MIT Integrated Microelectronics Device Experimentation and Simulation iLab

by

Boris Cukalovic

B.S., Computer Science and Engineering, 2005
B.S., Mathematics, 2005
Massachusetts Institute of Technology

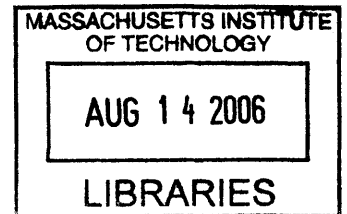Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author . . . .
Department of Electrical Engineering and Computer Science
May 26, 2006

Certified by . . . .
Jesús A. del Alamo
Professor of Electrical Engineering
Thesis Supervisor

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

MIT Integrated Microelectronics Device Experimentation and Simulation iLab

by

Boris Cukalovic

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We developed the MIT Integrated Microelectronics Device Experimentation and Simulation iLab, a new online laboratory that combines and significantly upgrades the capabilities of two existing online microelectronics labs: WebLab, a device characterization lab, and WebLabSim, a device simulation lab. The new integrated tool allows users to simultaneously run experiments on actual devices and simulations on the virtual ones, as well as to compare the results of the two. In order to achieve this, we considerably extended the capabilities of the original clients. We added the ability to graph the results of multiple experiments and simulations simultaneously, on top of each other, which allows for much easier comparison. We also added the ability to load, view and graph the results of experiments and simulations that were ran at any point in the past, even when the corresponding lab configurations are no longer available. Our hope is that this new integrated iLab will enrich microelectronics teaching and learning by allowing students to compare real life device behavior with theoretical expectations.

Thesis Supervisor: Jesús A. del Alamo
Title: Professor of Electrical Engineering

Acknowledgments

I would like to thank my supervisor, Jesús del Alamo, for mentoring me on this project, for providing invaluable feedback and insights, and for always having super-human patience for me. I would also like to thank James Hardison, Abhra Haldar, David Zych, and Adrian Solis for their contributions, advice, and, above all, time. Last, but certainly not least, my many thanks go to my academic advisor, Rodney Brooks, for looking after my best interests for the past five years.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Online microelectronic laboratories and simulators are tools of tremendous educational and practical value. An online microelectronics laboratory can allow a user located anywhere in the world to perform experiments on remotely stored devices. Although the user, in general, does need to posses the knowledge and the skills to operate the actual instruments, he does not have to be physically present in the lab - he just needs to issue a command and the software takes care of everything for him. In the case of potentially dangerous experiments, such as the ones involving high voltages, the spacial separation of the user from the actual equipment allows him to obtain the desired results without putting himself at risk [1, 2, 3].

Online microelectronic simulators are a similar and equally useful concept. First and foremost, these simulators are indispensable tools in engineering. They allow the engineers to test and verify their designs before fabricating the actual devices. This is crucial, since errors in design might be very expensive to fix, especially in the late stages of production. As for their educational value, microelectronic device simulators can allow a student to set up circuits and vary device parameters in ways that might be difficult or very time consuming to implement in practice. They also enable students to "operate" a virtual instrument under conditions that might be unsafe or able to destroy or damage the real device, such as extremely high voltages.

Finally, simulators can let us study devices that do not yet exist or that cannot exist at all, such as the ones with ideal characteristics[1] [4].

While the merits of online labs and simulators are clearly significant, especially in an educational setting, they stand to offer much more when combined together. An integrated tool that would allow users to simultaneously run experiments on real devices and simulations on the virtual ones, as well as to compare the results of the two could truly enrich microelectronics teaching and learning. In addition to having all the strengths of its components, such an integrated lab would enable students to compare real life device behavior with theoretical expectations.

1.2   Related Work

There is a wide range of online laboratories and simulators currently available. While some were developed by companies, often as support tools for their product lines, such as the National Semiconductor's WEBENCH Electrical Simulator [5], a large number were created at educational institutions, to serve as teaching, learning and research aids. Examples include the Virtual Chemistry laboratory at the University of Oxford [6], Alvin Simulator of a Deep Submergence Vehicle at the Woods Hole Oceanographic Institution [7], the Visual Perception Online Laboratory at Purdue University [8], and so on. Some of the systems were built on top of existing tools, such as LabView ("The Software That Powers Virtual Instrumentation" [9]). Many very successful ones, however, were entirely developed behind the university walls. This is, perhaps, the best recognition of the significance the online laboratories and simulators have in education.

MIT too has long recognized the potential of online experimentation tools. Its efforts in this area have in the recent times been concentrated within a major educational initiative called the iLabs project. Within this project, the MIT Microelectronics WebLab has been the single most successful online laboratory. It has been used as a learning tool by thousands of students at MIT and at numerous universities

---

[1]For example, an infinite gain op-amp.

throughout the world for almost a decade [1, 2, 3, 10].

1.2.1 MIT Microelectronics WebLab

The MIT Microelectronics WebLab, or WebLab for short, is an online microelectronics laboratory that interfaces an MIT based semiconductor parameter analyzer. It is a multi-tier Java application, built based on the iLabs' Batched Experiment Architecture (BEA) design template. WebLab allows users to remotely, through a web browser supporting Java applets, perform various experiments involving different microelectronic devices supported by the analyzer. The lab's development began in 1998 and since then this online laboratory has been successfully used in various projects by more than 3,600 EE students from MIT, National University of Singapore, Chalmers University of Technology in Sweden, etc [1, 2, 3, 10]. WebLab's overwhelming success inspired the creation of a sibling online microelectronics simulator – the MIT Device Simulation WebLab [4].

1.2.2 MIT Device Simulation WebLab

The MIT Device Simulation WebLab, or WebLabSim for short, is an online microelectronics simulator that simulates the behavior of various devices. Like WebLab, whose user interface it largely parallels, WebLabSim is a multi-tier Java application based on the BEA. It allows a user to remotely, through a Java-enabled browser, define experiments and submit requests for their execution. The requests are ultimately processed by a device simulator located at MIT server(s) and the results are sent back to the user [4]. WebLabSim currently uses the free WinSpice device simulator [11], but it is designed to support an arbitrary collection of such stand-alone simulators.

1.3 Goals of This Thesis

The main goal of this thesis was to create a system that combines the powers of a full-featured, stand-alone online microelectronics lab – WebLab – and a full-featured,

stand-alone online microelectronics simulator – WebLabSim. To this end, we have developed the MIT Integrated Microelectronics Device Experimentation and Simulation iLab.

The integrated lab is an online experimentation suite accessible through a Java Applet client that can be launched from any Java enabled browser. First and foremost, the lab allows users to specify and run various experiments on real devices as well as simulations on the virtual ones. The user has the ability to view and compare results of any of his experiments and simulations that successfully executed at any point in time. The comparisons can be achieved by graphing compatible sets of result data on top of each other. The graphing area is fully customizable and offers the users much freedom and capability.

1.4   Thesis Outline

This thesis is organized as follows. Chapter two describes the high-level architecture of the integrated lab and the factors that influenced our design decisions. Chapter three contains a description of the lab's user interface. It explains in detail the ways in which a user can interact with the lab. Chapter four focuses on the implementation of the lab's client. Finally, in chapter five we give the concluding remarks.

Chapter 2

MIT Integrated Microelectronics Device Experimentation and Simulation iLab Architecture

This chapter describes the high-level architecture of the MIT Integrated Microelectronics Device Experimentation and Simulation iLab. After analyzing the problem and the relevant background information, it presents the design that fully meets our project goals.

2.1 Problem Analysis

We wanted to create a lab that would seamlessly integrate WebLab and WebLabSim and that would be scalable, modular, extensible and able to run on any of the popular platforms. This task was simplified by the accomplishments of the iLab project and the efforts of people who developed WebLab and WebLabSim. First and foremost, out of the iLab project came a blueprint for how scalable online laboratories should be implemented: the iLab Batched Experiment Architecture, and both WebLab and WebLabSim were built based on it. Secondly, the WebLabSim user interface was based on and shared most of the code with the WebLab user interface. This, together with the fact that the software design and the implementation of the WebLab user

interface were result of years of development and continuous refinement, made some of our choices regarding the integration much easier and the actual implementation of them much more enjoyable.

