Expanding the Capabilities of the ELVIS iLab Using Component Switching

by

Bryant J. Harrison
S.B., Electrical Engineering and Computer Science, M.I.T., 2007

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 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

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

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

Expanding the Capabilities of the ELVIS iLab Using Component Switching

by

Bryant J. Harrison

S.B., Electrical Engineering and Computer Science, M.I.T., 2007

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

iLabs are online laboratories that allow users to access, control, and perform real experiments remotely through the Internet. Users are able to access laboratory experiments whenever and wherever they want, bypassing the problem of acquiring expensive equipment and waiting in long queues to use the equipment. iLabs allow students to complement their theoretical calculations and results with real data, providing them with a better understanding of engineering concepts. The ELVIS iLab was developed using the National Instruments Educational Laboratory Virtual Instrumentation Suite, a low cost, all-in-one electronics workstation that can be software controlled. ELVIS iLab is currently in the second version and this thesis explains the modifications made to the first version to add an additional power supply and switching capabilities. These changes increase the quantity, flexibility, and variety of experiments that can be created and provides the possibility of more challenging assignments for students. It also facilitates easier sharing between courses and institutions.

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

Acknowledgments

I would like to thank my thesis supervisor, Prof. Jesus del Alamo, for giving me the opportunity to work on this great project. Over the years, Prof. del Alamo has been very supportive of my extracurricular endeavors that involve education and technology in the developing world that ultimately enabled me to work on the iLab Africa project. I feel very fortunate to have the opportunity to work on something meaningful to me that has a significant impact on people across the world. The experience has been rewarding and has had a strong influence on my future career path.

I am extremely grateful for all of the assistance and support over the years from Kimberly DeLong, James Hardison, Judson Harward, Adnaan Jiwaji, David Zych, and the entire MIT iLabs team. I would also like to thank Samuel Gikandi, my predecessor on the ELVIS iLab, for his guidance early in the project. Their help has been critical to bringing my work into a reality.

Finally, I would like to thank the teams at Obafemi Awolowo University, University of Dar es Salaam, and Makerere University for their help and guidance throughout the project. The ideas and challenges they presented always kept me motivated and inspired. They also provided incredible hospitality during my visits to their campuses and made me feel at home, allowing me to focus on our work while traveling.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Any well-rounded science or engineering education will provide students with experience in theoretical as well as practical applications of concepts. To achieve the learning of the practical aspects, students must have access to laboratory equipment to conduct experiments. However, often students cannot get the necessary experience in traditional laboratories due to several limiting factors.

Laboratories require high initial startup costs, significant maintenance costs, and manpower to staff the facilities. This results in laboratories being underequipped and understaffed, and in the worst cases, non-existent. This leads to overcrowding of facilities or absence of experimental assignments, and students fail to get the understanding of the difference between theory and reality. Some equipment may be fragile, complicated, or burdensome, making the setup of the equipment overshadow the actual learning of the concepts. Students also have free reign to use the equipment as they want, opening the possibility of them damaging equipment if they input incorrect or excessive parameters.

Furthermore, traditional laboratories are inefficient and the equipment is often underutilized because they are limited to use during times when the facilities are open. This is often not when students prefer to work. Also, some equipment may only be useful for a few assignments in a course, and either goes unused outside of the time periods it is needed for a course or is not purchased in the first place because costs exceed the benefits and available funding.

1.1 Background on iLabs

iLabs is an attempt to bridge the challenges and inefficiencies of using traditional laboratories. Conceived at MIT in 1998, iLabs is the concept that the traditional laboratory experience can be virtualized and aspects of the traditional laboratory can be accessed remotely through the Internet. Laboratory equipment communicates with a computer that works as a server and the user opens a web-based client to control the equipment. The user interface looks like simulation software, but the user defined parameters are actually communicated to the equipment, the experiment is run on-demand, and real data is transmitted back to the user.

There are several benefits of iLabs over traditional laboratories. Since the laboratories are remotely accessed by students, they do not need to be physically in the laboratory. They can do their labs at their own convenient location and can access them at any time of the day. Students do not need to spend unnecessary time waiting for equipment availability, as the system automatically queues student requests on a first-come, first-served basis, and each experiment just takes seconds to execute. Students do not need to spend hours debugging or working with faulty components and equipment. iLabs greatly reduces the cost of a laboratory, because only one piece of equipment needs to be acquired and many users can share it. The sharing can expand beyond a campus and be shared across the world to students at other institutions. Institutions can create their own laboratories based on their curricula and different learning and teaching styles, and strengths and share them elsewhere. For example, a university with an excellent nuclear engineering department can develop a laboratory and share it with a university with a less endowed department, increasing the educational opportunities of those students.

iLabs is not perfect a replacement for a traditional laboratory. Students still do not get to experience the touch and feel of equipment or gain the valuable skills of setup configuration and debugging. iLabs are best suited as a supplement to exercises that may still include hands-on laboratory assignments or as a replacement in situations where no other options are available.

Over the past 10 years, several iLabs have been developed at MIT (Figure 1-1). They have ranged from civil engineering to electrical engineering and are used in courses in the undergraduate and graduate levels.

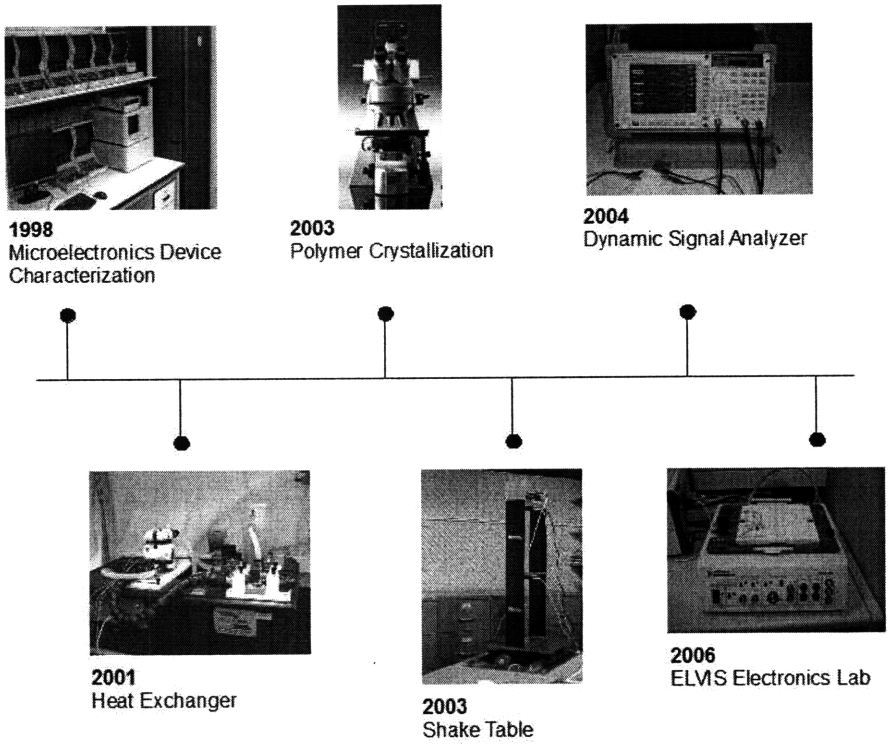


Figure 1-1: Timeline for deployment of iLabs.

iLabs have been used in 18 universities throughout Africa, Asia, Australia, Europe, and the United States. Several of these universities are developing their own unique iLabs. With the help of these universities, the iLabs concept is spreading throughout the world as an important teaching and learning tool. [4]

1.2 Background on iLab Africa Project

One major partnership of the MIT iLab project has been with universities in Africa. Starting in 2005, the MIT iLab project, in conjunction with the Carnegie Corporation of New York, formed a partnership with three African universities: Makerere University (MUK) in Kampala, Uganda, Obafemi Awolowo University (OAU) in Ile-Ife,

Nigeria, and University of Dar es Salaam (UDSM) in Dar es Salaam, Tanzania. [1]

This partnership came out of the finding that the iLabs technology is suitable for use in sub-Saharan Africa and that it can help alleviate some of the problems that are faced in academic institutions. These findings were exposed during a feasibility study conducted between 2003 and 2004 that sought to determine whether iLabs could be useful in sub-Saharan Africa and what challenges existed in utilizing and developing the technologies.

The results showed that there was a definite opportunity for iLabs technology to be utilized effectively. Many of the advantages of the iLab system matched well with the needs and limitations of the universities. Due to limited resources, laboratory equipment is scarce in many universities and students are often robbed of experience with experimentation as a result. With iLab, a university can purchase one piece of equipment or just access equipment housed at MIT so that scores of students can perform experiments. The study found that the curricula matched well with the concepts being taught with existing iLabs for courses like electrical engineering and physics, so iLabs could be used immediately without much modification. The flexibility and relative ease of changing experiment setups also allows for universities to match iLabs to their curricula even further. Faculty and students showed enthusiasm and desire for iLabs.

However, there were several barriers identified as well. Because of East Africa's lack of a high bandwidth fiber optic cable connection to the Internet, countries rely on slow and expensive connections to the Internet via satellite. [9] As a result, accessing the iLabs located at MIT is not ideal due to the bandwidth constraints. Although an iLabs setup only requires one piece of laboratory equipment for a class of students, this can still be cost prohibitive as some components can cost on the order of tens of thousands of dollars. To work around these issues, several modifications need to be developed to the traditional iLabs system. Integrating lower cost equipment that can still support basic functionality is one solution to the price issue. By deploying iLabs components on the high speed local campus networks, connection to the Internet is less of a limiting factor. [2]

While there is a general curriculum match and interest from faculty and students, there needs to be deeper involvement to ensure the laboratories meet the universities' pedagogical needs and there is solid collaboration and exchange of ideas. The iLab-Africa partnership was formed with three main goals in mind to achieve success in the integration of iLabs into sub-Saharan Africa. First, the utilization of iLabs housed at MIT in courses in the partner universities. By using existing laboratories, iLabs could have an immediate impact in teaching and learning, while having minimal setup and investment costs. The second goal is to create and develop iLabs for the African universities. These new iLabs would be developed by teams of faculty, students, and staff from the universities, and will address the specific needs of the university's curriculum. This will create an international research network with localized hubs across Africa. The final goal is to foster the development and use of iLabs through student and staff exchanges. Since 2004, these exchanges have taken place multiple times between all four universities resulting in training, exchanging of ideas, discussion of best practices, and a better understanding of the needs, capabilities, and challenges of each university.[2]

1.3 Background on National Instruments ELVIS

The iLab-Africa teams have been primarily focusing on electronics laboratories using the National Instruments Educational Laboratory Virtual Instrumentation Suite (NI ELVIS) as the base technology for the development of new iLabs (Figure 1-2). ELVIS is an all-in-one electronics platform that combines twelve instruments that can perform measurement and signal generation functions. These instruments include an oscilloscope, digital multimeter, function generator and an arbitrary waveform generator. A removable prototyping board is connected to the device to allow users to wire circuits and connect the various instruments. The platform can be controlled like traditional electronics equipment by using hardware knobs and switches or by connecting ELVIS to a computer and using a LabVIEW based software suite that is provided (see Figure 1-3). Because it is supported in LabVIEW, programmers have

access to many virtual instruments that can represent actual pieces of hardware. The hardware expands beyond just the controls of the ELVIS system as LabVIEW can integrate well with many other National Instruments products. [8]

