Modular Development of an Educational Remote Laboratory Platform for Electrical Engineering: the ELVIS iLab

by

Adnaan Jiwaji

B.S. in Electrical Science and Engineering
Massachusetts Institute of Technology, 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

August 2008

Signature of Author: _____
Department of Electrical Engineering and Computer Science
August 15, 2008

Certified by: _____
Judson Harward
Associate Director, Center for Educational Computing Initiatives
Thesis Supervisor

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

Modular Development of an Educational Remote Laboratory Platform for Electrical Engineering: the ELVIS iLab

by

Adnaan Jiwaji

Submitted to the Department of Electrical Engineering and Computer Science
on August 15, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

iLabs are remote online laboratories that allow users to perform experiments through the Internet. As an educational tool the iLab platform enables students and educators, who do not have access to laboratories, to complement their theoretical knowledge by carrying out experiments remotely on equipment located anywhere in the world and at any time of the day. Students perform experiments on actual instruments allowing them to get real data, instead of relying on simulations. The iLab project has been deployed in 3 universities in Africa using the National Instruments Educational Laboratory Virtual Instrument Suite platform which is a cheap all-in-one electronics workstation for electronics experiments. This thesis describes an increase in the functionality available on the current version of the ELVIS iLab in order to enable a wider range of experiments to be run on the platform. The functionalities explored include adding two arbitrary waveform generator channels and bode analyzer for frequency domain analysis, which was not possible in the previous designs.

Thesis Supervisor: Judson Harward
Title: Associate Director, Center for Educational Computing Initiatives

Acknowledgements

I am extremely grateful to my thesis advisor Dr. Judson Harward for giving me the opportunity to work on this project. He has provided constant support and feedback throughout the MEng year and his help was critical in making sure I finished my thesis work in good time.

I would like to thank Prof. Jesus Del Alamo who together with being my academic advisor is deeply involved in the iLab project. His guidance and advice on my thesis work and other activities has helped in making the MEng year an unforgettable experience.

I would also like to thank Bryant Harrison for introducing me to the project and ensuring I started off on the correct track. I am also highly indebted to Jim Hardison, Kimberly DeLong, Phil Bailey, Meg Westlund, Maria Karatzas and the entire iLabs team for the technical and personal help they provided over the year. I couldn't have asked for a more supportive team; it would not have been a smooth ride without them.

The iLabs teams at the University of Dar-es-salaam including Prof. Nzali and Dr. Mwambela and at Makerere University including Dr. Tickodri, Andrew Katumba, Lordewin Muhwezi and Michael Rutalo always made sure our trips went smoothly and provided challenging discussions that helped further the project.

I would like to thank my parents for all the support they have provided me over the year. My father helped us a lot in our trips to Dar-es-salaam and made sure we were comfortable and provided time out of his schedule to cater to our demanding requests at times. My mother made sure we were all well fed and healthy. Lastly thanks go to my brother and my friends at MIT for pushing me to finish and providing company and moral support.

Contents

1. Introduction ……... 13

1.1 Background on iLabs…….....……. 14

1.2 Background on iLab-Africa Project…… 17

1.3 Background on NI ELVIS ……..... 19

1.4 Overview of thesis ……... 22

2. Motivation to develop ELVIS iLab Version 3.0 ……. 25

2.1 iLabs Shared Architecture ……... 25

2.1.1 Client ……... 26

2.1.2 Service Broker ……..… 27

2.1.3 Lab Server …….. 29

2.1.4 Interconnection between ISA components in the Batched Architecture ……...……. 30

2.2 Background on Previous ELVIS Versions ……...…….. 31

2.2.1 ELVIS Version 1.0 …….. 31

2.2.2 ELVIS Version 2.0 …….. 34

2.3 Background and Overview of ELVIS Version 3.0 ……. 35

3. Design of ELVIS Version 3.0 ……... 39

3.1 Lab Server …… 39

3.1.1 Lab Server LabView ……. 42

3.1.2 Experiment Engine ……. 49

3.1.3 DLL Wrapper (OpAmpInverter) ……. 50

3.1.4 Validation Engine ……..…… 51

3.1.5 Lab Server Administrative Interface and Database…… 54

3.2 Client …… 55

3.2.1 Main flow of information in the client ……. 55

3.2.2 Changes made to the client …… 58

3.3 Testing using new circuits ……. 62

4. Conclusion and Recommendations for future work …… 65

4.1 Conclusion ……. 65

4.2 Recommendation for future work ……. 65

4.3 Progress at UDSM and MUK …….. 67

A. LabConfiguration.xml ……... 69

B. ExperimentSpecification.xml…….. 70

C. **ExperimentResult.xml ……..……... 72**

List of figures

1-1: Microelectronics Device Characterization equipment housed at MIT. …...…….. 16

1-2: Microelectronics WebLab client used to run remote experiments on the Microelectronics Device Characterization equipment. ……..… 16

1-3: The National Instruments ELVIS electronic workbench. ……... 20

1-4: Schematic diagram of the NI-ELVIS showing the features available on the platform. …..…… 20

1-5: The software interface that is provided with the NI-ELVIS hardware to access its instruments from a computer. ……... 21

2-1: The iLab Shared Architecture showing the various components of the architecture and where the ELVIS is plugged into the architecture. ……. 26

2-2: The Service Broker webpage. It shows the various labs that students in each group (underlined) have access to. …….. 28

2-3: The modified weblab client developed by Gikandi to run experiments on the ELVIS. The screenshot shows running an experiment on a differentiator circuit with a triangular wave input. ……... 33

2-4: Addition of a switch allows users to choose the component value that they would like to place in the circuit. …… 35

3-1: The lab server components consisting of the execution code, validation engine, database and administrative pages. It interacts with the client through the Service Broker. ...…… 41

3-2: The LabView VI hierarchy. ……. 43

3-3: Part of the ARB VI showing the sine wave simulator and the input parameters it takes. ……. 46

3-4: The formula generator VI. …… 46

3-5: Generating a waveform from an array of y values. ……...…... 47

3-6: The bode analyzer VI developed using the Express VI to run frequency analysis in a certain defined frequency range. …….. 48

3-7: Experiment Specification parser. ……. 49

3-8: The execution cycle showing details of the lab server. ……. ……. 53

3-9: The lab server administration interface where the creator of an experiment can choose what terminals and instruments to use in the experiment and specify constraints for the instruments. …… 54

3-10: Different parts of the Java Client. …… 57

3-11: Each feature on the ELVIS is an Instrument. All features inherit from the Instruments. …… 58

3-12: Associated with each instrument is a function. Functions for each instrument are sub classes of the SourceFunction class. …… 59

3-13: The dialog box to specify parameters for the bode analysis. …….. 60

3-14: In the ARB dialog box users can choose the type of waveform they would like to use and specify parameters for them. They can also enter a formula or load a file to specify the wave form. ……. 60

3-15: The UML diagram shows how the ARB instrument is related to the ARBDialog class that draws the dialog box and the ARBLabel class. …… 61

