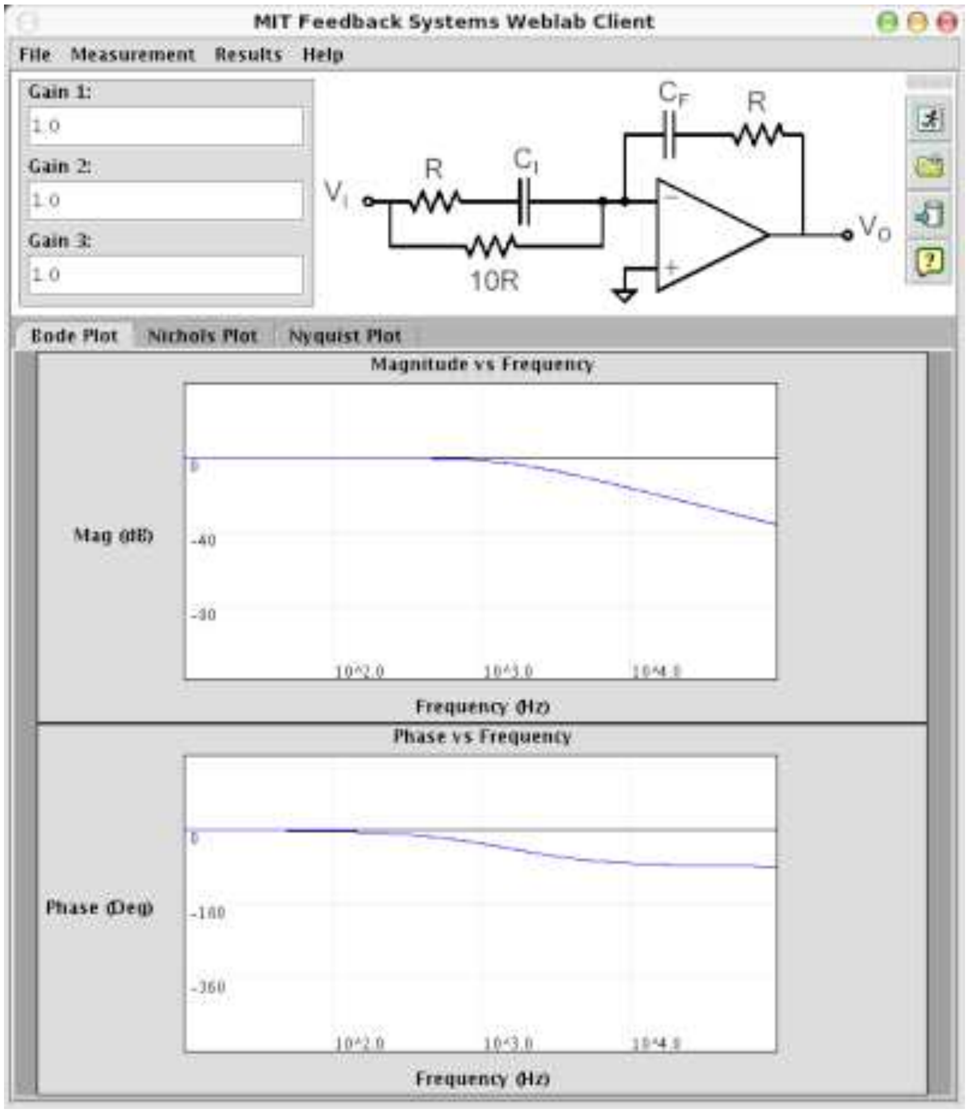


Figure 2-3: Client Applet for the Feedback Systems WebLab Prototype

Chapter 3

The Feedback Systems WebLab Prototype

This section describes the implementation of the WebLab prototype for Feedback Systems, and how it fits into the iLab framework. First, we specify the syntax and semantics for describing experiments between Lab Client and Lab Server. We then describe the implementation of Lab Client and Lab Server, and how communication between them takes place through web services.