2.2   iLab Batched Experiment Architecture

The iLab Batched Experiment Architecture (BEA) has been developed as part of the MIT's iLab project [12]. BEA specifies the details of how a scalable online laboratory should be designed and provides a part of the infrastructure expected to be common to most such systems. In particular, in this architecture an online laboratory's design is to be comprised of the following three tiers:

1. **Client application** is an application through which a user interacts with the online laboratory: sets up an experiment, sends it for execution, views results, etc.

2. **Laboratory server** is an application that accepts the experiment specification, runs the experiment, and returns the results.

3. **Service broker** is a trusted intermediary that mediates the requests and responses between the clients and the lab servers. In addition, a broker also provides user authentication and authorization, a web-based administrative interface, data storage, as well as some additional shared services. While the other two tiers of the Batched Architecture are created by the lab developers, the server broker is provided by the iLab project team.

The communication between the broker and the clients and servers is achieved via strictly defined set of web services (see Appendix A for a subset of these). It is implemented using SOAP calls, defined in WSDL (Web Services Description Language), over HTTP.

2.3 Design Overview of WebLab and WebLabSim

Both WebLab and WebLabSim were designed based on the Batched Experiment Architecture. Hence, they are both three tier systems, consisting of a client, a lab server and a service broker. The WebLabSim's client was based on that of WebLab and the two share most of their code. This code has been developed and gradually improved over WebLab's many years of active service. It is fairly modular, extensible and easy to read. Since it was written in Java, it is also system independent.

2.4 Design Overview of the Integrated Lab

The integrated lab follows the Batched Experiment Architecture blueprint. This design choice was a natural one. In fact, our project was perfectly suited for the BEA and serves as an important example of its power. Namely, thanks to the modularity and extensibility of BEA, and the clean separation between the three of its tiers, we were able to reuse much of the existing infrastructure of WebLab and WebLabSim and perform the integration with the least possible amount of effort. All that was needed was a new front-end, i.e. a new (integrated) client, that would communicate with the existing WebLab and WebLabSim servers through the, again existing, service broker.

2.4.1 Integrated Client

Our integrated client combines the capabilities of WebLab's and WebLabSim's clients and builds upon them to create a truly full-featured experimentation suite. It provides a rich graphical environment that, among other things, allows a user to set-up and execute both experiments and simulations, to load results and experiment/simulation descriptions of any old (successful) executions, as well as to plot multiple results simultaneously. The full description of the user interface can be found in Chapter 3.

Since both WebLab's and WebLabSim's clients were Java applet's that had much of high-quality code in common, it was natural for out integrated client to also be a Java applet and to reuse much of the existing implementation. We improved upon

17

the original code, making it more general and scalable, in order to support all the new features unique to the integrated client, as well as to simplify potential future add-ons. Some of the implementation details can found in Chapter 4.

2.4.2 Lab Servers

The integrated lab has two lab servers. However, we have not done any development on the server side – the two servers are the full, intact deployments of the original WebLab and WebLabSim lab servers. Again, this is one of the major strengths of the BEA architecture – the ability to reuse full modules.

2.4.3 Service Broker

Service Broker is the existing system provided by the iLab project and serves, as described earlier, to mediate all requests and responses between the client and the lab-side modules and provide user authentication, authorization, and registration etc.

Of course, as prescribed by the Batched Experiment Architecture, the three modules communicate via SOAP messages. The architecture of the integrated lab is summarized in Figure 2-1.
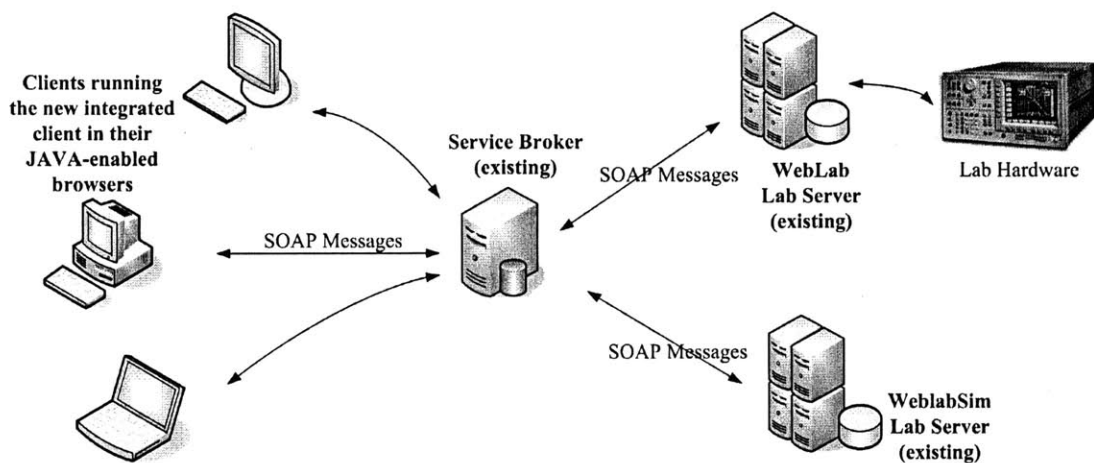
Figure 2-1: Overview of the Integrated Lab's Architecture

18

2.5 Summary

The MIT Integrated Microelectronics Device Experimentation and Simulation iLab is a three tier system, consisting of a client, lab servers, and a service broker, and is based on the iLab Batched Experiment Architecture. Thanks to the modularity of the BAE, we were able to reuse most of the components of WebLab and WebLab-Sim, and to perform the integration by only developing a new lab client. This is a perfect testament to the power of the iLab Batched Experiment Architecture, of its modularity and extensibility.

Chapter 3

Client User Interface

The client user interface is a Java applet. Together with the service broker, which a user must access in order to get authenticated and authorized, the client applet is the interface through which the user interacts with the integrated lab.

The client applet of our integrated lab was developed as an upgrade of the WebLab client applet, version 6.1, created by David Zych. In addition to being able to interface multiple lab servers at the same time, the integrated client has more general capabilities than its predecessor. It allows a user to plot multiple results on top of each other, has more advanced axis and plotting control, as well as the ability to load, view and plot results of old experiments. It also incorporates several cosmetic changes that make it more visually appealing.

3.1   Starting The Client

In order to start the integrated client, the user first needs to log into the service broker, using a Java-enabled internet browser. After he is successfully authenticated and authorized, the user needs to navigate to the screen that contains a button that launches the integrated client applet. Once the button is clicked, the browser downloads the applet and presents a prompt screen asking the user whether he trusts the downloaded content and wishes to run it. If the user answers affirmatively, the integrated client applet starts and displays its main window, shown in Figure 3-1.