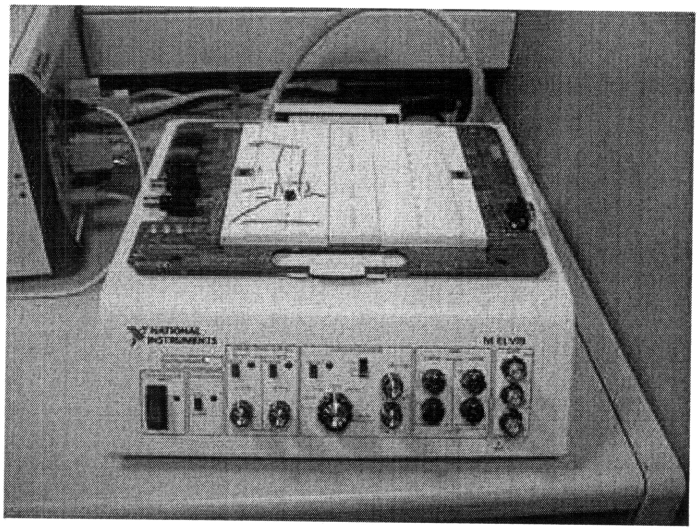


Figure 1-2: Educational Laboratory Virtual Instrumentation Suite (NI ELVIS) workstation.

ELVIS was designed for use in university education. It is an ideal platform because all of the components a student needs for basic electronics experiments are in the device. There is no need to manage and learn to use multiple devices. The ELVIS is portable and more compact than traditional instruments, making it ideal for the limited space available for laboratory equipment.

There are also precedents for ELVIS being used in university level education. The ELVIS is being used in the Bioelectronics Project Laboratory (course number 6.121) at MIT. [7]Other universities such as Georgia Institute of Technology and Vanderbilt University use ELVIS in teaching some of the required electrical engineering courses. [6]

Beyond the advantages the ELVIS has in academia, it is appropriate for use in sub-Saharan Africa given the cheaper price. The ELVIS bundle with a digital acquisition card retails for $1,999 in the United States at the current time. While this is still expensive for some institutions, using the iLab technology, one platform can be shared among many students, making the cost per student significantly lower than traditional

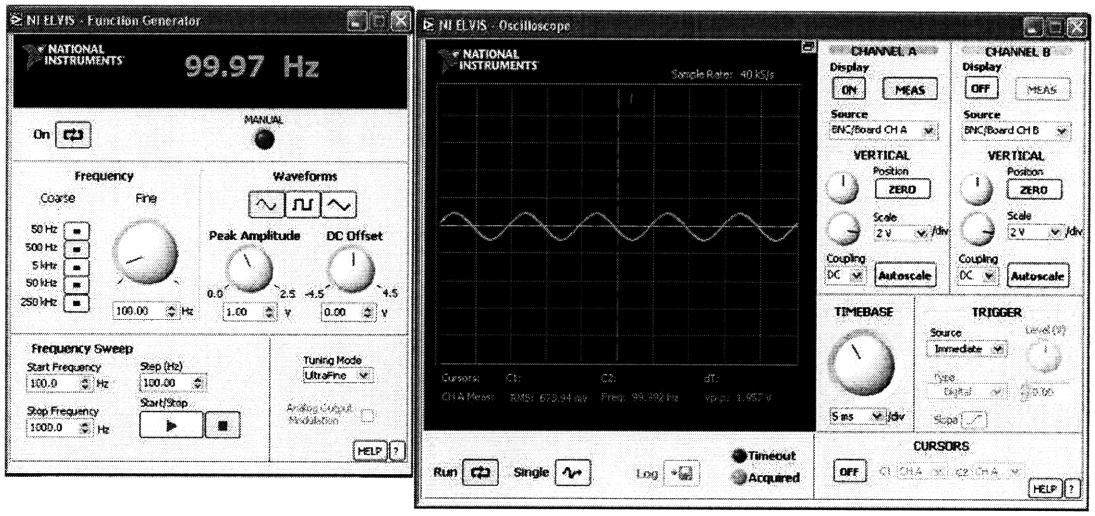


Figure 1-3: Screenshot of the function generator and oscilloscope instruments running simultaneously from the ELVIS software suite.

separate laboratory components.

1.4 Overview of Thesis

This thesis is about the development of a version 2.0 of the ELVIS iLab originally developed and deployed in 2006 by Samuel Gikandi. It includes a description of the additions that were made to the version 1.0 of the ELVIS iLab to include more functionality and flexibility. It also includes work completed while on exchange at UDSM and MUK.

Chapter 2 will focus on the reasons for creating a new version of the ELVIS iLab. It includes a discussion of the general structure of an iLab based on the iLab Shared Architecture (ISA). Chapter 2 also delves into an overview of the original version of the ELVIS iLab and the shortcomings and new ideas that sparked the development of a second version.

Chapter 3 goes into the details of the design for ELVIS iLab version 2.0. It includes information about all aspects of the ELVIS iLab: hardware, server, and the web-based client. The improvements for version 2.0 will be discussed as well as the limitations that still exist.

Chapter 4 will be the conclusions drawn from the research and development of the ELVIS iLab version 2.0. It will also discuss the outlook of the ELVIS iLab and make recommendations for future development and use.

Chapter 2

Inspiration for Developing Version 2.0 of the ELVIS iLab

The first iteration of the ELVIS iLab aimed to incorporate new, low-cost equipment in to the existing iLab architecture. It is important to understand the underlying iLab architecture the ELVIS iLab is a part of. To maintain consistency, version 2.0 of the ELVIS iLab continued using the same architecture as version 1.0 and previous iLabs.

2.1 Backbone of the ELVIS iLab: iLab Shared Architecture

Since the first iLab was conceived in 1998, there have been several iterations of the iLab architecture. The first microelectronics iLab consisted of a Java web client communicating directly with hardware connected to a server. In 1999 as other laboratories were being developed in different engineering disciplines, each laboratory was created with its own unique structure. Seeing the inefficiencies in this process, it was decided in 2001 to create a more standardized architecture that was generic and modular enough to work with many different laboratories and streamline the development process. [4]

The ISA consists of essentially four parts: web client, service broker, lab server, and lab equipment (Figure 2-1).

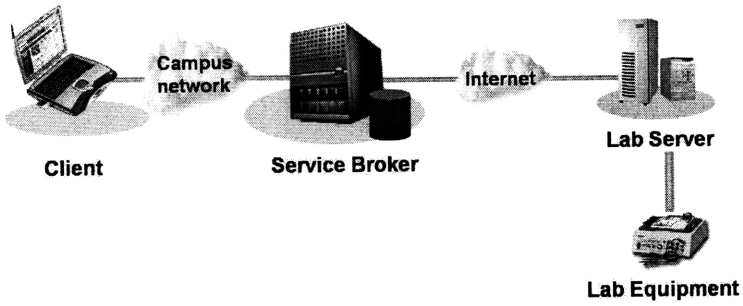


Figure 2-1: iLab Shared Architecture.

2.1.1 Service Broker

It is worth mentioning the service broker first, as this component is the heart of the ISA and is the main differentiator between the new architecture and the previous versions of iLab. The service broker is middleware developed in 2002 to simplify iLab management and provide some modularity for development. The service broker is a web-based portal that mediates communication between a client and an iLab server. Much of the information is passed as XML documents. Because it uses relatively generic function calls, one service broker can handle multiple clients and servers and stores the relevant information for all of them. The service broker is laboratory independent, so new iLabs are created around the service broker and utilize the same function calls provided in the service broker API. The service broker also stores user accounts. With this function users can be organized into groups (perhaps by course) and can have permissions set by an administrator to see different iLab clients (see Figure 2-2). For example, a student in a civil engineering course could have the shake table iLab exposed to them, but not the microelectronics iLab. Experiment data is stored through the service broker as well. Students and administrators can retrieve all of information (configurations, results, date/time information) for experiments that have been submitted. Each institution can have its own service broker for easy management of user accounts and laboratory resources or they can share.

There are two types of service brokers: batched and interactive. The batched service broker allows users to see the configuration of an experiment and specify what parameters they want to input to the equipment without using any of the equipment resources. This allows several users to do these steps simultaneously. Once a user has specified their parameters, they submit them to the lab server. Users can access the equipment one at a time, so there is a queuing system to ensure the person that submits a specification first is served first. After the user's specification is finally passed to the equipment, the execution time is only a few seconds, immediately freeing up the equipment for the next user on the queue. Given each user only needs a few seconds of the equipments time, this is a good model for large scale classes. Students can also access the laboratory whenever they find the time and are not under any time pressure in which to complete an assignment. However, a jam with a long queue can occur if too many students try to access the experiment at the same time (e.g. a large amount of students wait until an hour before a deadline to complete the assignment).

The interactive broker gives users exclusive, uninterrupted, and continuous access to laboratory equipment. Users sign up for time blocks in which they will be the other person allowed to dynamically control the laboratory equipment in real time. While the real-time control is certainly an advantage over the batched architecture, time is a limiting factor and the interactive laboratories may not scale as well in some scenarios (e.g. a time intensive experiment and a large class size). Students may also be pressured to finish their experiments in their given time slots. Since every student schedules their own time slot, there is not the possibility of long queues preventing access to the equipment in a timely manner.