3-16: This shows the waveform generated by loading a .csv file that was created in MS Excel. It generates a unique waveform that won't be found in any pre-defined library and for which a formula doesn't exist. ...…….. 62

3-17: Using the ARB feature to run an experiment on an adder circuit. A and B shows the two input waveforms and C shows the output of the adder circuit. Such a circuit with two inputs wasn't possible in ELVIS version 1.0. …….. 63

3-18: The various bode analyses done using the bode analyzer feature. A and B show the magnitude and phase plot of a Sallen-Key band pass filter. C and D show the magnitude response of a high pass and low pass filter respectively. ……. 64

4-1: Picture taken at the iLabs workshop at UDSM on 22nd January 2008. ……..…. 68

CHAPTER 1

Introduction

Laboratory experiments are an integral part of any science or engineering education. Laboratory experiments provide students with practical experience that can help them better understand the theory taught in class. However, students often don't have sufficient access to such equipment. This is because traditional laboratory facilities are usually expensive to set up and maintain. Enough equipment has to be bought to accommodate all the students taking a lab course. This can be difficult if expensive equipment is required. It can also lead to overcrowding of the laboratory. Students may have to stand in queues to wait their turn to use the equipment. Laboratory personnel also need to be hired to man the facilities, thereby imposing additional costs.

Even when these facilities are available laboratories are inefficient because they are usually accessible to students only during standard business hours, and hence the pieces of equipment are not fully utilized. Students can also damage equipment as they have unlimited access and control over the apparatus and may use settings that do not meet the specifications of sensitive devices.

Sometimes traditional laboratories don't provide the intended experience as students get caught up in details of setting up the equipment, dealing with faulty equipment, or taking intensive repetitive readings. Such students never get to focus on the concept that the experiment was designed to teach. With close supervision and guidance such problems can be avoided; however it is not practical to do this for each student.

By providing students remote access to laboratory equipment, the hurdle of costly traditional laboratories can be overcome by sharing the few laboratory resources available. Development of such laboratories is especially useful in developing countries where funds for education resources are hard to come by.

1.1 Background on iLabs

iLabs are remote laboratories developed at MIT in order to address the shortcomings of conventional laboratories. It is a technology that allows experimental setups to be accessed remotely through the Internet, allowing students and educators to carry out experiments from anywhere at any time [1]. Users are able to access the laboratory experiments whenever and wherever they want, bypassing the need to acquire expensive equipment or for students to wait to get access to it. Students get to use real instruments, rather than simulations, using a standard web browser. iLabs allow students to complement their theoretical calculations and results with real data, providing them with a better understanding of concepts in science and engineering.

Unlike conventional experimental facilities, iLabs greatly reduce the cost of a laboratory, because only one piece of equipment that can be shared by multiple users, is required. iLabs can be shared and accessed widely by students and other users around the world. While anyone can access the laboratories, only a few institutions need to maintain the equipment. This also allows the system to scale well. The iLab Project envisions a global network of iLabs to which institutions contribute their own laboratories and experiments in order to share them with other institutions.

The iLab Project was started at MIT in 1998 by Prof. Jesus Del Alamo to develop a remote laboratory for semiconductor device courses at MIT that did not have a laboratory component at that time. This led to the development of the Microelectronics WebLab[1] experiment that allowed students to remotely conduct device characterizations on an Agilent 4155B semiconductor parameter analyzer (Figure 1-1 and Figure 1-2) [2]. Since then various remote experiments have been developed in many different engineering fields at MIT. These include the Dynamic Signal Analyzer (2004) in electrical engineering, Heat Exchanger (2001) and Polymer Crystallization (2003) in chemical engineering, Shake Table (2003) in civil engineering and the Nuclear Reactor (2008) in nuclear engineering [3]-[6].

These iLabs were developed by Prof. Del Alamo and others faculty members who felt that their courses lacked a sufficient laboratory component [2]. Earlier on the iLabs were developed independently using different software approaches. It was not until 2002 that a standardized architecture was developed for creating online labs [6]. This architecture, known as the iLab Shared Architecture, is described in detail in Section 2.1.

During the past several years, MIT iLabs have been used worldwide in courses at over 18 universities in Europe, Asia, North America, Australia and Africa. Some of these institutions are also developing their own labs, adding to the growing number of experiments in the iLabs family. For example the University of Queensland in Australia is developing a radioactive half-life measurement experiment and is already using an inverted pendulum control experiment [2].

[1] Also called the Microelectronics Device Characterization experiment

Figure 1-1: Microelectronics Device Characterization equipment housed at MIT.

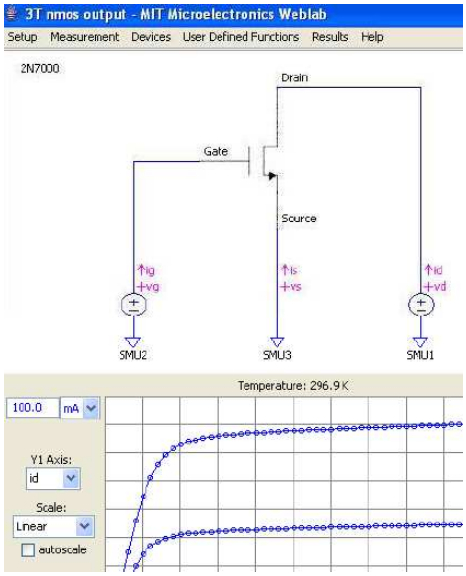


Figure 1-2: Microelectronics WebLab client used to run remote experiments on the Microelectronics Device Characterization equipment.

Remote laboratories like iLabs, however, have some shortcomings over traditional laboratories. The labs are run remotely by students so they are usually not able to handle the actual hardware equipment. Hence students do not get the necessary experience in setting up and debugging the experiment setup. For example, an instructor wires the circuit for the students in the Microelectronics Device Characterization

experiment, hence the student does not get to see and wire the device under test. Therefore, iLabs are intended to supplement traditional hands-on laboratory assignments where available. In places where the number of laboratory equipment is small compared to class size, iLabs assignments can be given more regularly together with less frequent hands-on assignments.

1.2 Background on iLab-Africa Project

Due to the scalable and economically efficient nature of iLabs, they have sparked significant interest from institutions of higher learning in Africa. iLabs are a particularly useful concept for universities in Africa [7] as well as the developing world in general. Such institutions usually do not have the resources to purchase and operate costly or delicate lab equipment. Students miss out on crucial hands-on learning experiences and are relegated to learning through solely theoretical or simulative methods. iLabs can substitute for expensive laboratory equipment in developing countries, and give students experience with real lab data.