Figure 3-1: Main window of the Integrated iLab client applet
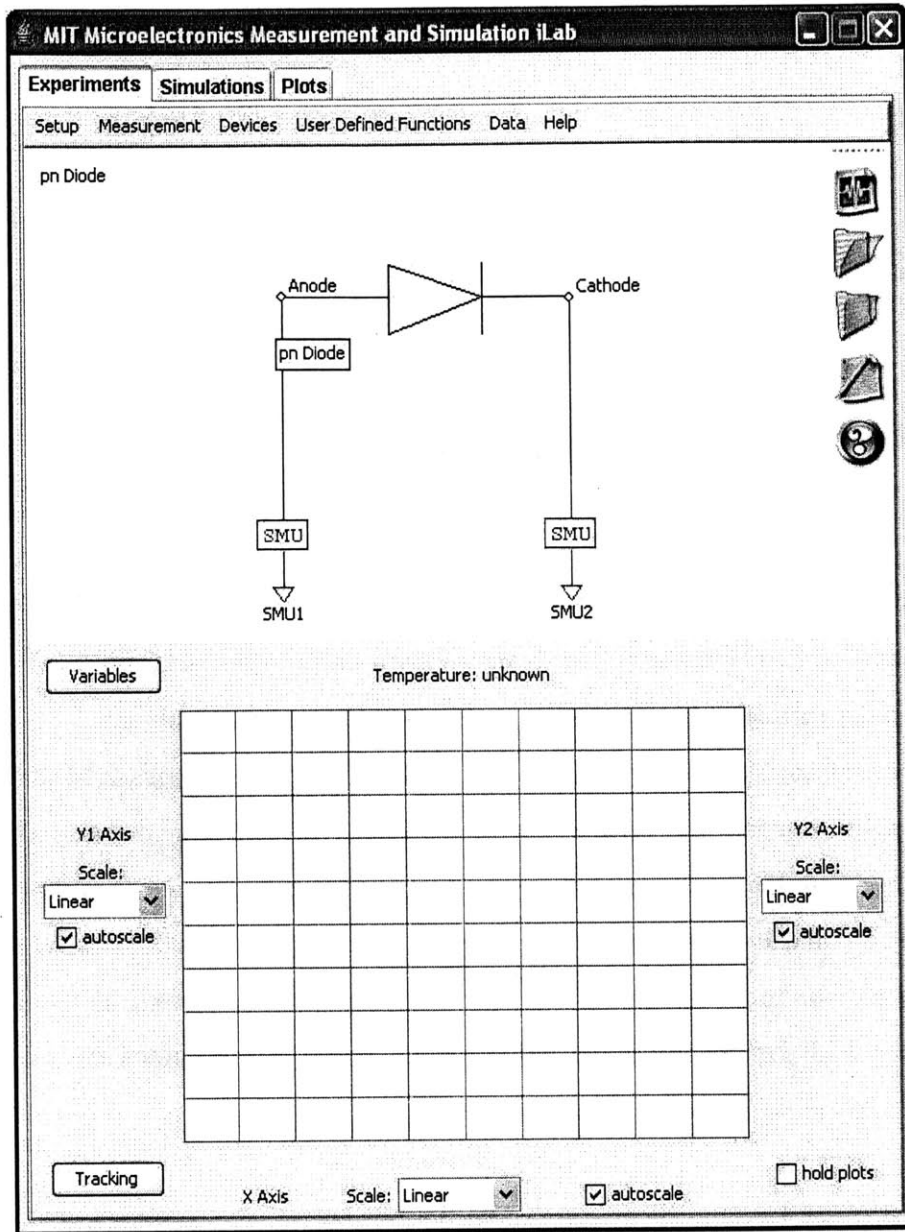
3.2   Applet Overview

The main window of the client applet consists of three panels, accessible through tabs: two *lab panels* and one *plots panel*. The lab panels correspond to Experiments and Simulations tabs and allow a user to set up and execute experiments and simulations, respectively, and to view, plot and download the results. The plots panel is accessible

through the Plots tab and allows a user to load previously executed experiments and simulations, to view the setups of those experiments/simulations, and to view, download and plot results.

3.3   Lab Panels

These two panels, accessed through Experiments and Simulations tabs, represent the "views" into WebLab and WebLabSim, respectively. They look exactly the same and have essentially the same capabilities, except that some features, depending on unique properties of WebLabSim devices, can only be used in the Simulations panel. The Experiments panel is the default one – i.e. it is the one in focus after the applet loads.

A lab panel consists of a menu bar and two subpanels: a *setup subpanel* and a *results subpanel*. Each subpanel takes up about half of the visible panel area. The menu bar has six menus: Setup, Measurement, Devices, User Defined Functions, Data, and Help.

The setup subpanel displays a schematic of the currently selected device, the name of the device and the current terminal configurations. Device, together with its terminal configurations, constitutes the current experiment setup. The setup subpanel also contains five action buttons that allow a user to: run an experiment, load a previously saved setup, save the current setup, access the user-defined functions dialog, and get help with the lab. If a mouse is hovered over an item in the setup subpanel, a tooltip box appears displaying information about the item.

The results subpanel contains the grid area, in which experiment and simulation results can be graphed. It also holds controls that can be used to select and customize what gets graphed and how.

3.3.1   Setting Up and Running Experiments

The user should verify that the Experiments panel is in focus. If it is not, he should correct this by clicking on the Experiments tab. Now, in order to prepare an exper-

iment, a user first needs to either create a new setup or load an existing one. If he chooses to create a new setup, the user first needs to select a device. This is done by clicking on one of the available items in the Device menu. The schematic of the chosen device will appear in the setup subpanel. Next, the user needs to specify terminal configurations. Dialogs in which such configurations can be defined are displayed when the user clicks on the image of a terminal. Once the setup is complete, the user can save it either by clicking on the Save Setup item in the Setup menu (Setup → Save Setup for short), or by clicking on the appropriate action button on the right side of the setup subpanel. Loading of a previously saved setup can be accomplished by Setup → Load Setup or by clicking on the appropriate action button. After a successful load the appropriate device schematic and terminal configurations are shown in the setup subpanel.

To send the current experiment to the lab server for execution, once the setup is complete, the user can either use Measurement → Run Measurement, or click on the appropriate action button. If there were no errors, after a short wait the client will receive the results. This is all described in more detail in the WebLab documentation [13].

3.3.2 Setting Up and Running Simulations

The user should verify that the Simulations panel is in focus. If it is not, he should correct this by clicking on the Simulations tab. The steps for preparing a new simulation, saving and loading a setup, and running the currently prepared simulation are identical to the ones for experiments, with one exception. Virtual devices, in general, allow a user to modify some of their parameters. The values of these parameters are considered to be part of a simulation setup, together with the device and the terminal configurations. A dialog in which the values of the parameters can be viewed and modified is displayed when the user clicks on the image of the device. For additional information about setting up terminal configurations, device parameters, as well as for a discussion of user-defined functions, see [4].

24

3.3.3 Plotting the Results

Once they are received, the results of experiments (simulations) can be plotted in the grid area of the results subpanel. The grid has one X axis and two Y axes: Y1 and Y2, which are all clearly labelled. In order to draw a y vs x graph, the user needs to select two variables (possibly identical) from a result: one to be associated with the X axis and one to be associated with a Y axis (Y1 or Y2). The variables are selected using the Available Variables dialog that is activated by clicking on the button Variables in the upper left corner of the results subpanel. An example of this dialog can be seen in 3-2.
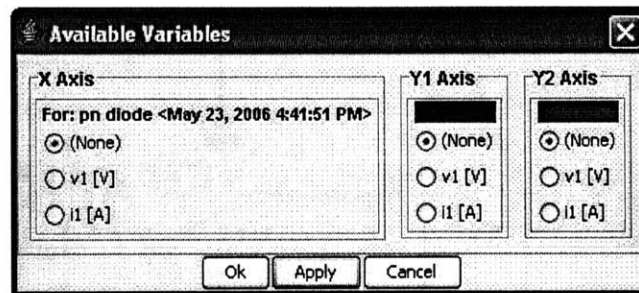

Figure 3-2: Available Variables dialog for a single experiment (simulation) result

The dialog holds a table that always has three columns – one for each axis. Each row of the table corresponds to a single result. In the above case there is only one result available, but, as we will see in the next section, there can be arbitrary many of them. To avoid confusion when there are multiple results, each row is labelled by the name of the experiment (simulation) that produced the corresponding result. In the example above, the name is pn diode <May 23, 2006 4:41:51 PM>. The naming conventions will be discussed in the section that describes the loading of old experiments and simulations.