The ELVIS iLab version 2.0 uses the batched architecture because it was modified from version 1.0 and the microelectronics iLab, which predates the interactive architecture. It is also a more appropriate design for use in the developing world. There is a need for the most students to access laboratories. Many students do not have access to the Internet 24 hours a day—the electricity is frequently out, students do not have access to computer laboratories outside of school hours, and many must go to Internet cafes to get access in the evenings. The Internet connection is also slow and unreli-

able, so the bandwidth intensive interactive laboratory may lag and a student may lose their connection part way through the process, causing them to have to schedule another time and restart the experiment. These constraints and unpredictable events would make scheduling student access time for an interactive experiment a logistical nightmare. The batched experiments can allow students to configure their experiments at any time without blocking access to other students. When the student is ready to submit, they will only use a few seconds to access the equipment. If they ever lose access to the Internet during their experiment configuration or submission, the time to resubmit their information is minimal. [4]

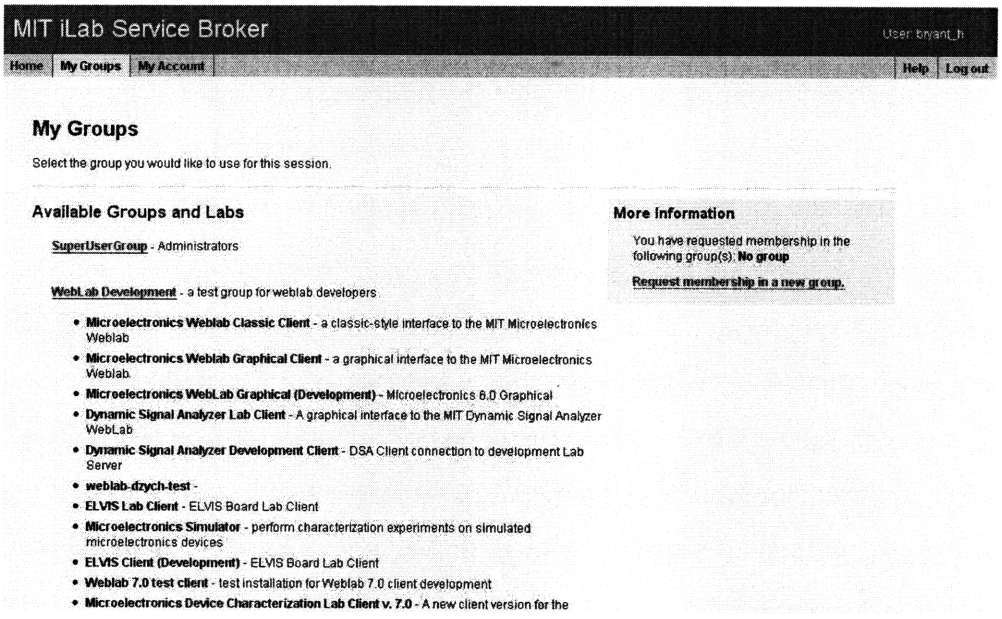


Figure 2-2: Screenshot of MIT iLab Service Broker. This page indicates the groups a user is a member of and the iLab clients the user has access to.

2.1.2 Laboratory Equipment

Theoretically any laboratory equipment that can communicate with a computer can be modified into an iLab. As long as the API is exposed to a programmer, the functionality can be called by the server.

2.1.3 Lab Server

The lab server is a laboratory specific part of the ISA that interacts directly with laboratory equipment. A lab server can interact with multiple service brokers simultaneously. The lab server accepts requests from a user for specification of values, parses the information, and submits them to the laboratory equipment. It then passes the results back through the service broker to the user.

Another function of the lab server is to serve as an administrative interface. The desired configuration of a laboratory is stored on a laboratory administration page and this information is passed to the user when they open the iLabs client. It also stores a log of experiment execution requests and manages administrator accounts.

One challenge that arises in developing a lab server is that it essentially needs to be reprogrammed for each device that is connected to it. For example, each device will have unique procedures to access its functionality; these will need to be included in the execution engine. Parsing and validation of the XML documents that are passed through the service broker will also need to be modified based on the information that will need to be included. Chapter 3 will discuss the individual components that were modified for version 2.0.

2.1.4 Web Client

The final component of the ISA is the web client. The web client is the user interface for interacting with the experiment. The client is a representation of what the laboratory hardware is doing so this will also be laboratory specific. For example, a basic electronics laboratory will have a schematic drawing of the circuit that has been wired on a prototyping board.

2.2 Previous Work on ELVIS: Version 1.0

The first version of the ELVIS iLab was completed in 2006 by Samuel Gikandi as part of his Master of Engineering thesis at MIT. [3] Gikandi integrated the ELVIS into the

iLab Shared Architecture by modifying the previous lab server and client code from the microelectronics iLab. The modification of the existing code is quite apparent as the ELVIS iLab still resembles the microelectronics iLab. A user will be comfortable using both of the web clients.

Gikandi exposed two of the twelve instruments from the ELVIS in version 1.0 of the iLab: the function generator and one channel of the oscilloscope. In the web client (see Figure 2-3) the user is presented with a schematic drawing of the circuit that is wired on the prototyping board. The user then has two instruments that must be configured. The first is the function generator (FGEN) instrument. The user can specify the amplitude, waveform type (sine, square or triangle), frequency, and DC offset. The second configurable instrument is the output channel (SCOPE). For this instrument the user specifies the sampling rate and sampling time. The specifications are then sent through the service broker to the lab server where the ELVIS functions are called. The results are then passed back along the chain and displayed to the user in graphical form on the client. [3]

For this first iteration of the ELVIS iLab, there were limitations on the design. Since only the FGEN and one channel of the SCOPE are exposed, only a small range of circuits can be implemented. There is also a variable (DC) power supply and two arbitrary waveform generators available on the ELVIS. Another limitation is the number of experiment setups that can be available concurrently. Since there is only one input and one output, a laboratory administrator can only wire the input and output channels to one circuit at a time. Every time a new circuit or component is needed, someone needs to remove the existing one and wire a new setup. This maintenance is not ideal for a system that is designed to be shared among different courses and universities that will not necessarily have the same assignments and setups simultaneously. There is also not much freedom for the student and their opportunity to experiment is limited. A student can only specify values for the FGEN and SCOPE. They cannot change the configuration of the circuit (e.g. change the value or position of a resistor) and see how the output changes. Being able to change the components of the circuit in addition to the input and output would greatly add to the educational

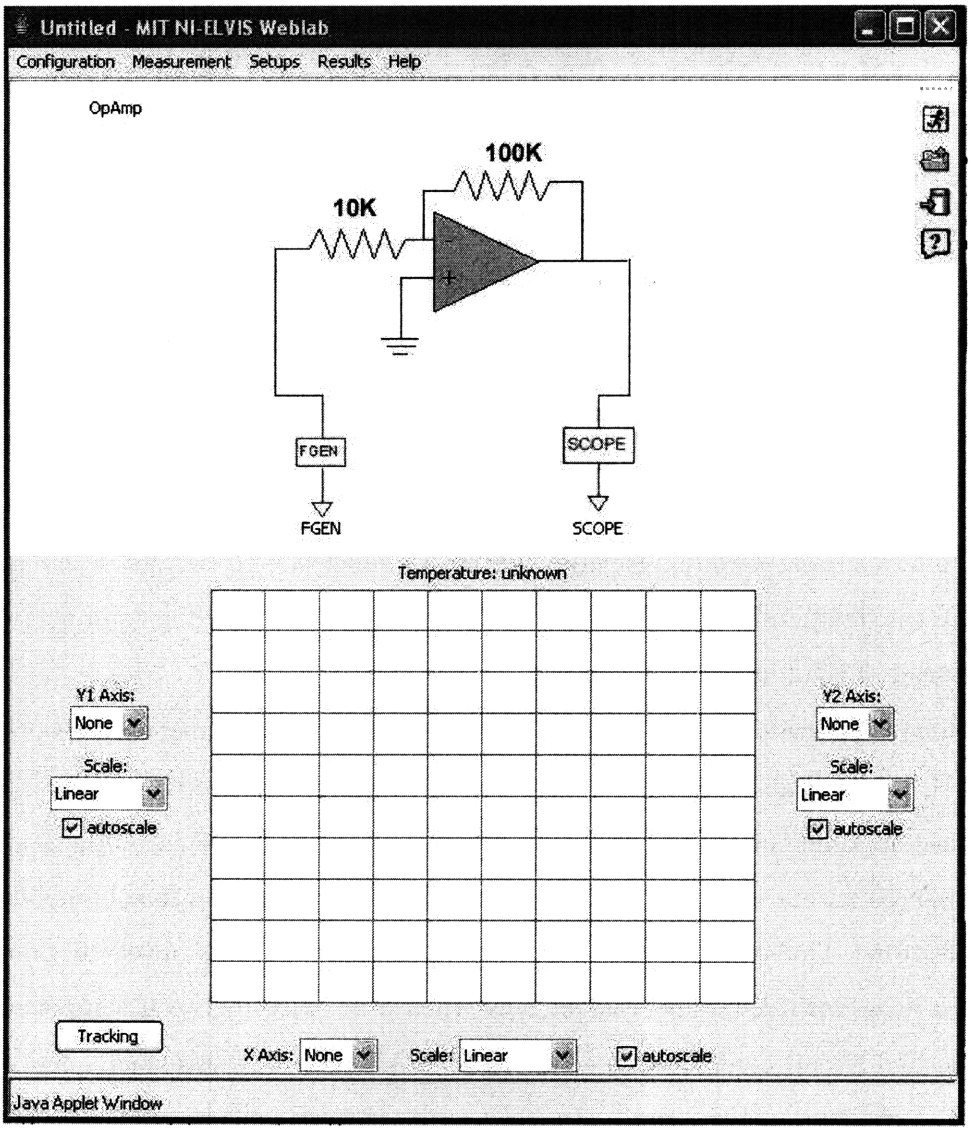


Figure 2-3: Screenshot of web client for the ELVIS iLab version 1.0. The schematic is a representation of an operational amplifier with a user configurable function generator (FGEN) and an oscilloscope channel (SCOPE).

value of the iLab. Version 1.0 did not fully exploit the capabilities of the ELVIS and at MIT and the African universities there has been a desire to expose more functionality and increase the flexibility of the ELVIS iLab.

2.3 ELVIS Version 2.0 Background and Overview

The desire to remove the limitations in ELVIS iLab version 1.0 has been the inspiration for developing a second version of the ELVIS iLab. Version 2.0 picks up from where Gikandi left off and essentially adds to and modifies his code base. There are two main features that have been added to create version 2.0: switching and an additional power supply.

The additional power supply from the ELVIS that has been added is the variable power supply. For version 2.0 a DC power supply that operates between -12 and +12 volts is at a laboratory administrator's disposal for inclusion in circuits. It can provide a constant signal in the -12 and +12 volt range. This can be used in conjunction with or instead of the function generator.

Integrating switches into the ELVIS iLab architecture was inspired by work done at OAU on the iLab project. OAU used a software controlled switch to allow students to select six different configurations of operational amplifier circuits. [3] The student is presented with a schematic of the circuit that displays an operational amplifier and the resistors. The students select the desired configuration (e.g. inverting amplifier) from a drop down list. The student must then draw wires between components for the desired circuit configuration. (see Figure 2-4) Once they have completed wiring, the iLab checks to make sure the student wired it correctly before submitting it to the lab server for execution. This enhances the student's educational experience, as they get to experiment with different configurations and learn by trial-and-error without putting equipment at risk.