3.1 Describing Experiments for the Feedback Systems WebLab

The iLab framework provides 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, these specifications may reside anywhere on the World Wide Web, meaning that the experiment can be modified dynamically on the fly by anone with the appropriate permissions.

The three specifications are the *Lab Configuration*, the *Experiment Specification* and the *Experiment Result*. In order to facilitate interoperability and the transfer of information across the Web, instances of these specifications are encoded in the Extensible Markup 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 specified in [5].

While actual instances of these specifications are encoded in XML, the specifications themselves are encoded using the XML Schema language defined in [8]. XML Schema formalizes the syntax and semantics used for creating and validating instances of the Lab Configuration, Lab Specification and Experiment Result specifications, as described in the following sections.

3.1.1 Lab Configuration

The Lab Configuration is used by the Lab Server to describe the general configuration of the experiment requested by the Lab Client. More specifically, it specifies a list 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 about the experiment, such as a text description of the experiment and a URL pointing to the experiment's schematic diagram. An example of a Lab Configuration is provided in appendix B.1, and the XML Schema definition is specified in A.1.

3.1.2 Experiment Specification

The Experiment Specification is prepared by the user at the Lab Client end, and represents the parameter values constituting this particular run of the experiment. Similarly to the Lab Configuration, an Experiment Specification consists of lists of inputs along with their values. An example of a Lab Configuration is provided in appendix B.2, and the XML Schema definition is specified in A.2.

3.1.3 Experiment Result

Once the experiment has completed successfully, the Lab Server produces a list of datavectors containing the measured results as part of the Experiment Result type.

The datavectors are specified by their name, their units, and a list of comma-separate numeric values. An example Experiment Result is provided in section B.3, and the XML Schema definition is specified in A.3.

3.2 Lab Server Implementation

The Lab Server implements the iLab WebLab web service using .NET technology. The web service was implemented in the Visual Basic .NET language on a web server running Internet Information Services (IIS), and can be invoked through http://serv-6302.mit.edu/labserver/services/WebLabService.asmx. Besides the web server, the Lab Server also has an SQL database running on Microsoft SQL Server 2000. This databse is called from the web services module for a variety of purposes, including authentication and authorization of the service, logging calls to the web service, and for retrieval of hardware information relating to the experiment. In addition, the database is used to enqueue experiment requests and to save any experimental results that were obtained at the Lab Server.

Running on a different thread from the web service, the Lab Server runs an experiment engine. The task of the experiment engine is to check for any incoming experiment requests using the database, and if any pending requests are found, it runs the corresponding experiment specified by the user in the Experiment Specification. The experiment is run by communicating over the GPIB bus with an HP 3562A digital signal analyzer attached to a prepared circuit. After sending the appropriate settings to the digital signal analyzer and beginning the measurement, the Lab Server can begin to receive the experimental results which will later be sent to the Lab Client as part of the Experiment Result XML data.

3.3 Lab Client Implementation

The Lab Client module runs a lightweight web services implementation for Java known as kSOAP [2]. Because of its small footprint, kSOAP is suitable for building a SOAPenabled Java applet from which to run the Lab Client.

The GUI of the Lab Client consists of three main components, as show in Figure 2-3. First, on the upper left side of the screen, there is a number of text fields with their corresponding labels. Here, the user can personalize the experiment by varying the value of the parameters that will be sent to the Lab Server in the Experiment Specification. The Lab Client applet parses the Lab Configuration specified by 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 also specified in the Lab Configuration and therefore can also be modified on the fly.

Lastly, on the lower side of the screen there is a graph panel developed by Brian Williams [7] that displays Bode, Nichols and Nyquist plots of the collected data.

Chapter 4

Discussion

This section discusses the different tasks that were completed, where the main challenges lay, and an account of the time spent on each task.