Within each row, either a single variable or the option (None) can be selected in each column – i.e. for each result, at most one variable can be associated with a single axis. Now, lets suppose the user wants to graph i1 vs v1 in the above example, using Y1 as the Y axis. He needs to make an appropriate selection in the dialog and then

click on either Apply or Ok in order to initiate changes in the graphing area. This variable selection and the resulting graph are shown in Figure 3-3.
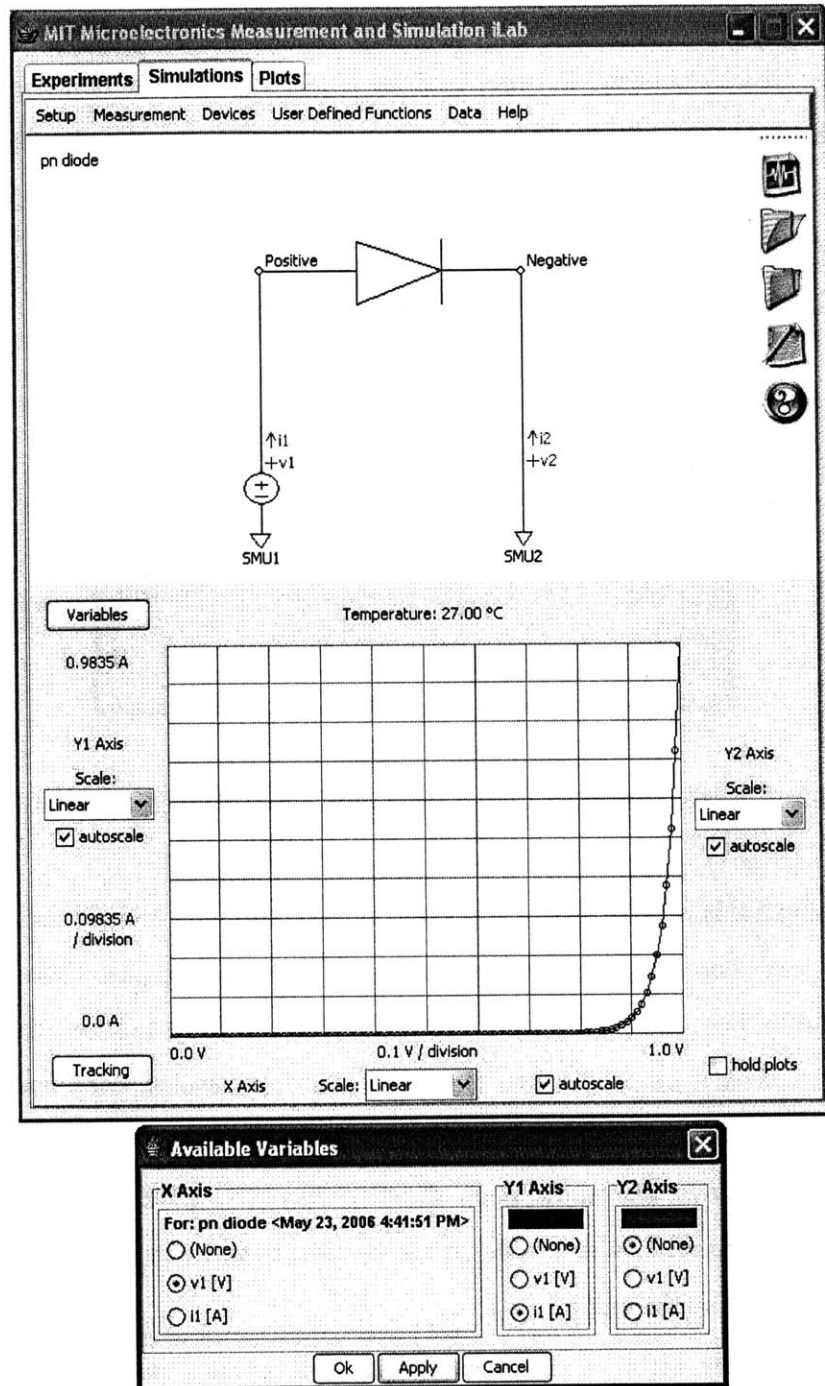

Figure 3-3: Plotting a graph based on a single result set

Holding Results

One of the major improvements to the original WebLab client is the ability to work with an arbitrary number of experiment (simulation) results and to plot them simultaneously in the same graphing area. This feature is toggled by the hold plots checkbox, located in the lower right corner of the results subpanel (see Figure 3-4).

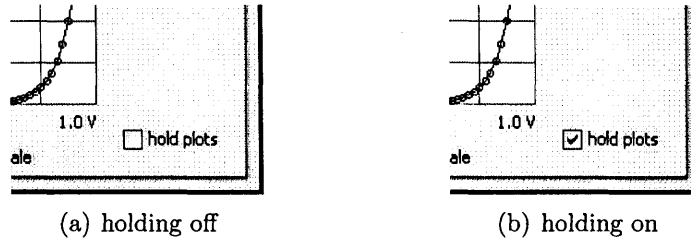

(a) holding off

(b) holding on

Figure 3-4: Checkbox that toggles holding of results

If the checkbox is not checked, receiving a new experiment (simulation) result replaces any previous ones. If the checkbox is checked, the new result is added to the set of currently held results, if any.

When there are multiple results available, the user can choose to plot graphs corresponding to any subset of these results. Plotting a graph for a single result is achieved by using the Available Variables dialog, as was described above. However, in the case of multiple results there is an additional restriction that all the variables associated with a single axis must have the same units. The units of (the physical quantity represented by) a variable are shown in square brackets immediately following the variable's name. A composite variable selection and the resulting graphs are shown in Figure 3-5.

Configuring the Axes

Each axis can be set to use either the linear or the logarithmic scale and, thus, linear, semi-logarithmic and logarithmic plots can all be produced. The scales of the axes can be set using the three Scale combo boxes, located around the grid.

Each axis also supports two modes of setting the range of values it spans within

27

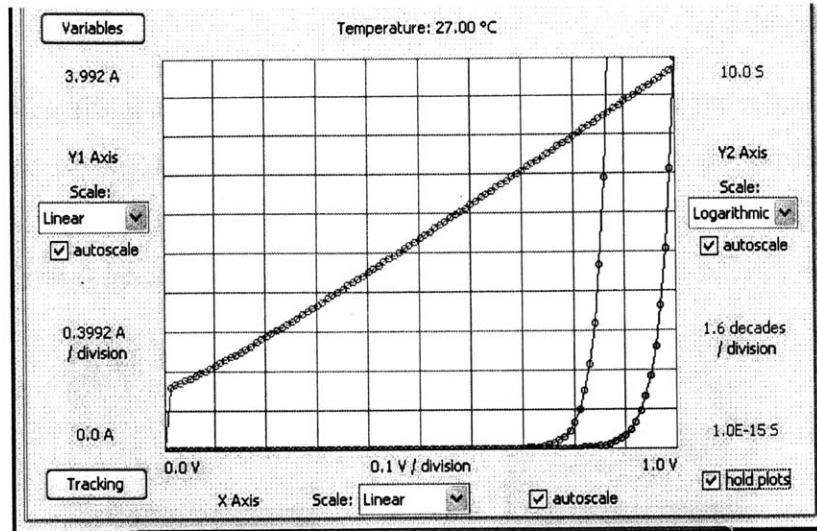Figure 3-5: Plotting graphs based on multiple result sets

the grid. Which mode is in use is determined by the state of the **autoscale** checkbox. If this checkbox is checked, the range is determined automatically, based on the minimum and maximum values of data sets currently associated with that axis. If the checkbox is not checked, two text fields appear and allow the user to specify the range himself.