Version 2.0 uses the same software controlled 100-channel switch that is used at OAU, but presents the information differently to the user in the client. Instead of showing components and no wires, ELVIS iLab version 2.0 displays wires with

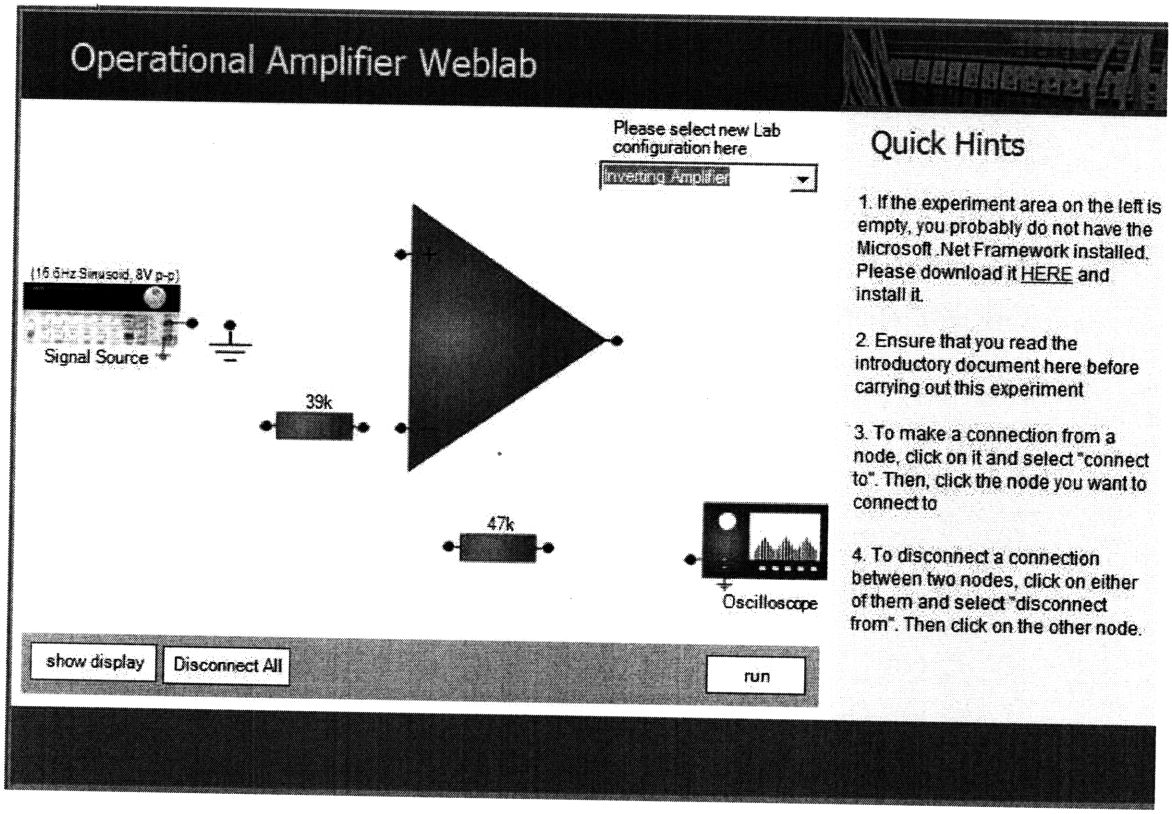


Figure 2-4: iLab from Obafemi Awolowo University that incorporates switching. Students have some flexibility in wiring different configurations together.

generic components that need to be configured (similar to the FGEN and SCOPE instruments from version 1.0 in Figure 2-3). The user must select the values for each component (e.g. a 100 ohm resistor or a 15 nF capacitor). Each of these components are physically wired on the board, but connected to the circuit with different switches. Like the OAU approach, this forces the student to think about how to create circuits instead of just having the circuit already properly wired for them. Switches can also be associated with different circuit setups, so sharing power supplies and output channels is possible. Unlike the OAU approach, my approach does not require error checking to ensure a student has wired something correctly. The wires are already in place, and the student has a limited selection of circuit components that can be chosen. In the OAU design in Figure 2-4, the user is given resistor values can only wire it in an inverting configuration. In my design a user can be given an operational amplifier with multiple resistor values (so they can modify the gains) and would not necessarily be restricted to just an inverting amplifier configuration. This allows for more possible circuits and a more extensive educational experience.

The changes to ELVIS iLab version 1.0 required modification the lab server and client code and the addition of new hardware. The next chapter will go into detail on exactly how this was accomplished.

Chapter 3

ELVIS Version 2.0 Detailed Design

This chapter gives a detailed outline of version 2.0 and the modifications made to create the new version. The changes and additions to version 1.0 were substantial in all aspects of the architecture with the exception of the service broker. There is also a brief discussion of testing and deployment of the system.

3.1 Laboratory Equipment

Version 2.0 consists of two pieces of hardware: the NI ELVIS academic bundle and a switching system. The NI ELVIS consists of twelve instruments the user can use either through the NI ELVIS software suite (see Figure 3-1) or through virtual instruments in LabVIEW. The ELVIS station consists of the actual instrumentation in a box and a removable prototyping board. The ELVIS station connects to a computer through the NI PCI-6251 digital acquisition (DAQ) card. For version 2.0, the function generator, oscilloscope, and variable power supply are utilized.

The specifications for the ELVIS are somewhat of a limiting factor. The function generator can generate signals between +/- 2.5V in three different waveform types (sine, square, triangle wave) in the frequency range between 5 Hz to 250 kHz. Through our experience, the ELVIS is not very accurate at relatively low voltage ranges (30-40 mV). There are inaccuracies in the amplitude and the offset. After unsuccessfully troubleshooting the problem through calibration, noise from within the ELVIS device

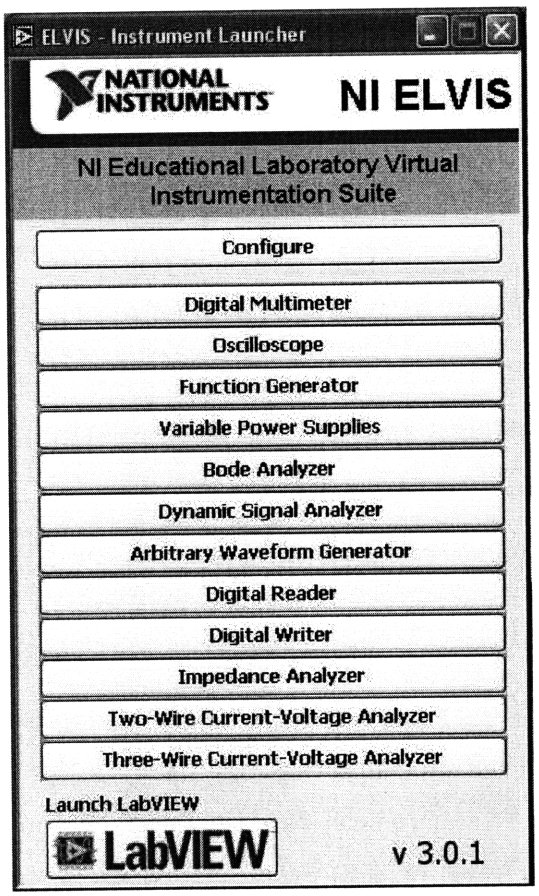


Figure 3-1: NI ELVIS software suite. Users can control the different instruments from this user friendly panel.

must be an issue.

The variable power supply is actually made from two separate channels: one that provides DC voltages between -12V and 0V and another that ranges between 0V and +12V. Finally, the oscilloscope channels can measure voltages in a range of +/- 10V at a maximum rate of 500 kHz. There are eight channels capable of measuring voltages, but only two are used in order to measure the input signal and the output signal. This is a limitation in this design, and not a limitation of the ELVIS.

The switch is the software controlled National Instruments SCXI-1169 switch. The module consists of 100 single pole, single throw (SPST) mechanical switches. This can be used to potentially have 100 different components available in a circuit; however there is limitation in how many can actually be used. The prototyping board is small and each switch requires two nodes, therefore it would be difficult to fit 100 switches on the board at the same time. For my implementation I used a NI LFH200 bare wire cable routed through a NI TBX-50 screw terminal block that can only support 25 switches. Wires are connected from the terminal block to the prototyping board. The wires are very thin, so they tend to come loose. For this reason, the ELVIS and the switch should be placed in a safe location where accidental contact is unlikely.

The switch, much like the ELVIS, can be controlled through a standalone software suite (see Figure 3-2) or through virtual instruments in LabVIEW. Since the switches are mechanical, they have a limited lifespan. Therefore, monitoring the usage of each switch and periodically testing them is necessary. The switches can be monitored through the software suite.

3.2 XML Specification Documents

Communication between the client and server via the service broker is done with three XML documents: LabConfiguration, ExperimentSpecification, and ExperimentResult. These three documents are crucial to start with in the development process because their structures dictate what and how information is represented, accessed

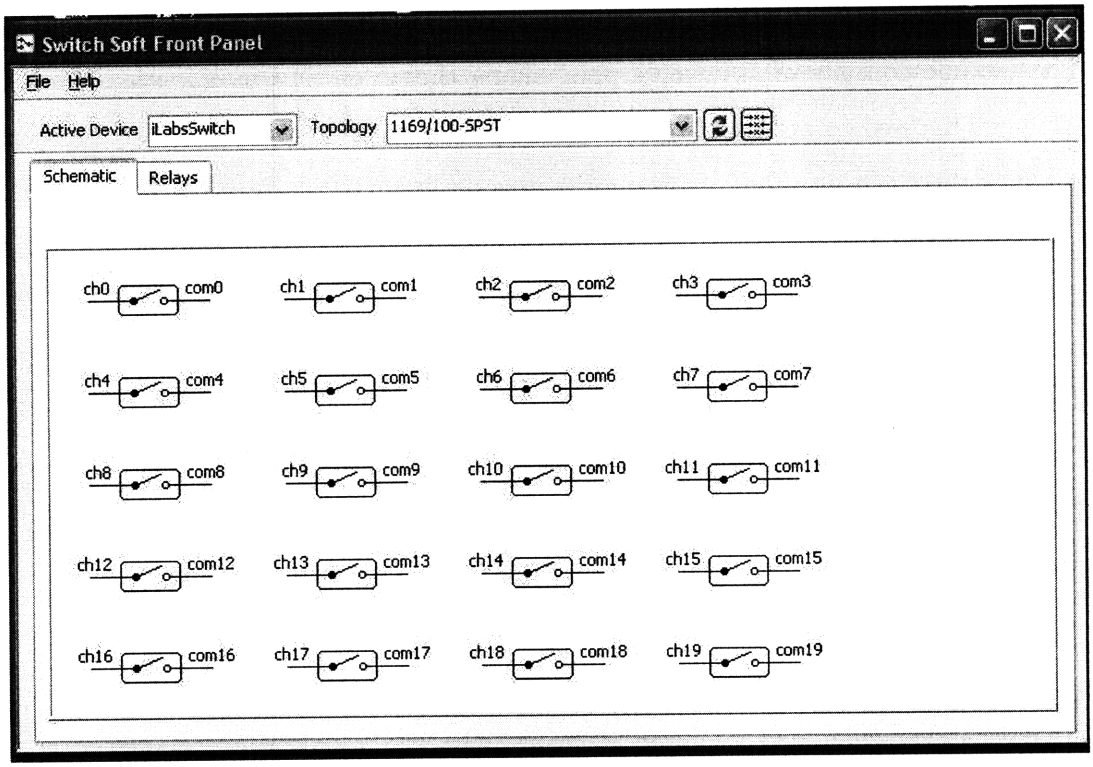


Figure 3-2: NI SWITCH software. Each switch can be opened and closed manually through this software.

and stored in the client and server.