4.1 Designing the Specifications

Following the footsteps of 6.012, our first implementation of the specifications was based around DTDs (document type definition) [6]. Although DTDs are the default schema of XML and have a minimum processing overhead, they are somewhat limited in expresiveness. Furthermore, it can be difficult for humans to interpret a DTD and acquire some basic information about a data type. In contrast, the new standard of XML Schema provides a possibly more expressive and flexible means for specifying a datatype [8]. Also, it is considerably easier for humans to interpret a datatype defined in XML Schema than one defined using the more rigid DTD syntax. For these reasons, and in spite of the slight additional overhead, I decided to rewrite the specifications using XML Schema for their second version.

Unfortunately, modifying the labConfiguration, experimentSpecification and experimentResult specifications caused a number of problems. The Lab Client made some unfortunate assumptions regarding the namespace which caused it to fail. In addition, modifying these specifications required several changes to the Lab Client and Lab Server, where the XML headers are sometimes hard-coded within the source code.

In an attempt towards a more distributed environment, we decided to place some of these configuration files, such as the labConfiguration, in a web-accessible location (e.g. a course locker). As a result, the Lab Server no longer uses the database to load these configuration files, but reads them instead from an arbitrary URL. This method would allow anyone with access to this file to easily modify the labConfiguration dynamically, without the need to log into the Lab Server's database. A yet more modular design could use the database to store a URL location for this type of configuration files, rather than storing the actual files themselves.

4.2 Deploying the Lab Client

In contrast to the Lab Server, the Lab Client was particularly easy to build. A centralized makefile allowed me to compile all of the Lab Client code and build the Lab Client jar file with a single command. However, the process of generating a certificate for the 6.302 WebLab was not as well documented, and required some extra effort and research. This step of the project took about one week to complete.

It also required some time to understand what classes were involved in the parsing, and what code was specific to 6.012 and could be safely removed. Overall, the Lab Client was designed in a very modular way, yet some of the design decisions and implementation strategies may have focused around specialized 6.012 needs. This part took about one additional week.

One particular problem I encountered at the Lab Client involved the parsing of XML messages. As discussed above, I decided to use XML schema using http://i-lab.mit.edu as the namespace. This modification had the unfortunate result of breaking one of the parsing classes, which assumed that the XML file used the default namespace instead.

Another of the challenges I faced was integrating the Service Broker for communication between Lab Client and Lab Server. After minor problems with Visual SourceSafe when deploying the 6.302 login page, some hidden assumptions in the

Service Broker design also emerged. In particular, we realized after much distress and perplexity that only accessing the Service Broker through its fully qualified name would allow the Lab Client to be successfully loaded at the browser. Omitting the *mit.edu* part of the Service Broker host name or using its IP address would silently fail, since the Service Broker made an unfortunate assumption regarding the login URL when loading the applet web page.

Integration and testing with the Service Broker required about two weeks to complete.

4.3 Lab Server Deployment

4.3.1 Getting Started

One of the greatest challenges in the deployment of the 6.302 Lab Server lay in understanding the diverse body of 6.012 Lab Server source code in order to modify it appropriately. The Lab Server consists of three main components: the web services front end, the database, and the experiment engine. To this end, the Lab Server was designed so as to cleanly divide the functionality among the different modules. In spite of the modularity of the design, the overall complexity of such a large system meant that understanding how and where each of these modules interact with one another often required some amount of trial and error and debugging. In addition, the number of dependencies between these modules made it a little difficult to follow an iterative development process at the Lab Server, since most of the components had to be working (or removed) for the others to also work or be tested.

4.3.2 Web Services

One of the first hurdles in using the 6.012 Lab Server's web services code lay in the lack of a well-defined building strategy. Often, no make files were available to build the executables and libraries, and only the plain Visual Basic files without their corresponding Visual Studio solution files could be found. After some experimenting, searching the Web, and much help from James Hardison, I was able to generate the necessary make files. To this end, I wrote my own executable batch files, and created the necessary Visual Studio solutions with the appropriate references and namespaces. In this way, I had the opportunity to recompile the web services front end in a much faster and effortless fashion.

In addition, I found some of the directory structures and namespaces used could be slightly confusing and sometimes difficult to work with. I made a few minor changes that I thought simplified the Lab Server code in this way.