Additional Features

As can be seen in Figure 3-5, each graph, or rather each pair $\{\mathtt{result}, \mathtt{Y}i\}$, $i \in \{1, 2\}$, has a color associated with it. The current color settings are shown by the colored rectangles located at the top of each row in Y1 and Y2 columns of the Available Variables dialog. The colors are initialized automatically by the client when a new result is loaded. The user can change any of them by clicking on the colored rectangle and choosing a new color in the color picker that gets displayed (Figure 3-6).
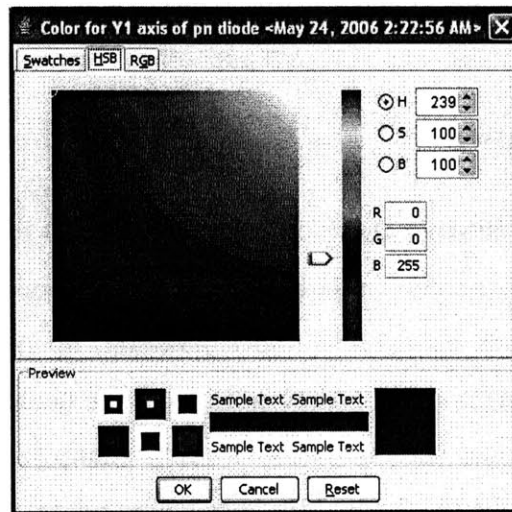
Figure 3-6: Color picker

By clicking on the Tracking button, located in the lower left corner of the results subpanel, the user brings up the Tracking dialog. When the mouse cursor is over the grid, this dialog shows the X, Y1 and Y2 coordinates of the grid point specified by the location of the cursor. If the Snap to data points checkbox in the dialog is checked, the cursor will snap to any currently plotted data point that it gets close enough to.

3.3.4 Other Operations

Data Operations

The user can view the data points from any currently held result by going to Data → View Data and clicking on the appropriate experiment (simulation) name. To save

these into a file of comma-separated values (CSV), he should go to Data → Download Data and, again, click on the desired experiment (simulation) name. All currently held results can be cleared by clicking on Data → Clear Data.

Additional Setup Operations

In addition to being able to save and load a setup, the user can also delete a previously saved one. The command for this is Setup → Delete Setup.... Finally, the command for resetting the current setup and starting a new one is Setup → New Setup (reset).

3.4   Plots Panel

This panel, accessible through the Plots tab, allows users to load previously saved, successfully executed experiments and simulations. Currently, the integrated client automatically saves each experiment and simulation, upon a successful execution. To each of these it assigns a name that consists of the name of the device that was involved in the experiment (simulation) and a time stamp (e.g. `pn diode <May 23, 2006 4:41:51 PM>`). In the future, the developers might choose to allow users to decide which experiments and simulations they want to save and under what name.

The Plots panel, shown in Figure 3-7, consists of a menu bar and two subpanels: *ex-setup subpanel* and a results subpanel, that each take up about half of the visible panel area. The menu bar has three menus: Results, Data, and Help.

The ex-setup subpanel is very similar to a setup subpanel. Once an old experiment or simulation gets loaded, this subpanel displays the setup that was used to specify that experiment or simulation. Recall that the setup includes the device, its terminal configurations, and, in case of simulations, the values of device parameters. The ex-setup subpanel can display the setups of arbitrary many loaded experiments and specifications. It organizes them using a tabbed pane, with one tab corresponding to one experiment (simulation) setup. Unlike the setup subpanel, the ex-setup one doesn't contain the action buttons. Rather, that window real estate is used to hold the navigation menu of the tabbed pane.
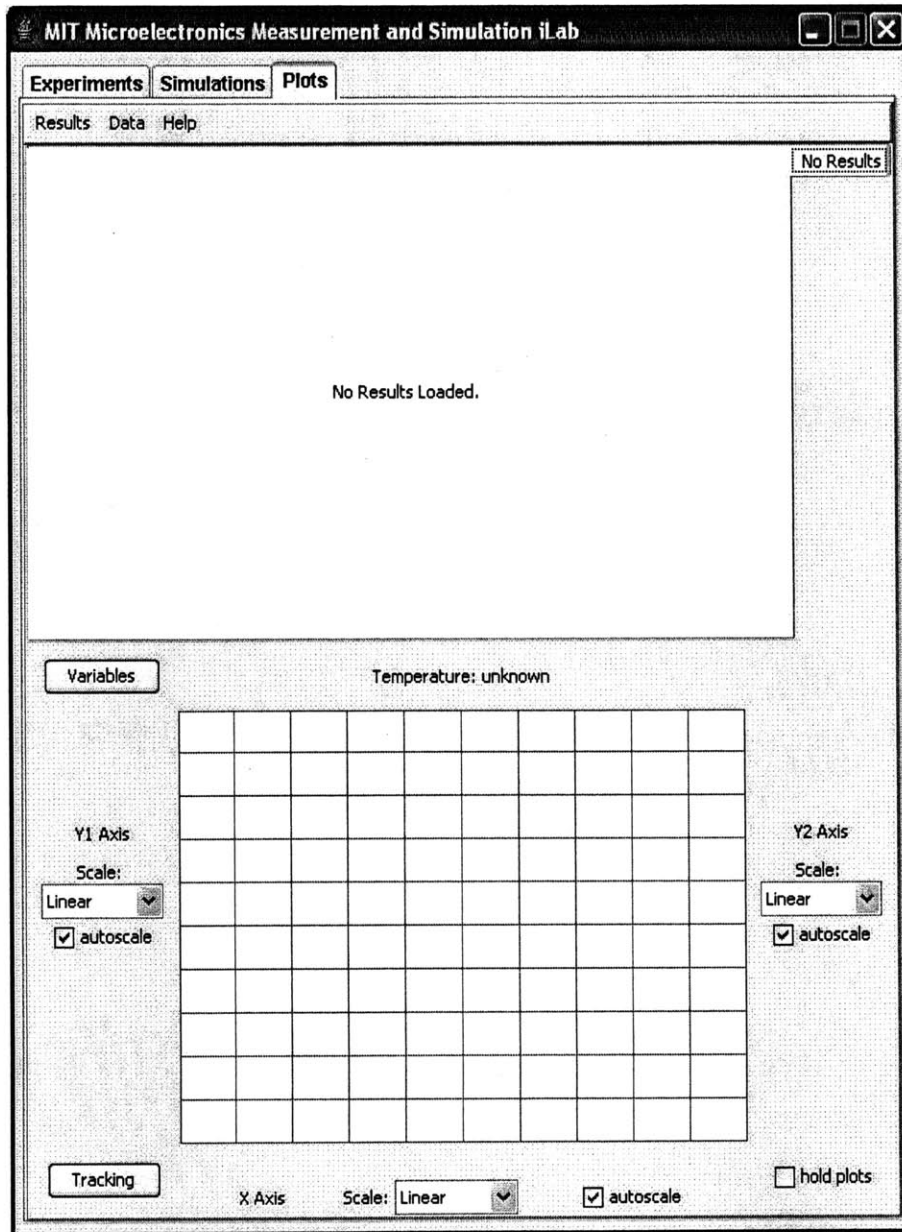
Figure 3-7: Plots panel of the Integrated iLab client applet

The results subpanel looks and functions exactly the same as the results subpanel described in the previous section.

3.4.1 Loading Old Experiment and Simulation Results

In order to load an old experiment (simulation), the user needs to click on Results → Load Experiment Result... (Results → Load Simulation Result...) and select an experiment (simulation) from the menu. Upon the load, an appropriate tab gets created in the ex-setup panel's tabbed pane. Like before, if the hold plots checkbox in the results subpanel was not checked, this last loading will remove the data and tabs corresponding to all previously loaded experiments and simulations. If the checkbox was checked, the newly loaded data will be added to the current collection, and the newly created tab will be added to the tabs currently in the tabbed pane. Figure 3-8 shows the tabbed pane after a few loads, with the checkbox being checked.
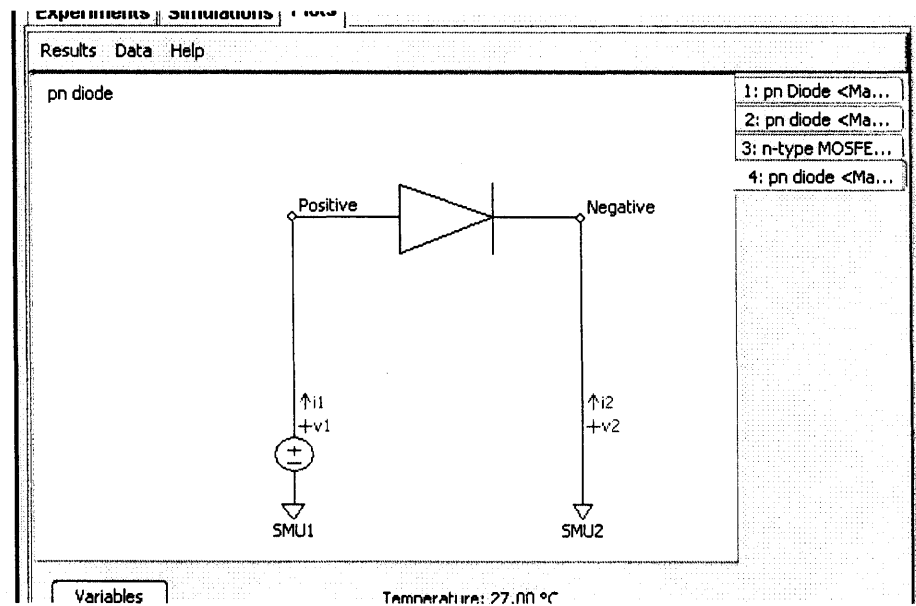