The LabConfiguration XML file (see Appendix A) contains information about the active setups [1] for a laboratory. It contains basic information about each setup such as the name, file path for the image representation, and a brief description. It then goes on to list all of the terminals and their information such as name, pixel location of the icon, and maximum values. When a user launches a laboratory client from the service broker a request is sent to the server to generate this file from the database. From this XML file, the client will display the schematic drawing and place the user configurable components on the client.

The ExperimentSpecification XML file (see Appendix B) is a document that contains the user's desired values for the experiment. After the user launches the client they are given the options for configuring the instruments. The user selects the options for parameters like amplitude, waveform type, and sampling time. After error checking (for values that may be outside of the allowed range or data type), these values are put into an XML file and sent to the lab server where they are parsed and executed on the laboratory equipment.

Finally, the ExperimentResult XML file (see Appendix C) contains the data from the experiment. This file is generic and does not need to be changed to reflect the switching capability or the variable power supply since it only cares about the measured input and output values.

LabConfiguration and ExperimentSpecification needed to be changed to account for two new instruments: the variable power supply (DC) and switchable components (COM). Switchable components can theoretically be any instrument on the ELVIS such as the function generator, variable power supply, or oscilloscope. They can also not be instruments at all-resistors, capacitors, or any other components found in a circuit. To maintain consistency with version 1.0, the FGEN and SCOPE instrument types were kept in the XML structure. For version 2.0, DC and COM were added. DC is actually very similar to FGEN, except there are fewer user configurable parameters

[1] Having a setup active means the laboratory administrator has wired a circuit setup and has made it accessible to users. An inactive setup is not presented to the user as a circuit to configure.

like frequency and waveform type. The only information needed is the DC voltage value. The COM instrument is a little more complicated since it can theoretically be any arbitrary component. A COM instrument does not have any functionality itself, but rather just a terminal number to keep track of it and a pixel location. Its children in the XML format are called subCOMs. These subCOM children share a common pixel location, but may different component types. For example, a COM may be made up of two subCOMs with a resistor type and one subCOM with a capacitor type. This structure allows for a terminal to have multiple components associated with it and can expand to different components and instruments in the future. The implementation in version 2.0 only allows for circuit components and not ELVIS instruments.

3.3 Lab Server

The lab server consists of three main components: the back-end code, database, and web-based administrative interface. Figure 3-3 outlines in detail how the three components interact with one another, the laboratory equipment, and the service broker. The back-end code is broken down further into individual modules that will be explained in further detail in this chapter.

3.3.1 LabVIEW

LabVIEW is a graphical programming language that assists in performing measurement and automation. LabVIEW contains virtual instruments (VIs), which are built-in functions that represent instruments that can either be simulations or actual pieces of hardware, such as oscilloscopes or function generators. [5] I developed in LabVIEW 8.2 Professional Development Edition because it includes the ability to compile dynamic-link libraries (DLLs) which are necessary to call VIs from outside of LabVIEW.

The code in LabVIEW is a VI that is made up of several subVIs, which are modular pieces that perform more specific functions. Figure 3-4 is a detailed breakdown of

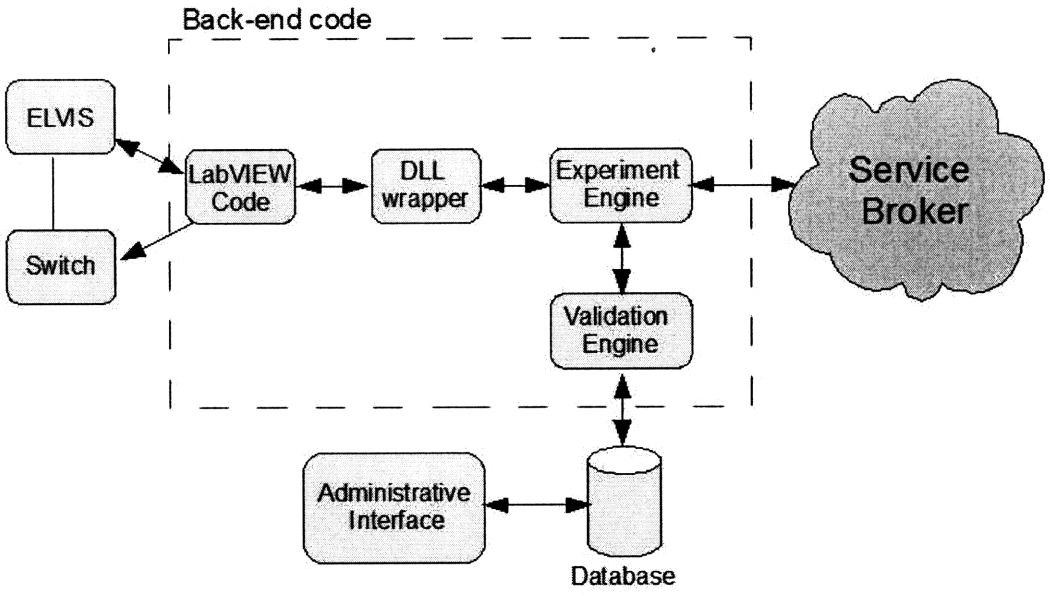


Figure 3-3: Diagram of the lab server. The lab server has back-end code, an administrative interface, and database. It communicates with the laboratory equipment and the service broker.

how the subVIs interact with one another in the ELVIS iLab code.

Figure 3-4 breaks the LabVIEW VI into three layers. Layer 1 is the highest layer and is the VI that the DLL is created from. When the DLL is called, all of the values that are needed to run the ELVIS (such as amplitude, waveform type, switch information, etc.) are passed through a wrapper function. The first step in *Switch.vi* is it takes a string that consists of which switches should be opened and closed. This string is then parsed and the information is passed on to the first of its four subVIs (*MainSwitch.vi*) located in layer 2. In *MainSwitch.vi* the switch is initialized and the switches that have been designated closed are closed. After waiting for the switches to debounce, communication with the switch is closed. *InitializeELVISupdate.vi* and *CloseELVIS.vi* are built-in VIs that initialize and close communication with the ELVIS station. *FGEN.vi* is a VI created for version 1.0. It houses all of the main ELVIS functions and is made up of three subVIs. *InitializeFGEN.vi* and *CloseFGEN.vi* open and close communication with the ELVIS function generator, respectively. *RunFGEN.vi* is the heart of the LabVIEW code. The values that the

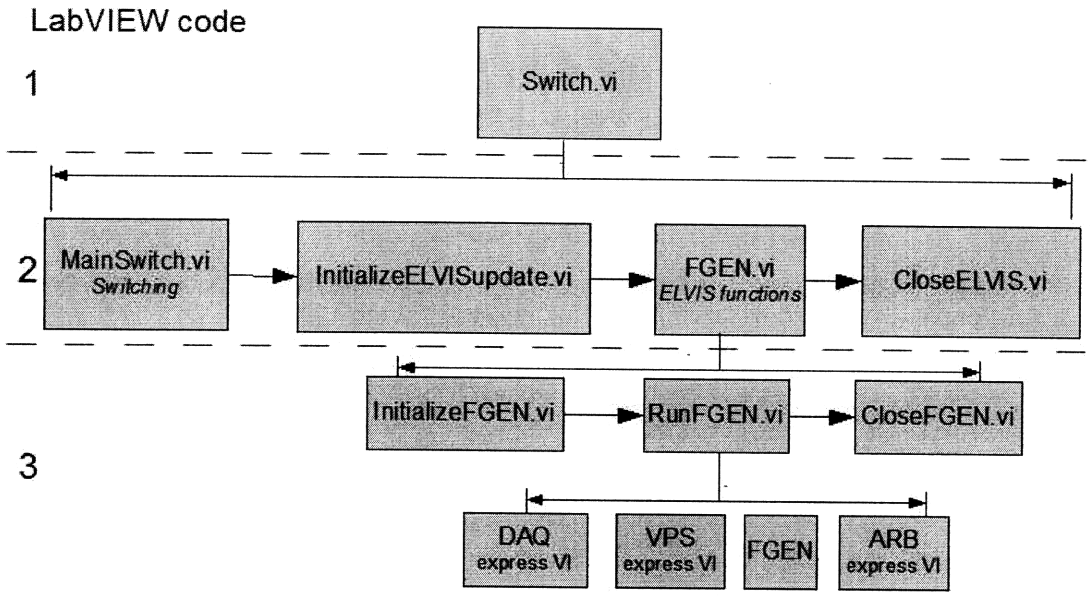


Figure 3-4: Flowchart of LabVIEW code.

user specifies through the DLL function call are passed from layer 1 down to the RunFGEN VI. Here they are passed on to the built-in LabVIEW VIs for the DAQ and FGEN. The DAQ always samples at 100 kHz, but there is a decimator function that resamples at the rate the user specified in the client, and returns the desired number of samples. The FGEN VI is called with the user specified amplitude, frequency, waveform type, and DC offset. Version 2.0 now includes the DC variable power supply (VPS). The VPS has two different channels: one for voltages in the 0 to +12 V range and one for the 0 to -12 V range. To simplify things for the user, a case structure is surrounding the VI (Figure 3-5) to determine whether the user specified value is positive and negative, and then routing this value to the correct channel.

The DLL is compiled from Switch.vi, where it allows you to select which of the parameters should be exposed in a generic function. These parameters are exposed by selecting "connectors" (anything that can be configured by a programmer, such as Supply+, Supply-, Device Name, etc. in Figure 3-5). There is a major limitation in the number of connectors that can be chosen. Each VI can only support 30 connectors.

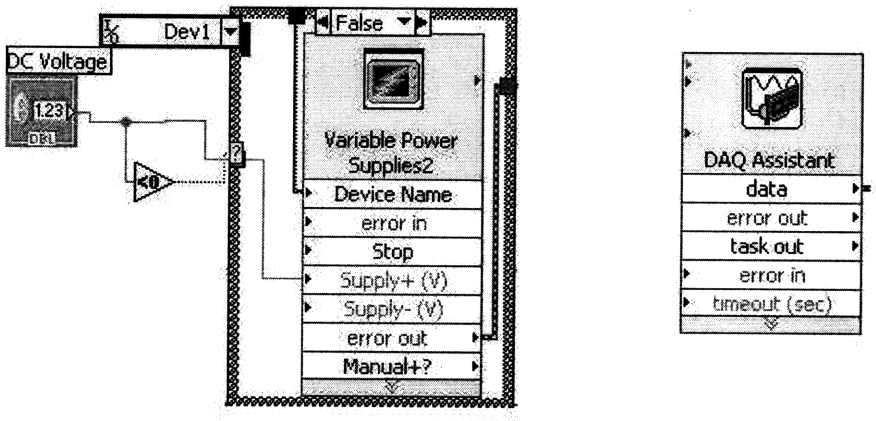


Figure 3-5: The built-in variable power supply and DAQ VIs in RunFGEN.vi.