When I finally began to implement the Lab Server web services interface, I stumbled against another problem: testing. The source code I was using assumed communication with a Service Broker employing SOAP headers for authentication. This requirement made it difficult to test any of the web service methods without involving a Service Broker. Unfortunately, our lack of a Lab Client at this early stage of the project meant that we had no mechanism against which we could test the Lab Server through the Service Broker. I decided to postpone the additional complexity resulting from incorporating a Service Broker by removing all of the authentication code. At this point, I was able to begin testing my web service methods through the web service test pages. For the initial deployment, it would have been helpful if there had been an option for enabling and disabling web service authentication dynamically (or through a switch) at the Lab Server end.

Overall, understanding the web services framework and how to use and modify it took me well over a month.

4.3.3 Database

The database generating code assumed certain parameters and directory structures which had to be modified for it to be deployed on our Lab Server machine. In addition, the database structure was sometimes very specific to 6.012 (e.g. table columns, hardware parameters, etc.) and needed to be somewhat adapted for 6.302.

Once a secret key and ID had been generated for the Service Broker and Lab Server, it took some effort to understand how to incorporate these into the database.

It would be very helpful if some of the administrative database methods were more extensively documented. An administrative webpage similar to that employed by Service Broker administrators might greatly facilitate these tasks.

Another design decision that I reconsidered was the mechanism for communicating between the web service and the database. The original Lab Server implementation proposed having a special Lab Server user to log in to the database with a password. Although this method has the advantage that the database server need not be present at the local host, it can be less secure since users and passwords need to be generated and incorporated as plaintext in the source code. I decided instead to use the reserved ASPNET user for logging into the database, using Windows Intergrated Security with SSPI (Security Support Provider Interface) [4].

4.3.4 Experiment Engine

The other major component in the Lab Server implementation was the experiment engine. The hardware used by the Feedback Systems WebLab is very different from that used by 6.012. As a result, very little of the code could be reused for the experiment engine implementation. Unfortunately, the GPIB card used was also different, which meant that the GPIB initialization and setup routines had to be implemented from scratch. Communicating with the dynamic signal analyzer also proved challenging. The age of the hardware meant that it did not adhere to the standardized GPIB command syntax that was specified in Agilent's manuals. However, the signal analyzer programming manual provided the necessary information to implement a simple experiment for obtaining the response of a low-pass filter. With our hardware, the syntax used for issuing GPIB commands consists of text-based aliases that the signal analyzer can interpret and execute.

Overall, rewriting the experiment engine took a few weeks, and learning to communicate with the signal analyzer over GPIB required two whole days.

Chapter 5

Further Work

Currently, the Lab Client and Lab Server components have been put in place and the potential to build a WebLab for Feedback Systems has just begun to be explored. A static non-configurable experiment was implemented in the Lab Server's experiment engine as a proof of concept of the architecture. For the WebLab to be truly useful in the class setting, however, much further work is necessary in creating dynamic configurable experiments for students to work with. To this end, a WebLab driver abstracting the experiment hardware and a more elaborate experimental setup would need to be developed in the short term future.

Another interesting addition to the project is to provide users with the choice of either simulating the experiment, running it directly on real hardware, or both. In this way, it would be possible to compare theory and practice by displaying both results simultaneously on the screen.

Finally, the fact that the Lab Server is currently limited to purely Microsoft technologies limits the flexibility and openness of its implementation. A parallel Lab Server implementation based on open source would be of great interest in the long run, also providing additional support to the notion of interoperability of web services technology.