Figure 3-8: Plots panel after a few experiments and simulations have been loaded, with holding of results being on.

The tab in focus will be the one corresponding to the last loaded experiment (simulation). To bring a different tab forward, simply click on its corresponding slip in the tabbed pane's navigation menu on the right side of the ex-setup subpanel.

3.4.2 Plotting the Results

Procedure for plotting results, as well as all of the functions of the results subpanel are completely identical to the ones we have described in the section on Lab Panels. This is also the case with viewing, downloading and clearing of currently held result data.

3.4.3 Other Operations

In addition to being able to load an experiment (simulation), the user can also delete a previously saved one. The command for this is Results → Delete Experiment Result... (Results → Delete Simulation Result...).

3.5 Summary

The client of the Integrated iLab significantly upgrades the capabilities of its predecessor – the WebLab client. It allows the user to define and execute both experiments and simulations, as well as to load any experiment or simulation that the user successfully executed in the past. The integrated client can handle an arbitrary number of experiment and simulation results and, with certain consistency constraints, simultaneously plot any subset of the graphs that can be defined by these results. Finally, although much more powerful and feature-rich than the WebLab client, the integrated client manages to retain its predecessor's praised simplicity and ease of use.

Chapter 4

Client Implementation

The Integrated iLab client is a Java applet, implemented as an upgrade of the WebLab 6.1 client[1]. The Java classes comprising the client implementation can be divided into the following four parts: server interface, user interface, data/core logic, and the graphing engine[2]. In this chapter we will examine how each part was changed from the WebLab client.

4.1 Server Interface

The server interface is a facade through which the data/core logic talks to the server broker and, indirectly, to the lab servers. The main class in this group is SBServer. It implements the client to server broker API of the iLab Batched Experiment Architecture (Appendix A). Through the server interface classes, among other things, the client loads the current lab configurations for WebLab and WebLabSim, sends experiment and simulation specifications to the lab servers, and retrieves results. Lab configuration is the description of the current state of the lab: full information about all available devices, including their names, schematics, data about their parameters (if any), terminals, etc. Experiment specification is a detailed description of what we

---

[1]The WebLabSim client was itself based on the earlier, 6.0 version of the WebLab client, and we began the process of integration by first bringing the front-end to WebLabSim up to date.

[2]For consistency, we are using the terminology introduced in [4]. Note that here we refer to "service broker interface" part from [4] simply as "server interface."

called an experiment (simulation) setup in chapter 3.

The changes made to the server interface are detailed next.

4.1.1 Supporting multiple lab servers

Thanks to the existence of common, strictly defined interfaces, we did not have to make any modifications to classes in this group in order to support multiple lab servers. It was enough to instantiate two `SBServer` classes – one to talk to WebLab and the other to WebLabSim lab server.

4.1.2 Saving and loading of old experiments and simulations

For the sake of convenience and in order to be consistent with the actual implementation, from here onwards we will use the term "experiment" to refer to both an experiment on a real device and to a simulation on a virtual one. In other words, the experiment is the entity that gets specified on the client side and executed on the server side, regardless of whether that server is WebLab or WebLabSim. That said, several changes were necessary in the server interface in order to support the saving and loading of old experiments.

Firstly, we needed a way to save an experiment. Saving an experiment meant saving all the relevant information: current lab configuration, experiment specification and the experiment result. We made a decision to save only those experiments which execute successfully and return a result. Another decision, a fairly simple one, was that all the saved information was to be held on the service broker. The reason is that the service broker is the only data store that persists between every two lab sessions[3]. This also reduced the amount of data that we explicitly had to save, since the service broker was implemented to store all the three components: lab configuration, experiment specification and result automatically for each experiment. This data could later be retrieved from the service broker using the web service calls

---

[3]Information saved on a particular workstation would not be accessible whenever the user access the lab from a different machine.

`RetrieveLabConfiguration`, `RetrieveSpecification` and `RetrieveResult` (Appendix A).

Each of the above three web service calls takes a single argument: the experiment ID that uniquely identifies an experiment to the service broker[4]. Thus, one way to implement the saving and retrieving of experiments was to implement a scheme for saving and retrieving associated experiment IDs and to combine it with the aforementioned retrieve methods.

We solved the problem ofsaving an experiment ID by using the `SaveClientItem` web service call. This call takes two arguments: `key` and `value`, which can be arbitrary character strings. It allows a client to save the string `value` in a table, indexed by the string `key`. String value can later be retrieved by the client using the service call `RetrieveClientItem` with the argument `key`. This scheme is very similar to the one used by the client to save and retrieve experiment setups and, consequently, we had to make certain that the sets of keys used for these two tasks had no elements in common. All the keys used to index setup entries began with a prefix `savedSetup_`. Thus, by making all keys used to index ID entries begin with `savedResultID_` we insured that the two sets of keys were disjoint.

Finally, we also needed to save the name of the experiment[5] and to be able to tell whether the ID belongs to a WebLab experiment or a WebLabSim experiment. To store all this information, it was sufficient to make our key strings be a concatenation of `savedResultID_`, lab name (`WebLab` or `WebLabSim`) and the experiment name, and have the value strings be the experiment IDs. We also note that we used the web service call `RetrieveAllClientItems` in order to extract all available experiment IDs and the associated names, which was necessary in order to allow the user to choose which experiment to load.

---

[4]A service broker generates a new, unique experiment ID for each new experiment (experiment specification) that gets submitted by a client.

[5]Experiment names were discussed in chapter 3, section 3.3 and 4.

4.2 Data/Core Logic

The data/core logic is the main part of the client. It maintains the state of the client and bridges the gap between the user and the lab server, i.e. between the user and the server interfaces. For further description of this part of the WebLab and WebLabSim clients see [4].

The main changes made to the data/core logic are detailed next.

4.2.1 Abstractions for experiment specification

For WebLab and WebLabSim a single abstraction for the experiment specification was enough. The integrated client, however, has to deal with the fact that WebLab and WebLabSim have slightly different ways of describing experiment setups. To address this, we created three separate abstractions. `ExperimentSpecification` is an abstract class that stores the experiment specification data and specifies the methods through which other classes can access and interact with this data. `ExperimentSpecification` is subclassed by the `WeblabExperimentSpecification` and the `WeblabsimExperimentSpecification`. These last two classes implement those methods of `ExperimentSpecification` whose behavior is specific to either lab.

We note that a single abstraction, `LabConfiguration`, was enough to represent lab configurations of both WebLab and WebLabSim, due to them being sufficiently similar.

4.2.2 Abstraction for the experiment

We created the `Experiment` abstraction that aggregates all the information needed to fully specify an experiment: experiment name, lab configuration, experiment specification and the result.