This is an issue with the addition of the switches, because one implementation choice could be to have a separate connector for each switch number. This would limit the number of switches that can be exposed. The workaround for this is to use only one connector by using a string that has each element of the string corresponding to a switch (element 0 corresponds to switch 0). The elements in the string would be "1" if the switch is closed, "0" if it is opened. This implementation conserves connectors, and in the future all parameters could be passed in a single string instead of as separate parameters.

3.3.2 OpAmpInverter.vb

OpAmpInverter is a misnomer—this is actually the code that wraps around the LabVIEW DLL and calls the function. This class imports the DLL and creates a RunExperiment function which executes the ELVIS VIs with the user specified values. This value also receives back from the ELVIS the data from the DAQ device as a single array. This array needs to be broken down into its three components (input function generator voltage, input DC variable power supply voltage, and output voltage). These are passed back to the client and displayed in a graph for the user.

3.3.3 Validation Engine

The validation engine serves three main functions. It parses the experiment specification (which changed), ensures there are proper permissions, and validates it for correctness before allowing the experiment engine to call its `runExperiment` function. The parsing and validation steps had to be modified for version 2.0 to account for the new XML structure. The validation engine is compiled into a DLL and run when the user submits an experiment to the service broker.

3.3.4 Experiment Engine

The experiment engine is an executable that runs in the background of the lab server waiting for activity from the service broker. When the service broker passes a request from user to see a certain laboratory, a `LoadJob` function is called and creates the XML lab configuration file from information in the lab server database (see Appendix A). This function needed to be modified for version 2.0 to account for the DC and COM instrument types that were included in the new LabConfiguration XML file. It passes this file back through the service broker to the client where it is parsed and further steps are taken to display the information to the user.

The experiment engine also waits for experiments to be submitted from users via the service broker. The experiment specification created by the client when the user submits a valid experiment is first parsed by the `ParseExperimentSpec`. This function removes the relevant information from the XML document and loads the values into local variables. These are passed to a `runExperiment` function which creates an instance of the `OpAmpInverter` class and calls its `RunExperiment` function with these variables.

I added a `getSwitchNums` function to handle the switching. As mentioned before, only 30 arguments can be supplied to a LabVIEW function so I had to use one string argument which contains all of the information for the 25 switches. This function gets the switch numbers from the database of all of the components in the user's specification. It forms a comma separate values of switches that should be closed. It

then translates this to a string of 25 "0" characters and replaces an element with a "1" if the corresponding switch is to be closed. This 25 character string is passed as an argument to the `runExperiment` function.

3.3.5 Lab Server Management

The lab server is managed by an administrator through a web interface that interacts with the lab server database. Overall the database has been extended and the web interface reorganized to account for the switching and variable power supply functionalities.

The database is a modified version from the Microelectronics iLab. It was created using Microsoft SQL Server. The database is used for tables as well as stored procedures that are used in conjunction with the administrative web interface.

The relevant tables for the ELVIS iLab are below with brief descriptions of their purpose.

Table 3.1: Database tables of ELVIS iLab and brief descriptions of their purpose.

Table Name	Modified in V2.0?	Brief Description
ActiveSetups	No	Keeps track of which setups are active/accessible
Brokers	No	Keeps track of service brokers and passkeys for authentication
ClassToResourceMapping	No	Maps usage classes to resources and keeps track of permissions
JobRecord	No	Keeps tracks of jobs submitted, current status, execution time and results
LoginRecord	No	Keeps record of logins for admin interface
Resources	No	What functions (e.g. ability to manage a setup) & experiment setups can be accessed
Setups	Yes	Stores image location, # terminals used, switches needed for setup
SetupTerminalConfig	Yes	Stores information about terminal (see Table 2)
SiteUsers	No	Stores info on user accounts for the admin interface
UsageClasses	No	Stores guests and administrators; serves to group together for control of resources
WebMethodRequestLog	No	Logs methods that have been called like GetExperimentStatus, GetLabConfiguration, etc.

Since most of the tables were generic enough that they did not need to be changed for version 2.0, I will only go into detail on the two tables that needed modification for switching: Setups and SetupTerminalConfig.

Setups needed a new column (*setup_switches*) to handle a multiple setups scenario with switches. If there is a case where multiple experiment setups are desired, and certain switches must be thrown (i.e. if there is sharing of the function general in the setups) and the user will not have the ability to change the switches, then the *setup_switches* column will be used. Any switches that must be thrown are input as a

comma-separated value list of the switch numbers. These will be added to the switch string that is created in the experiment engine.

SetupTerminalConfig stores the information for each terminal and component that is needed for the lab configuration. Crucial information like the location of the image and maximum values allowed for instruments are kept here. This table needed four columns added to handle switching. The column *setupterm_id* was added to create a unique identification number for a component. In version 1.0, *number* served this purpose, as there was only one component (FGEN or SCOPE) per terminal and the terminal number sufficed. Now in version 2.0, the structure allows for multiple components to exist at a single terminal. So there needs to be a terminal number so components can be identified as being at the same location, but they need to be distinguished from one another as well. All terminals and components have a *setupterm_id.* The *switch_id* column keeps track of the order of components for a terminal that has multiple components. This is mainly for listing them on the web interface page in some sort of order. The *switch_num* column contains the comma-separated values of the switch numbers that need to be thrown to get the component. The *subCOMType* is the type of component that is connected. There are predefined choices to the administrator (resistors, capacitor, inductor, open circuit, short circuit, other—all in a horizontal or vertical orientation for the image) but it can be expanded in the future if necessary. Finally, *is_switch* is just a Boolean value that the administrator selects to indicate whether the terminal uses switches. This is for error checking and robustness.

The web interface was changed for aesthetic and functional reasons, but any user of the version 1.0 interface will find it familiar. In version 1.0 there were only two terminals (FGEN and SCOPE) so all of the information for those terminals such as pixel location, maximum values, etc. could be displayed on the same page. But in version 2.0 there is the possibility of more than two terminals, and multiple components for each terminal. This information could be very difficult and confusing to read all on one page, especially if components are not added to existing terminals in order. For version 2.0 there is a table that displays all of the terminals and components in order

Table 3.2: Columns and data types of the SetupTerminalConfig table. Columns marked with * were created for version 2.0.

Column	Data Type	Column Name	Data Type
setup_id	int	max_sampling_time	float
number	int	max_sampling_rate	float
name	varchar	max_offset	float
x_pixel_loc	int	max_points	float
y_pixel_loc	int	setupterm_id*	int
max_amplitude	float	switch_id*	numeric
max_current	float	switch_num*	varchar
instrument	float	subCOMType*	varchar
date_created	datetime	is_switch*	bit
date_modified	datetime		

and allows the user delete or view more information about the terminal/component individually (see Figure 3-6).

When a user chooses to create or view a terminal or component, the setup screen is basically the same screen as in version 1.0 with added options for switches (see Figure 3-7). Now the user can indicate whether a terminal uses switches, what the switch numbers are. If the terminal is a component (such as a resistor), the user can choose the component the orientation the image should be on the client.

Another aspect related to the tables and the web interface is the stored procedures that alter the database after a user makes a command. There are several procedures that perform functions like add brokers, copy setups, and delete terminals, but these have been largely unchanged. The only procedure that has been changed for version 2.0 is `AddSetupTerminal`. This procedure creates a new instrument terminal (similar to a node on a circuit) with the values entered by the user (in Figure 3-7) and adds it to a setup. It also updates information in the Setup table such as number of terminals used, date modified, etc. `AddSetupTerminal` needed to be modified to account for the five new columns that were added in the SetupTermConfig table. A new procedure that is modeled after `AddSetupTerminal` was created to handle the situation where a component is created and added to a terminal. `AddTerminalComponent` needs the same information as `AddSetupTerminal`, but needs to check if a terminal already

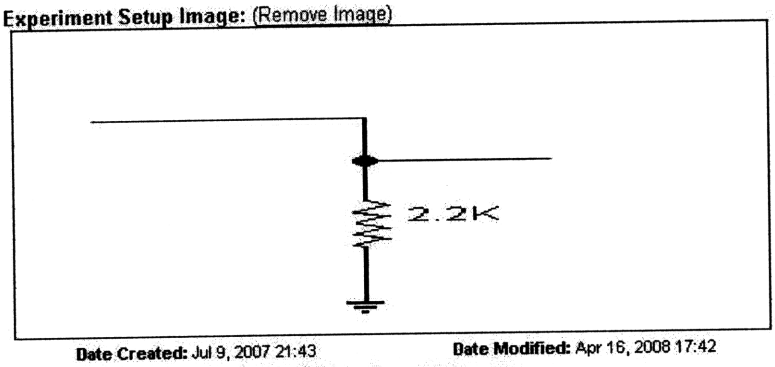
General Information

Name: ELVIS Voltage Divider **Terminals Used:** 3

Description: Voltage divider (1/2) R1=R2

Switches for Inputs/Outputs (CSV): 7

Experiment Setup Image: (Remove Image)



Date Created: Jul 9, 2007 21:43 **Date Modified:** Apr 16, 2008 17:42

Update | Copy Setup | Delete Setup

Terminal Definitions

Terminal #	Component #	Name	Instrument Type	
1	1	Input Voltage	FGEN	View | Delete
2	1	Output Voltage	SCOPE	View | Delete
3	1	3.3K	COM	View | Delete
3	2	2.2K	COM	View | Delete
3	3	1.32K	COM	View | Delete

Figure 3-6: General Information page for a setup. It contains a list of the terminals and components for easier viewing.

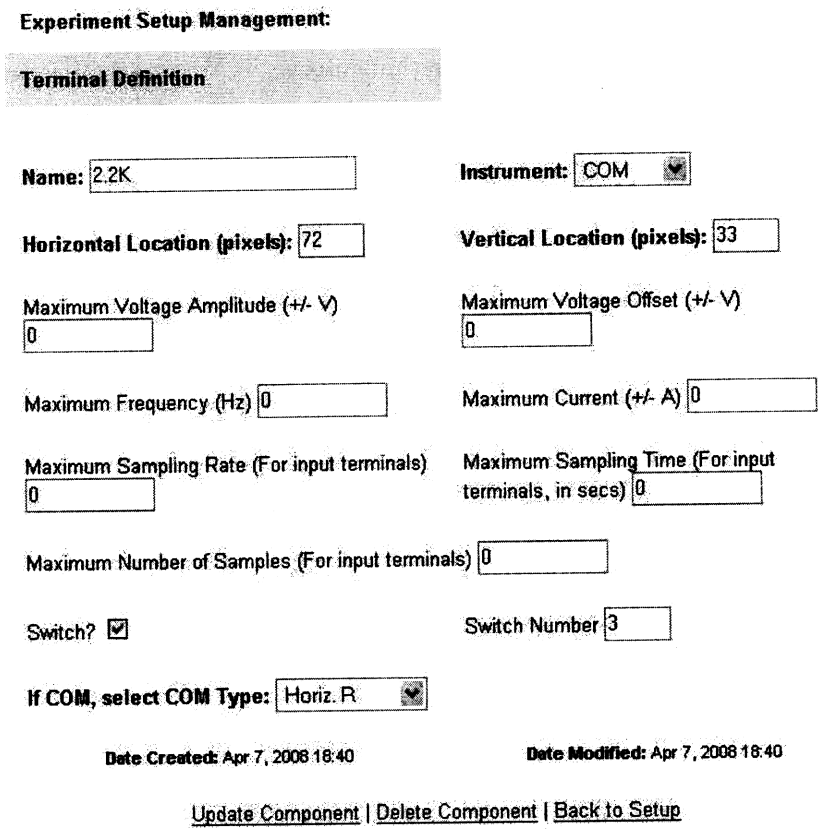


Figure 3-7: Terminal Definition page on administrative interface. Here a terminal or component is defined with user specified values.

exists, and if it does adds the component using the same terminal number. If this is the first component of a terminal it needs to create a new terminal.