In 2005 a major partnership was established between MIT and three African universities to form the iLab-Africa Project. The three universities are Obafemi Awolowo University (OAU) in Ile-Ife, Nigeria, University of Dar-es-Salaam (UDSM) in Dar-es-Salaam, Tanzania and Makerere University (MUK) in Kampala, Uganda. It is supported by a grant from the Carnegie Corporation of New York. The aim of the iLab-Africa project was to exploit the potential of online laboratories (iLabs) in Sub-Saharan Africa. The iLab-Africa project attempts to inject iLabs through the curricula in Africa to enrich science and engineering education. A feasibility study conducted between 2003 and 2004

to determine whether iLabs could be useful in sub-Saharan Africa found that electrical engineering was one of the main areas that lacked laboratory facilities and that could easily utilize existing iLabs experiments from MIT like the Microelectronics Device Characterization experiment [2].

The partnership has enabled these universities to use the iLabs housed at MIT. It gives students in these countries access to high cost equipment that local institutions cannot afford. Over the course of this project, a total of 694 students at UDSM and OAU have used labs at MIT in several laboratory assignments [2]. However, effective implementation of iLabs in Africa has been impeded by slow internet speeds. East Africa does not have a fiber optic cable connecting it to the rest of the world; hence Tanzania and Uganda have to rely on slow and expensive satellite connections. These countries are bandwidth starved, as they usually cannot afford to pay for large bandwidth connections. For this reason students working with iLabs in those countries cannot efficiently access lab equipment housed at MIT.

To solve this problem, new labs that integrate low cost equipment that could be housed locally at each university were created. Consequently, these labs could be accessed using the high speed local area networks (LAN) avoiding the need to have high speed internet access. This was possible because each university has a good internet LAN infrastructure (usually with a fiber optic back-bone). Hence, instead of relying on the internet to access hardware at MIT, each university can use its LAN to access the hardware at their university. This avoids the bottle-neck of accessing the internet through a small bandwidth satellite connection. It is also more convenient as they have access to

the hardware and can easily change the experiments they would like to run on the hardware instead of coordinating with MIT every time they need to change the setup.

The need for electrical engineering laboratory facilities in these countries combined with the fact that the iLabs team has historically been strong on developing electrical engineering remote laboratories led to a search for new inexpensive lab equipment that could be used for electronics experiments. The platform that was chosen for this purpose was the affordable National Instruments Educational Laboratory Virtual Instrumentation (ELVIS) hardware platform that has since been installed at the universities in Nigeria, Tanzania and Uganda. The ELVIS is a compact and cost effective platform to implement various electrical engineering labs. Each university is encouraged to develop the ELVIS platform to meet the needs of its curriculum. Development efforts are then shared among partner institutions. The approach has encouraged strong collaboration between MIT and the African partners to work together to develop new software and hardware features for the ELVIS platform. This has led to many students and staff exchanges for training and development purposes.

1.3 Background on NI ELVIS

The ELVIS (Figure 1-3) is an electronic workbench that is used to design and test circuits. It offers twelve hardware instruments that can be used for circuit analysis and debugging [8]. These instruments include: Oscilloscope, Digital Multimeter, Function Generator, Variable Power Supply, Bode Analyzer, Arbitrary Waveform Generator, Dynamic Signal Analyzer, Impedance Analyzer, Digital Reader, Digital Writer and Microelectronics Device Characterization capability.

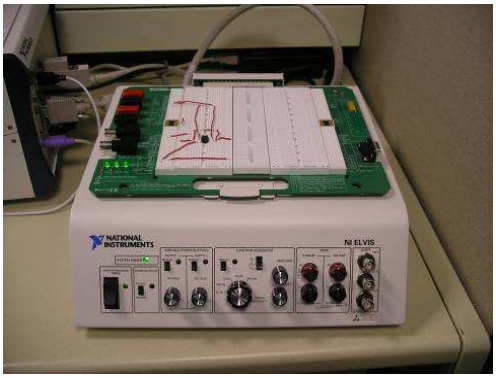


Figure 1-3: The National Instruments ELVIS electronic workbench.

Figure 1-4 shows an annotated diagram of the channels and instruments available on the ELVIS board. The ELVIS has a removable prototype board that can be used to build circuits which students can connect to the various ELVIS instruments.

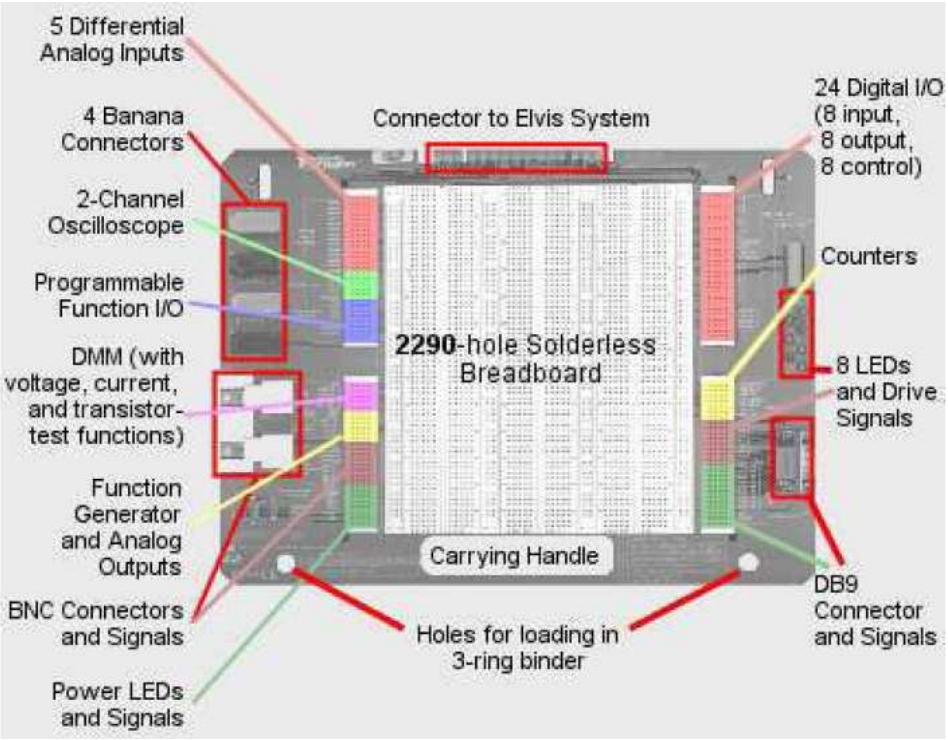


Figure 1-4: Schematic diagram of the NI-ELVIS showing the features available on the platform [9].

The ELVIS hardware can be connected to a computer via a DAQ card. It comes with a software suite that can be used to access the different instruments available on the ELVIS from the computer. It also comes with API modules that make the ELVIS completely open and customizable in the LabVIEW graphical programming environment. Thus, the ELVIS instruments can be programmed to be accessed remotely.