4.2.3 Abstraction for a collection of experiments

`ExperimentCollection` is a new abstraction we created to represent the set of `Experiments` currently available. Unlike WebLab and WebLabSim, which have to store only one experiment at a time, the integrated client has to be able to store an arbitrary number. The client creates three instances of this class: one for each of the three main panels (Experiments, Simulations, and Plots).

4.2.4 Abstraction for a lab client

Similarly as with `ExperimentSpecification`, in WebLab and WebLabSim a single class was enough to manage the direct interaction between the user interface and the server interface layers and to store current lab configuration, experiment specification and result. Now, however, in order to achieve clean separation between different data abstractions, one class was not enough. Again, we implemented three related classes – `LabClient`, `WeblabClient` and `WeblabsimClient`. The first one of them specifies the common interface and implements the methods that are not domain specific. Domain specific client classes – `Weblab-` and `WeblabsimClient` – contain the domain specific information. `WeblabClient`, for example, knows that it has to use `WeblabExperimentSpecification` to encapsulate an experiment specification, it know that its name is "WebLab" (used when saving experiment IDs, as described in the section on server interface), etc. The analogous is true for `WeblabsimClient`.

Unlike in WebLab and WebLabSim, where the lab client abstraction used to provide experiment data to the user interface layer, in the integrated client this function is delegated to the `ExperimentCollection`. As soon as a new experiment result is received from the server interface, the new `Experiment` is added to the `ExperimentCollection`. That is, it is added if and only if the collection is "interested" in that `Experiment`. Namely, an `ExperimentCollection` can be set to gather either only newly executed `Experiments` (such as the collections used in the Experiments and Simulations panels), only loaded ones (such as the collection associated with the Plots panel), or both (this feature is not currently used).

The communication between the `LabClient` and the `ExperimentCollection`, as well as between the `ExperimentCollection` and the `UI` layer, is achieved using the same mechanism as in the predecessor labs: the Observer design pattern.

Finally, we note that the `LabClient` abstraction was also extended to support new functionality: loading and saving of `Experiments`.

4.3   User Interface

User interface classes implement the graphical environment that the user sees and uses to interact with the lab. One of the main changes to this part of the client was mentioned in the previous section. Namely, as opposed to drawing experiment information from the `LabClient` abstraction, the user interface classes now rely on an `ExperimentCollection`. The remaining changes made to the user interface are not particularly interesting, from a conceptual point of view, so we merely list them, without going much into detail.

Most obvious changes include the new tabbed pane with the three tabs described in chapter 3: Experiments, Simulations, and Plots tabs. In implementing these, we have achieved a high degree of code-reuse: Experiments and Simulations tabs are both implemented by the same `LabPanel` Java class, results subpanel abstraction in the Experiments (Simulations) tab is also used in the Plots tab, the setup panel and the tabs in the the ex-setup panel (see Chapter 3) have much of the code in common, etc.

The results panel itself has seen several significant changes. These include the hold plots checkbox, the Variables button and dialog, as well as the introduction of modifiable result/axis-dependent colors.

Among other contributions, we emphasize the new, higher quality icons for the action buttons of the setup subpanel, a new loading screen, as well as various bug fixes.

4.4   Graphing Logic

The classes in this part are responsible for graphing the results. The changes we made mainly involved implementing the support for handling multiple variables and for plotting of multiple, differently colored graphs.

4.5   Summary

In order to integrate the WebLab and WebLabSim clients and implement the new features, we had to perform a substantial overhaul of the original code. While the user interface got a major face-lift, the most interesting changes were made to the server interface and to the core/data logic parts of the client. In particular, we extended the range of requests that the server interface can send to the service broker and redesigned and expanded the set of abstractions that comprise the core of the clients business logic. Through carefully planned and executed modifications, we managed to achieve a significant increase in client capabilities, while producing only a limited increase in complexity.

Chapter 5

Conclusion

We have designed and implemented the MIT Integrated Microelectronics Device Experimentation and Simulation iLab, an integrated laboratory environment that combines and qualitatively exceeds the experimental capabilities of WebLab and the simulation capabilities of WebLabSim. The Integrated iLab is a three-tier application based on the iLabs Batched Experiment Architecture. Exploiting the design strengths of BEA – its modularity and simple, clean interfaces – we were able to reuse most of the components of WebLab and WebLabSim, and to create an entirely new iLab by developing only a new client application.

The client application was built as an upgrade of the highly successful WebLab 6.1 client. In addition to allowing a user to run both experiments on actual devices and simulations on the virtual ones, it offers important novel features of potentially significant educational impact. In particular, the client has the ability to simultaneously plot and manipulate any number of compatible graphs of experiment and simulation results, thus facilitating easy comparisons of real life device behavior and theoretical expectations. The client also provides the option of loading, viewing and graphing the results of experiments and simulations that were ran at any point in the past.

The MIT Integrated Microelectronics Device Experimentation and Simulation iLab fulfilled the goals of the project in their entirety. However, it is not without rough edges or room for improvement. In the future, we hope to add the following

capabilities: letting the user name and rename new and saved experiments, letting the user choose which experiments to save, being able to select a data vector to use for autoscaling, having the tracking dialog display more information in presence of multiple results, etc. We hope that this list will continue to grow as we start testing the client in a classroom setting and getting the feedback from the students.

Appendix A

iLab Client to Service Broker API

The following descriptions of web service calls was taken from a document written by David Zych and last modified on January 22, 2005 by Chris Felknor [14].

Cancel

Purpose:

```
/* Cancels a previously submitted experiment. If
    the experiment is already running, makes best
    efforts to abort execution, but there is no
    guarantee that the experiment will not run to
    completion. */
```

Arguments:

```
int experimentID
/* A token that identifies the experiment.  */
```

Returns:

```
bool cancelled
/* true if experiment was successfully removed from
    the queue (before execution had begun). If false,
```

user may want to call GetExperimentStatus() for more detailed information. */

GetEffectiveQueueLength

Purpose:

/* Checks on the effective queue length of the lab server. Answers the following question: how many of the experiments currently in the execution queue would run before the new experiment? */

Arguments:

string labServerID

/* A token that identifies the lab server. */

int priorityHint

/* Indicates a requested priority for the hypothetical new experiment. Possible values range from 20 (highest priority) to -20 (lowest priority); 0 is normal. Priority hints may or may not be considered by the lab server. */

Returns:

waitEstimate WaitEstimate

public struct waitEstimate
{
    public int effectiveQueueLength

    /* Number of experiments currently in the execution queue that would run before the hypothetical new experiment. */

```
    public double estWait
    /* [OPTIONAL, < 0 if not supported] estimated wait
       (in seconds) until the hypothetical new experiment
       would begin, based on the other experiments currently
       in the execution queue. */
}
```

GetExperimentStatus

```
Purpose:
    /* Checks on the status of a previously submitted
       experiment.  */


Arguments:
    int experimentID
    /* A token that identifies the experiment. */


Returns:
    LabExperimentStatus experimentStatus
    /* See description below. */


public struct LabExperimentStatus
{
    public ExperimentStatus statusReport
    /* See description below. */


    public double minTimeToLive
    /* Guaranteed minimum remaining time (in seconds) before
       this experimentID and associated data will be purged from
       the lab server. */
```

```
}


public struct ExperimentStatus

/* indicates the status of this experiment. */

{
     public int statusCode

     /* 1 if waiting in the execution queue, 2 if currently

        running, 3 if terminated normally, 4 if terminated with

        errors (this includes cancellation by user in mid-

        execution), 5 if cancelled by user before execution had

        begun, 6 if unknown experimentID. */


     public WaitEstimate wait

     /* Described on P. 12. */


     public double estRuntime

     /* [OPTIONAL <0 if not used] estimated runtime (in seconds)

        of this experiment. */


     public double estRemainingRuntime

     /* [OPTIONAL < 0 if not used] estimated remaining runtime

        (in seconds) of this experiment, if the experiment is

        currently running. */

}
```