Appendix A

Document Type Definitions for Feedback Systems WebLab

A.1 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">

  <xs:element name="labConfiguration">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="description" type="xs:string" />
            <xs:element name="imageURL" type="xs:string" />
            <xs:element name="input" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <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:sequence>
                  <xs:attribute name="name" type="xs:string" />
                  <xs:attribute name="units" type="xs:string" />
               </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.2 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" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="value" type="xs:decimal" />
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string" />
               </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.3 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

Running an Example Experiment

In order to clarify the WebLab architecture described in previous chapters, this chapter works through all the steps needed to run a toy experiment. It starts with a description of the experimental setup specified in the Lab Configuration. It then describes a possible interaction with the user to produce a personalized experiment defined in the 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 Lab Configuration specifies a simple experiment with only one configurable input parameter *gain1*. Note how this parameter is further described with a name, units, a label, a default value and its valid range. The Lab Configuration also specifies the location of the image file representing the experiment schematic for display to the user.

```
<?xml version="1.0" encoding="utf-8" ?>
<labConfiguration lab="lab0" specversion="0.1"
xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
http://web.mit.edu/gviedma/Public/weblab/xml/labConfiguration.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <description>6.302 Weblab Experiment 1</description>
   <imageURL>
     http://web.mit.edu/gviedma/Public/weblab/images/labs/lab0.png
   </imageURL>
   <input name="gain1" units="units">
      <label>Gain 1</label>
      <defaultValue>1.0</defaultValue>
      <minValue>0.0</minValue>
      <maxValue>10.0</maxValue>
   </input>
</labConfiguration>
```

B.2 Creating an Experiment for Execution

Once the experiment described above has been loaded at the client, the user is now able to set the value of the input parameter *gain1* defined in the Lab Configuration.

For the sake of this example, the user choses a value for *gain1* equal to 5.0. The user then requests her experiment to be executed at the lab server, generating the following Lab Specification which is sent as part of her request to the lab server:

```
<?xml version="1.0" encoding="utf-8" ?>
<experimentSpecification lab="lab0" specversion="0.1"
xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
http://web.mit.edu/gviedma/Public/weblab/xml/experimentSpecification.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <input name="gain1">
      <value>5.0</value>
   </input>
</experimentSpecification>
```

B.3 Obtaining the Results

Finally, once the user's experiment request has been successfully processed at the lab server, the client receives a set of experimental results embedded in the Experiment Result XML data type. In the following example, the results consist of two vectors of data, one for real values, and the other for the corresponding imaginary values. Also, notice that in the following case there are a total of 4 data points; in other words, there are 4 pairs of real/imaginary values.

```
<?xml version="1.0" encoding="utf-8" ?>
<experimentResult lab="lab0" specversion="0.1"
xmlns="http://i-lab.mit.edu" xsi:schemaLocation="http://i-lab.mit.edu
http://web.mit.edu/gviedma/Public/weblab/xml/experimentResult.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <datavector name="Real" units="None">
     +9.988314E-01,+9.988299E-01,+9.988285E-01,+9.988271E-01
   </datavector>
   <datavector name="Imaginary" units="None">
     -6.548476E-03,-6.627961E-03,-6.707444E-03,-6.786927E-03
   </datavector>
</experimentResult>
```

Once the Experiment Result has been parsed at the client, the frequency response of the experiment can be plotted and the data points can optionally be saved to a file.

Bibliography

[1] Victor Chang. Remote Collaboration in Weblab — An Online Laboratory. Master's thesis, Massachusetts Institute of Technology, May 2001.

[2] Enhydra. kSOAP Project. `http://ksoap.enhydra.org`. Date downloaded: May 15, 2004.

[3] Judson Harward. Service Broker/Lab Server API.

[4] Microsoft Developer Network. Accessing SQL Server from a Web Application. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/%html/vbconaccessingsqlserverfromwebapplication.asp`. Date downloaded: May 15, 2004.

[5] W3C. Extensible Markup Language (XML). `http://www.w3.org/XML/`. Date downloaded: May 15, 2004.

[6] W3C. Guide to the W3C XML Specification DTD, Version 2.1. `http://www.w3.org/XML/1998/06/xmlspec-report.htm`. Date downloaded: May 15, 2004.

[7] Brian Williams. A Graphical Package for Bode, Nichols and Nyquist Plots.

[8] XMLSchema. W3C XML Schema. `http://www.w3.org/XML/Schema`. Date downloaded: May 15, 2004.

[9] David Zych. Lab Client/Service Broker API.