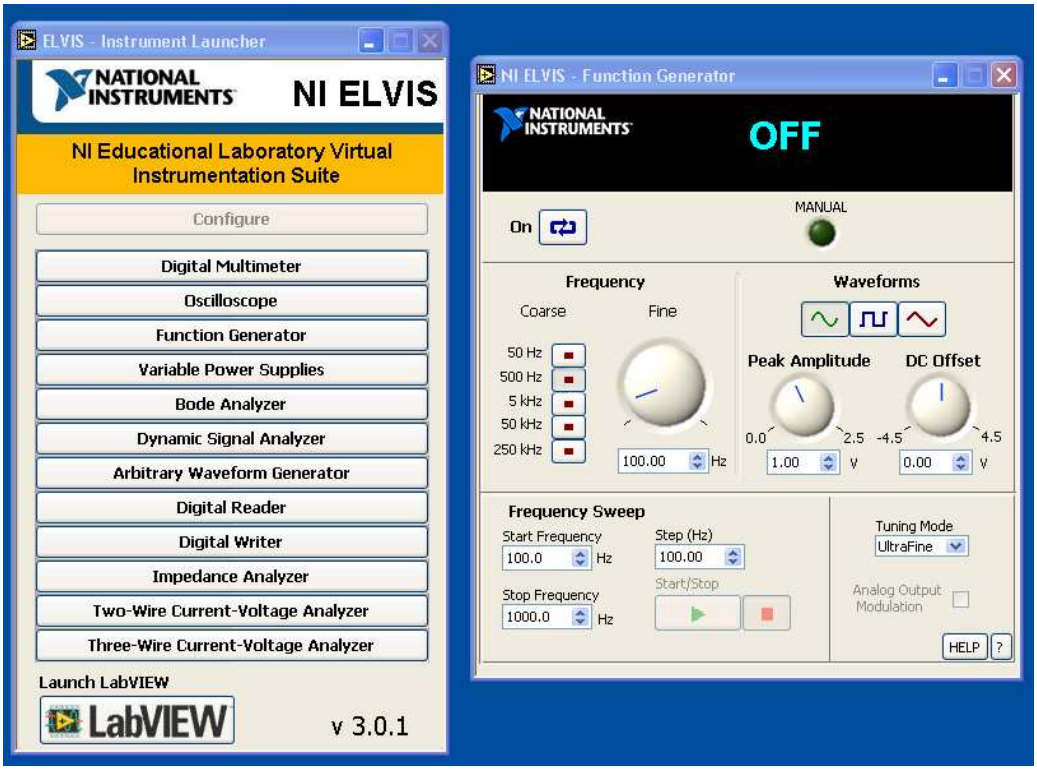


Figure 1-5: The software interface that is provided with the NI-ELVIS hardware to access its instruments from a computer.

The ELVIS is ideal for university education because all the components a student needs for basic electronics experiments are on the platform. The ELVIS is portable and more compact than traditional instruments, making it ideal for the normally limited space available for laboratory equipment. It is also affordable for our African partners; each

unit costs about USD 2000. Using the iLab architecture to access the ELVIS remotely, it can be used by many students, thereby significantly lowering the cost per student.

1.4 Overview of the thesis

The focus of my research has been further feature development of the ELVIS iLab platform for electrical engineering application in Sub-Saharan Africa. The ELVIS hardware platform was originally integrated into the iLab architecture by Samuel Gikandi in 2006 in version 1.0 [10]. This is the current version that is deployed at the African universities, although some of the institutions have made some modifications to the original version. Many of the capabilities in the ELVIS platform were left unexplored in the original version. This thesis aims to look at how some of these capabilities can be used to increase the utility of the ELVIS platform and the range of experiments that can be done on it.

This thesis describes the development of a version 3.0 of the ELVIS iLab. 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 during visits to UDSM and MUK. This work was done parallel to development of ELVIS version 2.0 by Bryant Harrison [11].

Chapter 2 will focus on the reasons for creating a new version of the ELVIS iLab. It includes an overview of the iLab Shared Architecture (ISA) into which the ELVIS platform is integrated. It also describes the original version of the ELVIS iLab and the limitations that encouraged further developments that were made in the ELVIS version 3.0.

Chapter 3 goes into the details of the design for ELVIS iLab version 3.0. It includes a technical description of all the components of the ELVIS iLab architecture and the changes that were made to the various components. It also discusses design decisions that were made throughout the process and reasoning behind them. In the end I will describe some examples of circuits that can now be studied that were not possible in the previous version of the ELVIS iLab.

Chapter 4 discusses the limitations that still exist in the ELVIS version 3.0. It also makes recommendations for future development on the ELVIS platform. Furthermore, it discusses work and progress made while on visit to UDSM and MUK.

CHAPTER 2

Motivation to develop ELVIS iLab Version 3.0

The ELVIS hardware platform is built on the iLab software architecture also known as the iLab Shared Architecture (ISA). This is a web service infrastructure that was developed at MIT starting in 2002 that allows remote access to various types of experimental equipment [12]. It is important to understand this architecture because it forms the backbone of the ELVIS platform. ELVIS version 1.0 and 2.0 will then be described.

2.1 iLab Shared Architecture

The original iLabs developed between 1998 and 2002 had a client communicating directly with a server connected to the hardware [2]. This server managed student accounts and implemented student logins. However as the usage of iLabs increased at MIT and as other institutions started using them, having a single point of account management became an administrative bottleneck. This was especially true for the Microelectronics WebLab experiment. It was also difficult to share development efforts among faculty members, who independently made their own remote labs using varied approaches, and this sometimes led to duplicate development work [6]. There was a need to create a more standardized infrastructure that would provide an easy framework for developing and sharing new experiments, hence the development of the iLab Shared Architecture. The ISA has been continuously modified to increase scalability and

usability since the first version. In its current state, the iLab architecture (Figure 2-1) consists of a three main software parts: client, Service Broker and lab server. These components form a unified framework to which various hardware lab devices can be connected to so that they can be accessed remotely.

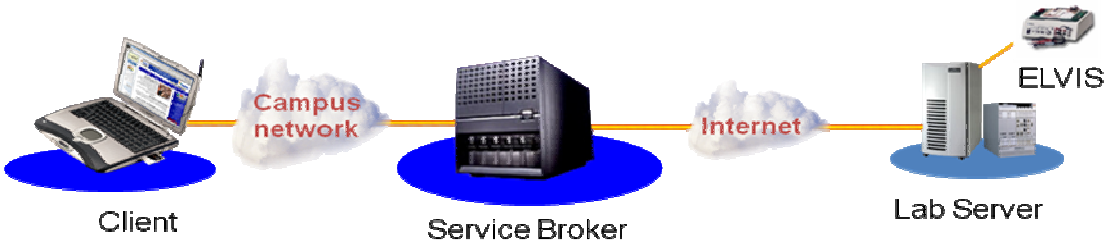


Figure 2-1: The iLab Shared Architecture showing the various components of the architecture and where the ELVIS is plugged into the architecture [13].

2.1.1 Client

The client in the ISA is the graphical interface that the student uses to access the experiment setup. The design of the interface usually depends on the hardware used to implement the experiment. For example, an electrical engineering experiment in which you are trying to run a circuit, the client would show the schematic representation of the circuit that is wired on the hardware. The client can take different forms; it can either be implemented as an application to run on the student's own workstations, or it can be run online as of .NET Windows Forms or HyperText Markup Language (HTML) clients. A student uses the client to specify the experiment parameters they want to run. The student's specification is then passed through the Service Broker to the Lab Sever via an XML file.