3.3.6 Resource Permission Manager

The final portion of the lab server worth mentioning is the resource permission manager (RPM). This works with service brokers and manages, creates, and removes their associations with the lab server. It also handles site users, groups, and deals with permissions. These are generic functions that interact mostly with the service broker, so they did not need to be changed.

The main components of the RPM that needed to be changed are the functions that work with the administrative interface. The RPM takes the values that the user inputs in the administrative interface and passes them on to the database's stored procedures. It opens a connection with the lab server database and executes the stored procedures on the values the user specified. Since `AddSetupTerminal` and `AddTerminalComponent` procedures were modified and created, respectively, for version 2.0, the RPM needed to be changed. The `AddSetupTerminal` function in the RPM needed to be changed to include the new switch parameters and pass the values to the stored procedure. `AddComponent` is a new function and needed to be created in order to pass the values to the `AddTerminalComponent` stored procedure.

There is also a `GetLabConfig` function that needs modification. The function opens a connection with the database and pulls the configuration information from the Setup and SetupTerminalConfig tables. It then puts this into a string in XML format. Because of the change in the XML LabConfiguration document, this function needed to be modified to include the DC and COM instrument types.

3.4 Web Client

A user specifies experiment values, submits those values, and receives the results through the web client. The version 2.0 client is based on the ELVIS version 1.0 client, which was created from the microelectronics client. The client is written using

Java and has a modular structure that could be reused. I wanted to keep the same structure and reuse previous code to maintain consistency and it would be backwards compatible with version 1.0. The generic graphing, xml, and server interface packages that were derived from the Microelectronics client could be used without modification. Much of the back end functionality of the client was kept. The client has options to do things like save and retrieve experiments and view data as a comma-separated table.

The user experience in the client is virtually the same as it was in version 1.0. When the user launches the client, the LabConfiguration XML document is sent from the server. The client has classes that parse the information from the LabConfiguration and create instrument objects that store the information. The schematic image is displayed to the user along with configurable instruments that are represented by red icons (see Figure 3-8). The user then clicks on each of the instruments and specifies their desired values in the popup dialog boxes (see Figure 3-9). Once all of the instruments have been configured the user submits the experiment. It is checked for correctness first to ensure all instruments have been configured with valid values (data type and within the allowed ranges). Once the check has been passed, the ExperimentSpecification XML is created and sent to the server to be parsed and executed.

After the server runs the experiment it receives the input and output data, puts the results in the ExperimentResult XML document, and sends them back to the client. The client then parses the XML and graphs the data.

As mentioned in this chapter, two of the XML documents were modified and this had an effect in the client. The *ExperimentSpecification.java* class, which creates the XML document and the *LabConfiguration.java* class, which parses the LabConfiguration XML sent from the server, needed to be changed to reflect the new structure of the XML documents.

The client package, which contains the majority of the user interface code, needed to be modified substantially. Classes like *Instruments* and *Terminals* needed to be modified in order to handle the new DC and COM types. These existing instrument

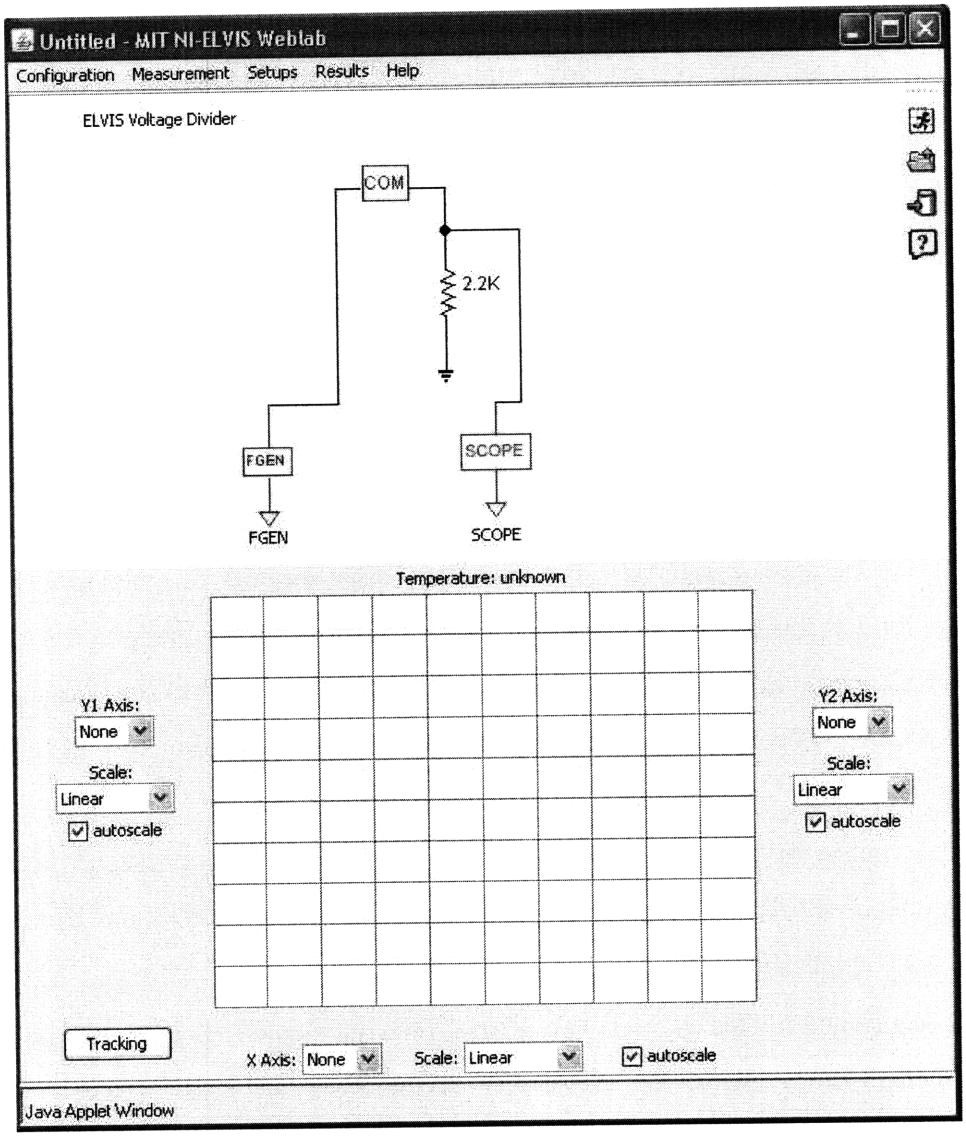


Figure 3-8: Client with unconfigured FGEN, COM, and SCOPE instruments.

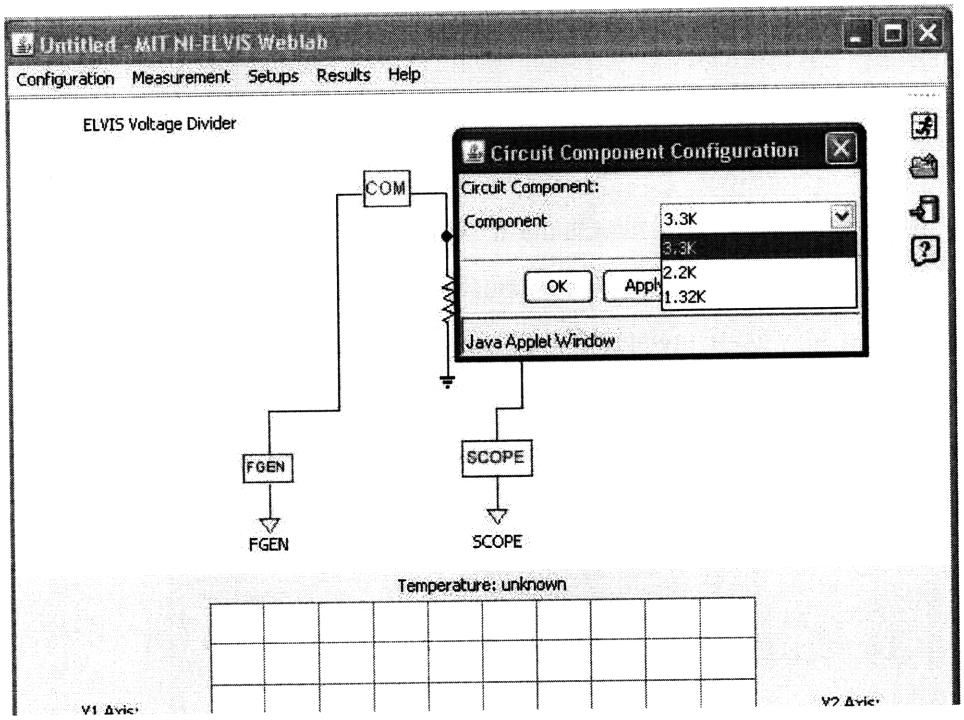


Figure 3-9: After clicking on a configurable instrument the user is given options for the parameters of the instruments.