GetLabConfiguration

Purpose:

```
     /* Gets the configuration of a lab server.  */
```

Arguments:

```
string labServerID

/* A token identifying the lab server.  */


Returns:

string labConfiguration

/* An opaque, domain-dependent lab configuration.  */
```

GetLabInfo

```
Purpose:

/* Gets general information about a lab server.  */


Arguments:

string labServerID

/* A token identifying the lab server.  */


Returns:

string URL

/* A URL to a lab-specific information resource, e.g. a
   lab information page.  */
```

GetLabStatus

```
Purpose:

/* Checks on the status of the lab server.  */


Arguments:

string labServerID

/* A token that identifies the lab server.  */


Returns:
```

```
    LabStatus labStatus

    /* See description below. */


public struct LabStatus

{

    public bool online

    /* true if lab is accepting experiments. */


    public string labStatusMessage

    /* Domain-dependent human-readable text describing status

        of lab server. */

}
```

RetrieveResult

```
    Purpose:

        /* Retrieves the results from (or errors generated by) a

            previously submitted experiment.  */


    Arguments:

        int experimentID

        /* A token identifying the experiment.  */


    Returns:

        ResultReport resultReport

        /* See description below. */


    public struct ResultReport

    {

        public int statusCode

        /* Indicates the status of this experiment. 1 if waiting
```

in the execution queue, 2 if currently running, 3 if
terminated normally, 4 if terminated with errors (this
includes cancellation by user in mid-execution), 5 if
cancelled by user before execution had begun, 6 if
unknown experimentID. */

public string experimentResults
/* [REQUIRED if experimentStatus == 3, OPTIONAL if
experimentStatus == 4] an opaque, domain-dependent set
of experiment results. */

public string xmlResultExtension
/* [OPTIONAL, null if unused] a transparent XML string
that helps to identify this experiment. Used for
indexing and querying in generic components which can't
understand the opaque experimentSpecification and
experimentResults.*/

public string xmlBlobExtension
/* [OPTIONAL, null if unused] a transparent XML
string that helps to identify any blobs saved as part
of this experiment's results. */

public string[] warningMessages
/* Domain-dependent human-readable text containing
non-fatal warnings about the experiment including
runtime warnings. */

public string errorMessage
/* [REQUIRED if experimentStatus == 4]

domain-dependent human-readable text describing why
the experiment terminated abnormally including runtime
errors. */

}

Submit

Purpose:

/* Submits an experiment specification to the lab server
for execution. */

Arguments:

string labServerID

/* A token identifying the Lab Server. */

string experimentSpecification

/* An opaque, domain-dependent experiment specification. */

int priorityHint

/* Indicates a requested priority for this
experiment. Possible values range from 20 (highest
priority) to -20 (lowest priority); 0 is normal.
Priority hints may or may not be considered by the
lab server. */

bool emailNotification

/* If true, the service broker will make one attempt to
notify the user (by email to the address with which the
user's account on the service broker is registered) when
this experiment terminates. */

Returns:

    ClientSubmissionReport clientSubmissionReport

    /* See Description Below. */

public struct ClientSubmissionReport

{

    public ValidationReport vReport

    /* See description below, under Validate. */

    public int experimentID

    /* The experiment ID. */

    public double minTimeToLive

    /* Guaranteed minimum time (in seconds, starting now) before

       this experimentID and associated data will be purged from

       the lab server. */

    public WaitEstimate wait

    /* See an earlier description */

}

Validate

Purpose:

/* Submits an experiment specification to the lab server for

   execution. */

Arguments:

    string labServerID

    /* A token identifying the lab server. */

```
        string experimentSpecification

        /* An opaque, domain-dependent experiment specification. */


Returns:

        ValidationReport validationReport

        /* See description below. */


public struct ValidationReport

{

        public bool accepted

        /* true if the experiment specification would be (is) accepted

            for execution. */


        public string[] warningMessages

        /* Domain-dependent human-readable text containing non-fatal

            warnings about the experiment. */


        public string errorMessage

        /* [If accepted == false] domain-depend human-readable text

            describing why the experiment specification would not be

            accepted. */


        public double estRuntime

        /* [OPTIONAL, < 0 if not supported] estimated runtime (in

            seconds) of this experiment. */


}
```

DeleteClientItem

Purpose:

    /* Removes a client item from the user's opaque data
       store. */

Arguments:

    string name
    /* The name of the client item to be removed. */

Returns:

    void none ListAllClientItems

Purpose:

    /* Enumerates the names of all client items in the user's
       opaque data store. */

Arguments:

    none

Returns:

    string[] clientItems
    /* An array of client items. */ LoadClientItem

Purpose:

    /* Returns the value of a client item in the user's
       opaque data store. */

Arguments:

    string name

/* The name of the client item whose value is to be
    returned. */

Returns:

    string clientItemValue

    /* The value of a client item in the user's opaque
        data store. */

SaveClientItem

Purpose:

    /* Sets a client item value in the user's opaque data
        store. */

Arguments:

    string name

    /* The name of the client item whose value is to be
        saved. */

    string itemValue

    /* The value that is to be saved with name. */

Returns:

    void none

Bibliography

[1] J. A. del Alamo, L. Brooks, C. McLean, J. Hardison, G. Mishuris, V. Chang, and L. Hui, "The MIT Microelectronics Weblab: a web-enabled remote laboratory for microelectronic device characterization," in *World Congress on Networked Learning in a Global Environment*, Berlin, Germany, 2002.

[2] J. A. del Alamo, L. Brooks, C. McLean, J. Hardison, G. Mishuris, V. Chang, and L. Hui, "Educational experiments with an online microelectronics laboratory," in *International Conference on Engineering Education*, Manchester, UK, 2002.

[3] J. A. del Alamo, L. Brooks, C. McLean, J. Hardison, G. Mishuris, V. Chang, and L. Hui, "MIT Microelectronics WebLab," in *Lab on the Web: Running Real Electronics Experiments via the Internet* (T. A. Fjeldly and M. S. Shur, ed.), pp. 49–87, New Jersey: Wiley-IEEE, 2003.

[4] A. Solis, "MIT Device Simulation WebLab: An Online Simulator for Microelectronic Devices," Master's thesis, Massachusetts Institute of Technology, 2004.

[5] National Semiconductor Corporation, "WEBENCH Electrical Simulator." http://webench.national.com/appinfo/power/webench/elec_sim.html, May 2006.

[6] University of Oxford Department of Chemistry, "Virtual Chemistry." http://www.chem.ox.ac.uk/vrchemistry/, May 2006.

[7] Woods Hole Oceanographic Institution Fraunhofer Center for Research in Computer Graphics, "Alvin Simulator." http://www.whoi.edu/marops/vehicles/alvin/alvinsim/index.html, May 2006.

[8] Purdue University, "Visual Perception Online Laboratory." http://www2.psych.purdue.edu/~coglab/VisLab/welcome.html, May 2006.

[9] National Instruments, "LabVIEW." http://www.ni.com/labview/, May 2006.

[10] K. O. Jeppson, P. Lundgren, J. A. del Alamo, J. L. Hardison, and D. Zych, "Sharing online laboratories and their components: A new learning experience," in *5th European Workshop on Microelectronics Education*, Lausanne, Switzerland, April 2004.

[11] M. Smith, "WinSpice." http://www.winspice.com, May 2006.

[12] V. J. Harward, J. A. del Alamo, V. S. Choudhary, K. deLong, J. L. Hardison, S. R. Lerman, J. Northridge, D. Talaver, C. Varadharajan, S. Wang, K. Yehia, and D. Zych, "iLab: A scalable architecture for sharing online experiments," in *International Conference on Engineering Education 2004*, Gainesville, Florida, 2004.

[13] MIT WebLab Group, "WebLab Documentation." http://weblab2.mit.edu/docs/weblab/v6.1/manual/, May 2006.

[14] D. Zych, J. Harward, and C. Felknor, "Client to Service Broker API." http://icampus.mit.edu/ilabs/architecture/content/?iLabsFebRelease, 2005.

2233. 20