2.1.2 Service Broker

The Service Broker mediates exchanges between the Lab Client and the Lab Server passing information in XML format between the two. It also provides storage and administrative services that are generic and can be shared by multiple labs within a single university. The Service Broker manages access to different labs. One Service Broker can be connected to many lab servers. Usually each institution will have its own Service Broker housed locally. From this Service Broker they can connect to labs that can either be on local lab servers or on lab servers at other institutions anywhere else in the world.

This is also the interface that is used to manage student accounts. Each student is given an account on the Service Broker and the administrator gives the student rights to access certain labs but restrict access to other labs depending on the class and course the student is registered in. To start a session, a student logs in to the Service Broker using his/her account information. There, the student can choose an experiment to run. This launches the client for the chosen lab. The client communicates with the Service Broker (Figure 2-2) using a “Service Broker to Client” web service. This web service is used to transfer the user specification of the experiment to be run to the Service Broker. The Service Broker then uses another set of web services to transfer this information to the lab server where the experiment is run on the hardware.

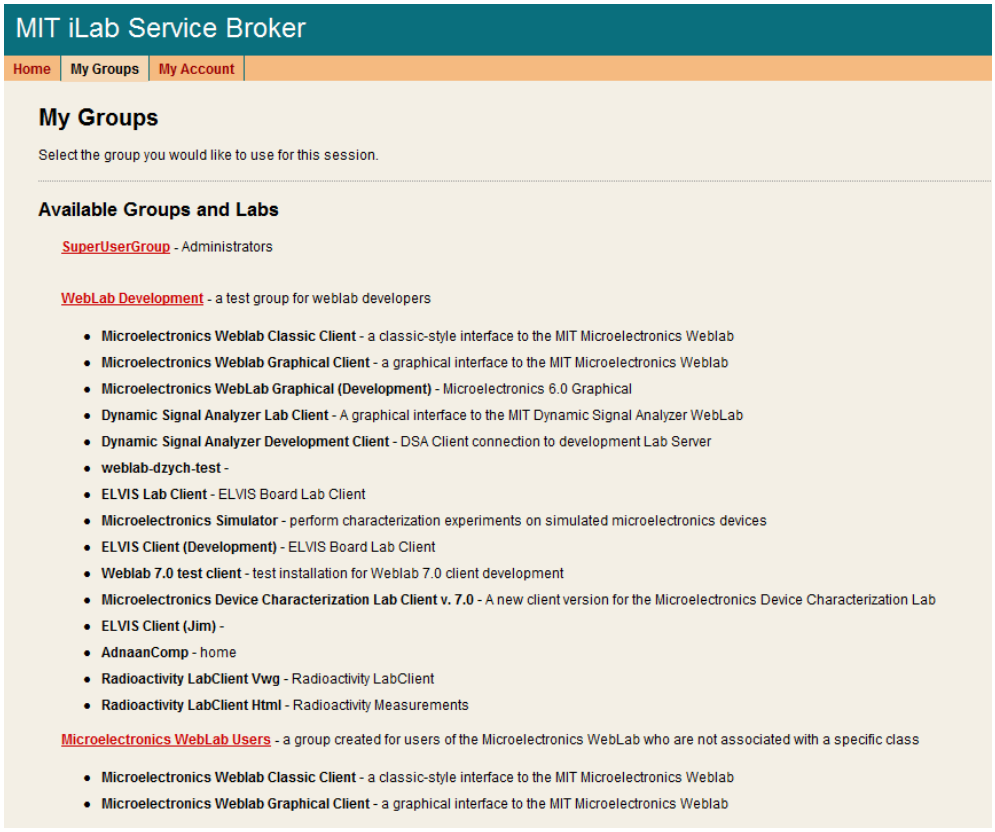


Figure 2-2: The Service Broker webpage. It shows the various labs that students in each group (underlined) have access to.

There are two different sets of web services that define the type of Service Broker. The two types of Service Brokers are the batched and interactive architectures. In batched Service Broker, users specify the parameters for the experiment that are submitted to the lab server without actually accessing the hardware resource. The submitted experiment specification is then queued up at the lab server and the submitted experiments are run in a FIFO (First In First Out) manner. Once an experiment is run, the results are transmitted to the user and the next experiment is run. Given that each user only needs a few seconds of the equipments time, this is a good model for large scale classes.

The interactive Service Broker gives users access to laboratory equipment while they are running the experiment. Users reserve time blocks in which they are allowed exclusive access to the lab equipment. This way they can control the laboratory equipment in real time and get real time output. This is similar to going to the actual lab and waiting your turn to use the equipment. Hence, although interactive labs can appear more realistic, they possess some of the same limitations as traditional labs. If used for time intensive experiments in large classes, they can lead to long queues. This architecture also demands more bandwidth since the client interface tends to be more complex, and changes in experiment parameters are affected in real time from the user to the hardware just as experiment results are immediately returned to the client.

The ELVIS uses the batched architecture because it is also a more appropriate design for use in the developing world where the number of users per lab setup tends to be higher. Internet connections tend to be slow and unreliable, so bandwidth intensive interactive experiments are not possible. The lack of reliable electric power causes scheduling of equipment time to be chaotic because students are not sure if they will be able to access the experiment in the time they reserved. Finally, the ELVIS is most suited for introductory electronics classes that can have close to 300 people registered at one time. This makes the fast batched architecture the better choice.

2.1.3 Lab Server

The lab server is the machine connected directly to the ELVIS platform using a Data Acquisition (DAQ) card. It has software that communicates with the Service Broker

to receive information the user entered in the client. It sends this experiment specification to the ELVIS where the experiment is run. The results are then returned to the user. ·

The lab server also serves as an administrative interface where you can create experiments. The lab server has a database connected to it that serves as persistent storage for all the experiments that can be run on the lab server. The database also stores a log of experiment execution requests and manages administrator accounts. To facilitate sharing of labs between institutions each lab server can be connected to multiple Service Brokers so that the associated experiments can be run from many institutions.

The lab server must execute code specific to the hardware connected to it. In general, each piece of experiment hardware connected to the iLab architecture needs a separate set of drivers to access the hardware. Hence, building an iLab that is based on a new piece of lab equipment typically requires adapting both the client and the lab server code.

2.1.4 Interconnection between ISA components in the Batched Architecture

As mentioned earlier the ELVIS iLab uses the batched ISA architecture. This architecture requires lab specific messages to be passed between the lab client and lab server. This is done via the lab client/server communication framework (LC/SCF). The LC/SCF encodes the lab-specific information that is relayed between the lab clients and servers using the generic mechanisms [2]. XML is currently the preferred technology for encoding this information because it can be transmitted as plain text. The ELVIS iLab uses three XML files: the *LabConfiguration*, *ExperimentSpecification* and

ExperimentResult files. When the user logs on to the Service Broker to launch an experiment a *LabConfiguration* file (See Appendix A) is passed from the lab server connected to the required hardware through the Service Broker to the client. It contains information about the experiment such as the name, file path for the image representation of the experiment, and a brief description. It also has a list of the components that make up the experiment and their information such as name and pixel location. This XML file is used in the client to display a schematic drawing of the experiment that users can use to configure the various inputs to the experiment.