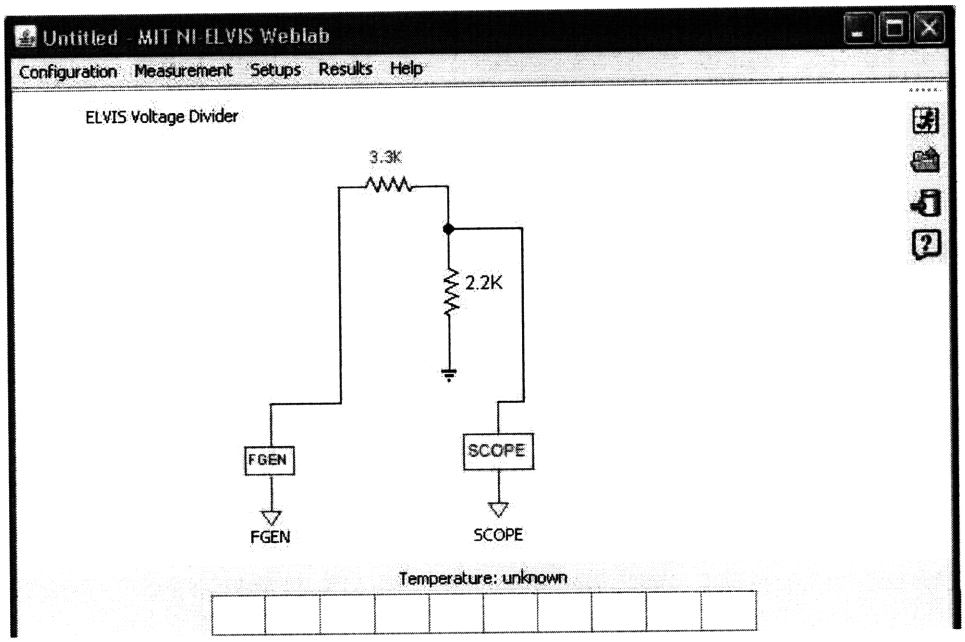


Figure 3-10: After selecting values for an instrument, the box icon changes into a schematic representation of the instrument.

classes and their associated methods (FGEN, SCOPE) needed to be modified because they now possess more characters such the unique setupterm_id. I created new classes for DC and COM that contain most of the same methods as the FGEN and SCOPE types.

In *SchematicPanel.java* the schematic drawing is displayed to the user along with the icons for the user configurable instruments. This needed to be extended to include the DC and COM instrument images. The process of displaying these images is a source of hard-coding that limits the possible COM types without changing and compiling the client code. The COM types are currently restricted to a resistor, capacitor, inductor, open circuit, short circuit, and an arbitrary "other" type.

3.5 Testing and Deployment

The entire system has been tested for three main scenarios. The first scenario is the traditional version 1.0 setup, where there are no switches used. Version 2.0 still works in this case and therefore is backwards compatible. A university that does not have the switch hardware can still use this version to run their old experiments and can use it to build new experiments that utilize the variable power supply. Another scenario is with one setup with multiple components. Switches are connecting components to the rest of the circuit. This performed well, although in one instance it appeared that the switches were not closing or resetting in time before the power supply was run. For this reason I added a delay between the switch initialization and the then the initialization of the ELVIS. This appears to have solved the problem. The third scenario is having multiple setups active and available to the users. In this case two setups share the function generator and an output channel by using switches to connect them to the different circuits. This is accomplished by using the setup switches option in the lab server management page. I was able to switch between two setups and perform experiments successfully. The system has not been load tested with multiple users yet, so this must be performed before it is deployed for use in a class.

When the system is ready for deployment the client is packaged as a JAR file and can be stored on the service broker computer. The lab server code needs to be overwritten on existing servers since most of it has been changed for version 2.0. The database will also need to be rebuilt using a new script. The lab server and client can be registered on the service broker using the existing process.

Chapter 4

Conclusions and Recommendations for Future Work

ELVIS iLab version 2.0 is a drastic improvement over version 1.0, but there are still opportunities for the iLab to be expanded and improved in order to enrich the educational experiences of students.

4.1 Conclusions

ELVIS iLab version 2.0 has expanded the capabilities of the existing ELVIS iLab that is being used worldwide in courses. It adds the variable power supply instrument from the ELVIS workbench and, with additional hardware, the capabilities of component switching. Adapted from version 1.0, ELVIS iLab version 2.0 keeps in the iLab tradition of modularity and code reuse. This will be useful for future development and as more features are added.

There are several benefits to the new functionality. ELVIS iLab now can have multiple setups available to users simultaneously, allowing more options for the students, less maintenance for administrators, and easier sharing within and between universities. The ELVIS iLab can be used in the same courses it is currently used in at MIT, but because there can be multiple setups on each prototyping board now, other universities can have their own custom experiments on the same ELVIS station.

Switching can make experiments more interesting and educational because students have more options and flexibility in configuring instruments. Having to choose the values of more components such as resistors and capacitors can make the assignments more challenging for advanced courses.

4.2 Recommendations for Future Work and Development

While the variable power supply has been added, there are still a total of twelve instruments on the ELVIS station. This leaves nine instruments that have not been exposed in the ELVIS iLab architecture. Currently the Bode analyzer is being developed by students at MIT and UDSM. The arbitrary waveform generator, which has two channels that are capable of generating virtually any sort of waveform, is in the final stages of development as well. While these will significantly expand the types of experiments and analyses that can be performed within the ELVIS iLab, there is still the digital domain that has not been explored in detail by any iLab team. This would be a logical next in development of a new version. This would probably deviate from version 2.0, at least in terms of the client as representation of digital circuits is much different from the analog circuits used in versions 1.0 and 2.0.

There is another recent development that should be incorporated into this new version of the ELVIS iLab. Since I began development, the ELVIS iLab version 1.0 has been modified significantly at the LabVIEW level. Overall functionality has remained largely the same, but the code has become easier to use and more parameters for the oscilloscope are exposed by using LabVIEW's built-in Express VIs instead of Low Level VIs. Instead of using three or four low level VIs at different levels to control the ELVIS instruments, they can be controlled using one Express VI. These should be combined in order to maintain consistency and to simplify the LabVIEW code.

Another potential development that may help as more functionality is added from the ELVIS is to use fewer arguments in the LabVIEW DLLs. As mentioned in Chapter

3, LabVIEW can only support 30 connectors which can be exposed through a DLL. It may be beneficial to use the approach used for setting the switch positions in version 2.0 by providing a string argument that contains all of the parameters to control the ELVIS. Instead of having separate connectors and arguments for amplitude, frequency, DC offset, etc., they can all be in a delimited list and passed as one string argument and parsed in LabVIEW. Only one connector would need to be used instead of many. This may be necessary as the number of instruments used on the ELVIS increases and the potential for incorporating other National Instruments (such as the switch hardware) devices is explored further.

Finally, in spirit with the intellectual sharing nature of the iLab project, a more collaborative effort with the other development teams across the world, particularly in Africa would be beneficial. Team development is needed to get everyone on the same level and would greatly speed development through exchange of best practices and division of labor. It can also create more specific labs for needs and expand sharing of the iLabs once they are complete.

Appendix A

LabConfiguration.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE labConfiguration SYSTEM "http://ilab-labview.mit.edu/LabServer/xml/labConfi
<labConfiguration lab="MIT ELVIS Weblab" specversion="0.1">
<setup id="5">
<name>Sample Circuit</name>
<description>A Sample Circuit for V2</description>
<imageURL>http://localhost/labServer/setupImages/genericimage.gif</imageURL>
<terminal instrumentType="FGEN" instrumentNumber="1" setupTermID="1">
<label>Input Waveform</label>
<pixelLocation>
<x>121</x>
<y>94</y>
</pixelLocation>
</terminal>
<terminal instrumentType="SCOPE" instrumentNumber="2" setupTermID="2">
<label>Oscilloscope</label>
<pixelLocation>
<x>195</x>
<y>156</y>
```

```
</pixelLocation>
</terminal>
<terminal instrumentType="DC" instrumentNumber="3" setupTermID="3">
<label>DC Waveform</label>
<pixelLocation>
<x>180</x>
<y>120</y>
</pixelLocation>
</terminal>
<terminal instrumentType="COM" instrumentNumber="4" setupTermID="0">
<label></label>
<pixelLocation>
<x>100</x>
<y>130</y>
</pixelLocation>
<subCOM subCOMType="horizR" instrumentNumber="4" setupTermID="4">
<label>100 K</label>
</subCOM>
<subCOM subCOMType="horizR" instrumentNumber="4" setupTermID="5">
<label>200 K</label>
</subCOM>
</terminal>
</setup>
</labConfiguration>
```

Appendix B

ExperimentSpecification.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE experimentSpecification SYSTEM "http://ilab-labview.mit.edu/labServer/xml/e
<experimentSpecification lab="MIT NI-ELVIS Weblab" specversion="0.1">
<setupID>1</setupID>
<terminal instrumentType="FGEN" instrumentNumber="1" setupTermID="1">
<vname download="true">VIN</vname>
<iname download="true">IIN</iname>
<mode>V</mode>
<function type="WAVEFORM">
<waveformType>SINE</waveformType>
<frequency>100</frequency>
<amplitude>0.5</amplitude>
<offset>0.1</offset>
</function>
</terminal>
<terminal instrumentType="SCOPE" instrumentNumber="2" setupTermID="2">
<vname download="true">VOUT</vname>
<iname download="true">IOUT</iname>
<mode>V</mode>
<function type="SAMPLING">
```

```
<samplingRate>100</samplingRate>
<samplingTime>0.01</samplingTime>
</function>
</terminal>
<terminal instrumentType="DC" instrumentNumber="3" setupTermID="3">
<vname download="true">DCIN</vname>
<iname download="true"></iname>
<mode>V</mode>
<function type="DC">
<dcAmp>1</dcAmp>
</function>
</terminal>

<terminal instrumentType="COM" instrumentNumber="4" setupTermID="0">
<vname download="true">100 K</vname>
<iname download="true"></iname>
<mode></mode>
<function type="subCOM" setupTermID="4">
<subCOM>horizR</subCOM>
</function>
</terminal>
</experimentSpecification>
```

Appendix C

ExperimentResult.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE experimentResult SYSTEM "http://ilab-labview.mit.edu/labServer/xml/experime
<experimentResult lab="MIT NI-ELVIS Weblab" specversion="0.1">
<datavector name="VIN" units="V">0.1,0.2,0.3,0.4,0.5</datavector>
<datavector name="VOUT" units="I">1,2,3,4,5</datavector>
</experimentResult>
```

Bibliography

[1] 2008. http://www-mtl.mit.edu/ alamo/iLabKick-offMeeting.htm.

[2] Jesus del Alamo. Realizing the Potential of iLabs in sub-Sahara Africa, 2005. http://www-mtl.mit.edu/ alamo/del%20Alamo.pdf.

[3] Samuel Gikandi. ELVIS iLab: A Flexible Platform for Online Laboratory Experiments in Electrical Engineering. Master's thesis, Massachusetts Institute of Technology, 2006.

[4] V. Harward, Jesus del Alamo, and Steven Lerman. The iLab Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories. *Proceedings of the IEEE*, 2007.

[5] National Instruments. *Getting Started With LabVIEW*, 2007. http://www.ni.com/pdf/manuals/373427c.pdf.

[6] National Instruments. Universities Using NI ELVIS, 2008. http://www.ni.com/academic/ni_elvis/universities_using_nielvis.htm.

[7] Soumyajit Mandal. Course Info, 2008. http://stellar.mit.edu/S/course/6/sp06/6.121/courseMateria

[8] National Instruments. *NI ELVIS*, 2008. http://www.ni.com/academic/ni_elvis/.

[9] John Odyek. East Africa: Submarine Network to Lower Internet Costs, 2008. http://allafrica.com/stories/200804230084.html.