When the user is done specifying the parameters that define the experiment in the client, the experiment can be run. An *ExperimentSpecification* document (See Appendix B) is created with all the configured values and sent to the lab server through the Service Broker. The *ExperimentSpecification* is parsed and executed on the lab server. Once the experiment is finished running, the results are packaged in the *ExperimentResult* document (See Appendix C) and sent back to the client through the Service Broker. The client's graphing tools then use this information to display these results to the user.

When designing a new iLab the developer must make sure that the client and the lab server are able to produce and interpret the XML documents according the LC/SCF for that specific lab.

2.2 Background on Previous ELVIS Versions

2.2.1 ELVIS Version 1.0

The development of ELVIS version 1.0 was the thesis work of Samuel Gikandi (MEng '06). It involved integrating the National Instruments-ELVIS platform into the

current iLab and allows for simple preconfigured circuits setup on the ELVIS to be run remotely. The circuit to be tested is setup on the ELVIS prototyping board. Wires then connect the circuit to the ELVIS instrument to test the circuit according to parameters chosen by the student. An easy to use client was developed in Java to manage this process (Figure 2-3).

Most of the circuits that have been used so far are op-amp circuits. The ELVIS lab is currently deployed for use in courses including Circuits and Electronics (6.002). Version 1.0, however, had a number of limitations. It allowed only one setup per board, possessed only one input and output, and exposed only two hardware instruments on the ELVIS, the Function Generator (FGEN) and the Oscilloscope (SCOPE). As mentioned in Chapter 1 the ELVIS has ten additional instruments. The client developed by Gikandi is a modified version of the client used in the Microelectronics Device Characterization lab (Figure 1-2). The client shows the schematic diagram of the circuit set up on the ELVIS board. The user can click on the FGEN and SCOPE instrument to configure them. For the FGEN instrument the user can choose what type of waveform to input into the circuit. The options available are a Square, Sine or Triangle waves. The user can then specify the amplitude, frequency and offset of the waveform. For the SCOPE the user can choose the sampling rate and sampling time in a way that will meet the sampling requirements of the output waveform. After specifying the instruments the user can run the experiment. The user specification is passed through the Service Broker to the lab server. The experiment is run on the ELVIS hardware using the specifications provided and the results are passed back to the client where the output waveform is displayed on the client.

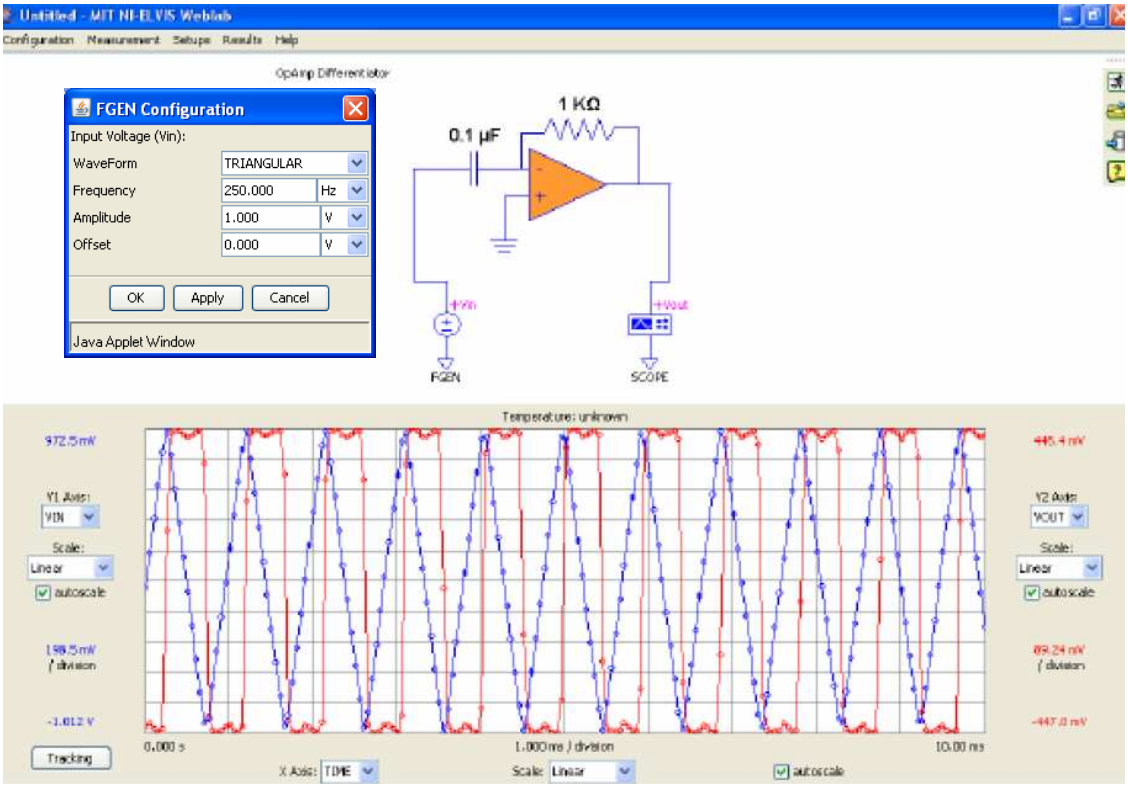


Figure 2-3: The modified weblab client developed by Gikandi to run experiments on the ELVIS. The screenshot shows running an experiment on a differentiator circuit with a triangular wave input.

This version of the ELVIS achieved a great deal in terms of getting the ELVIS hardware incorporated into the iLab architecture. However there were significant limitations with this initial version that prompted the development of ELVIS version 2.0 and 3.0. For one, you can't run any circuit that requires more than one input waveform. Also the user is limited to only the three waveforms available in the FGEN instrument.

Besides this the student is not given any flexibility in the design of the circuits. The circuits are static as designed by the creator of the experiment and hard-wired on the ELVIS bread-board. This definitely reduces the learning experience as students cannot experiment with different configurations and circuit values. The student is also limited to the view of the output waveform specified by the creator. The student cannot probe the

circuit to view the waveform at different nodes, a feature possible with a traditional oscilloscope. Also with the first ELVIS version only one circuit can be used at a time. If a different circuit is run, it would need to be physically rewired to connect the FGEN and SCOPE instruments to the new circuit. With switching the rewiring can be done automatically.

2.2.2 ELVIS Version 2.0

The first modification to the initial ELVIS version was the thesis work of Bryant Harrison. Bryant introduced computer controlled switches into the setup to allow for multiple setups per board. This will also allow students to use switches to choose different component values and provide more flexibility in circuit design. He has also exposed the variable power supply feature to allow the user to input a DC voltage.

Adding the ability to switch components in and out of a circuit setup greatly enhances the capabilities of the ELVIS iLab. There are switching modules made by National Instruments that are compatible with the LabVIEW software that composes part of our ELVIS lab server. Using a National Instruments SCXI-1169 100-channel switch, the module can be used to potentially have 100 different components available in a circuit. This gives the administrator the option of creating multiple setups on one ELVIS prototyping board or having different types and values of circuit components (such as resistors or capacitors).

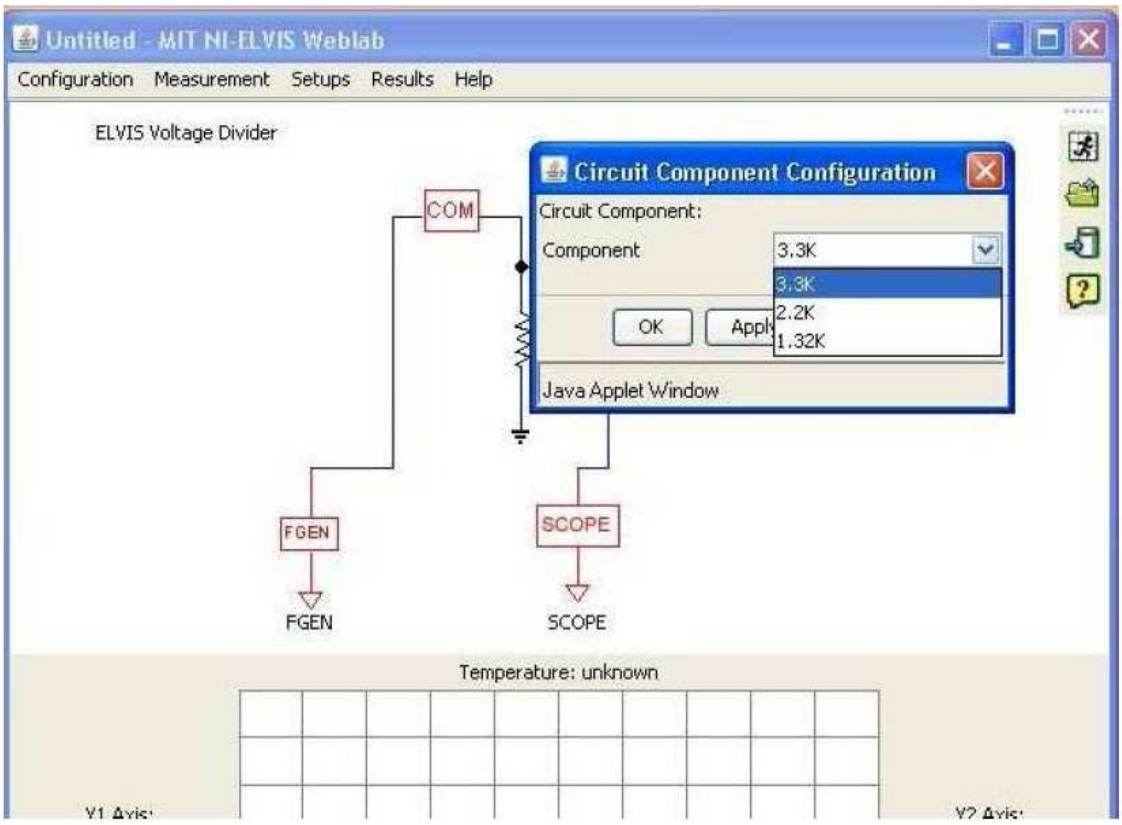


Figure 2-4: Addition of a switch allows users to choose the component value that they would like to place in the circuit [11].

2.3 Background and Overview of ELVIS Version 3.0

To enable a larger variety of experiments to be run on the ELVIS, the next step is to extend all the functionalities available on the ELVIS to the end user. The ELVIS platform offers quite a few capabilities that are still not exposed to the user. These include:

- Digital Multimeter (DMM)
- Arbitrary Waveform Generator (ARB)
- Dynamic Signal Analyzer (DSA)
- Bode Analyzer (BODE)
- Digital Reader

- Digital Writer
- Impedance Analyzer
- Two-wire and Three-wire Current-Voltage Analyzer

To increase the variety of experiments that can be done on ELVIS, I chose to add the Arbitrary Waveform Generator (ARB) and Bode Analyzer (BODE) instruments. Although it would have been useful to add more instruments I limited myself to two because of the limited time and also because this project required additional commitments including travelling to UDSM and MUK to train the iLabs teams there and also training them when they visited MIT.

The ARB adds two more input channels to the existing FGEN channel provided in version 1.0. The two ARB channels are identical and can be used independently or together. This enables users to design circuits that use up to three channels (2 ARB channels and 1 FGEN channel).

The ARB also provides extra flexibility in choosing the input waveform to be used. The FGEN feature in ELVIS version 1.0 provided the option of a Sine, Square or Triangular waveform for which you can specify the amplitude, frequency, phase and offset. The ARB feature allows users to choose from the predefined options of either: Sine, Square, Triangular or the additional option of a Sawtooth waveform. Unlike the FGEN, the ARB also allows users to choose a duty cycle for the Square wave allowing the user an option of generating a Square waveform with a 1% to 100% duty cycle. Additionally, the user can generate an input waveform from a MatLab syntax formula or by inputting a file that uses values to define the waveform. This gives the user wider options on the type of input that they can apply to their circuits.

The current ELVIS versions only allow for time domain analysis on circuits. However one of the most important analyses that electrical engineers need to do is frequency domain analysis of circuits to determine how the circuit responds to various input frequencies. That is why the addition of the BODE feature tremendously increases the utility of the ELVIS.

CHAPTER 3

Design of ELVIS Version 3.0

In this section I will describe the technical structure of various parts of the iLab software architecture and changes made to them. The careful compartmentalization of the various ISA components allows a modular approach to making changes to the current iLab architecture. The various modules can be executed and tested separately.

3.1 Lab Server

The lab server is the part of the ISA to which the hardware is connected. The lab server plays a major role in the creation and execution of experiments. The lab server has driver software that interacts directly with the ELVIS hardware instruments such as the Oscilloscope, Digital Multimeter (DMM), Function Generator, Variable Power Supply, Bode Analyzer and Arbitrary Waveform Generator. For the rest of the chapter I will refer to them as instruments.

The lab server also has backend code that is used to interpret the experiment specifications received from the Service Broker to the ELVIS drivers. It also has an administrative interface and persistent storage for setting up and storing experiments. Figure 3-1 shows the structure of the ELVIS lab server and how it connects to the client, Service Broker and the ELVIS hardware [2].

You can use the administrative interface of the lab server to create an experiment setup. A setup is the experiment circuit together with the terminals and instruments present in the experiment. The Differentiator circuit in Figure 2-3 is an example of an

experiment setup. It has an input terminal that the FGEN instrument connects to and an output terminal that the SCOPE instrument connects to. You can create an experiment setup by loading an image of the circuit, creating terminals and specifying the hardware instruments that connect to each terminal. You can also specify any constraints on the instruments.

This experiment setup is stored in the form of a *LabConfiguration* XML file in the lab server database. When a student launches the client, the Service Broker fetches that lab configuration from the lab server database and passes it to the client. The client uses the configuration to show a schematic representation of the experiment circuit to student. The student then configures the instruments available in that specific circuit. For example in the case of the Differentiator circuit, the student could configure the FGEN instrument with the type of waveform to input together with its amplitude, frequency and offset. They can then submit the experiment to the lab server for execution.

At that point an *ExperimentSpecification* XML file is created by the client with all the values specified by the student and is sent to the lab server. Before the experiment is submitted for execution the validation engine checks the configuration to make sure the values used by the student meet the constraints specified by the educator. For example in the Differentiator circuit case, it would check the student's wave amplitude value to ensure that it is less than a certain maximum voltage. This is to prevent the hardware (Operational Amplifier) from being damaged. The validation engine then stores the *ExperimentSpecification* file in the database.

The experiment engine retrieves the experiment specification from database and parses it to determine the parameters specified by the student. For the case of the

Differentiator experiment in Figure 2-3, the parser would extract the FGEN waveform as Triangular wave with a frequency of 250Hz, amplitude of 1V and offset of 0V. The experiment engine then passes these parameter values along to the DLL wrapper class. The wrapper class calls the LabView drivers of the ELVIS hardware instruments that run the experiment on the ELVIS with the given parameters. Figure 3-1 shows how all these lab server modules fit together.

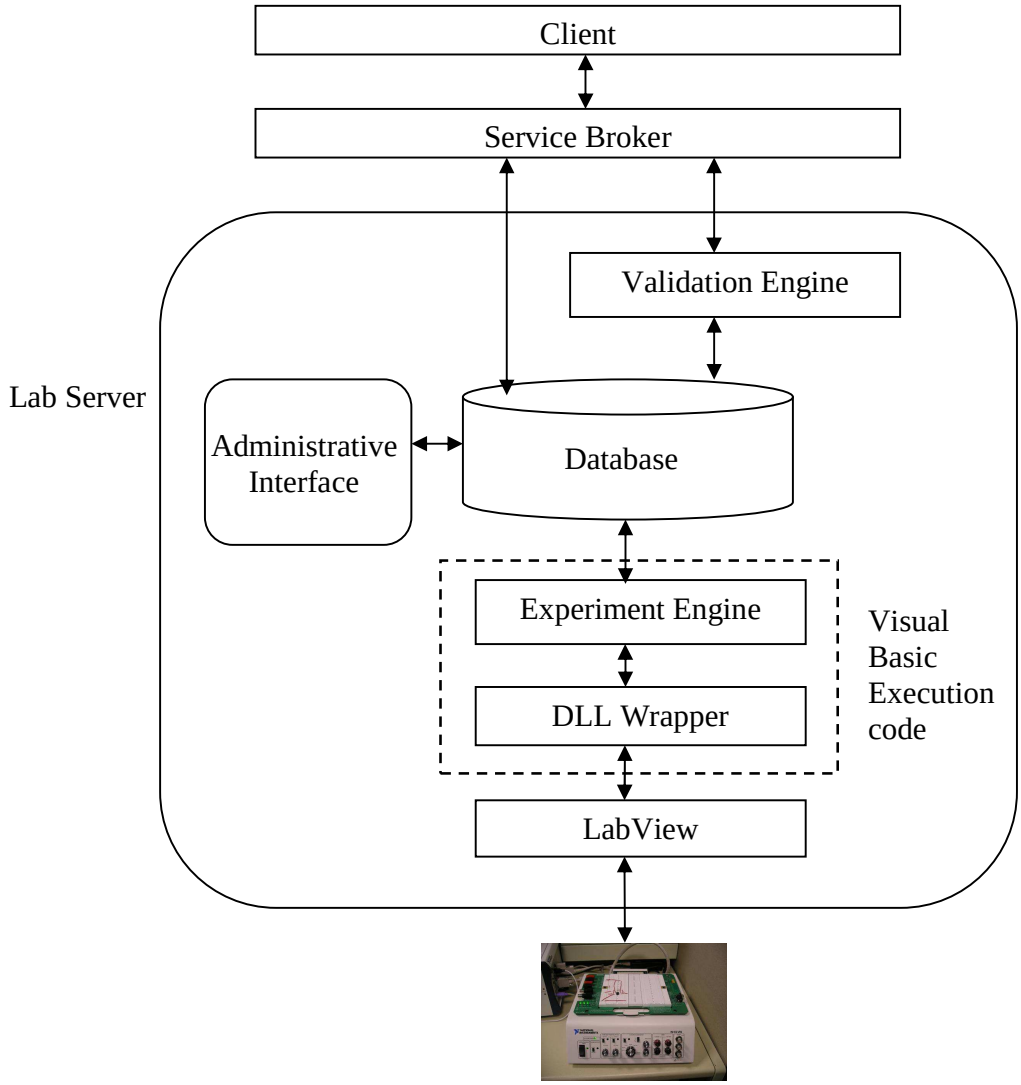


Figure 3-1: The lab server components consisting of the execution code, validation engine, database and administrative pages. It interacts with the client through the Service Broker.

Once the experiment has been executed on the hardware the results are passed back through the DLL wrapper to the experiment engine. The experiment engine packages the result in the *ExperimentResult* XML files and stores it in the database. The Service Broker waits for a notification from the experiment engine that the experiment is finished. Once it gets the notification it fetches the experiment results from the database and passes it to the client where the results are displayed to the student. Figure 3-8 elaborates in more detail the execution cycle of an experiment.

Next I will describe in detail each component of the lab server and the changes I made to them for developing the functionalities in the new ELVIS version.

3.1.1 Lab Server Labview

National Instruments provides the LabView software for accessing drivers that can be used to control the various hardware instruments on the ELVIS. LabView is a graphical programming language that can be used to control lab equipment including the ELVIS. LabView is similar to any programming language except that you use drag and drop graphical components and connect them using virtual wires to form a program. Like any language LabView has built in primitive data types that are also represented graphically. LabView also provides a rich library of Virtual Instruments (VIs) that can be used to form more complex programs. These VIs are similar to function or procedure calls in conventional programming languages. For example, there are VIs for arithmetic procedures like addition or multiplication; string procedures like reversing a string or splitting a string; array procedures like reversing an array or finding the size of an array.

The ELVIS hardware comes with drivers that can be used to run the different hardware instruments on the ELVIS platform including the ARB and BODE instruments. I also found that National Instruments provides VIs that give a higher level programming interface. These special VIs, called Express VIs, make programming the ELVIS simpler. For example there is an Express VI for the FGEN, ARB and BODE instruments. These Express VIs can be integrated with the LabView language to form a set of modules that can run all the required instruments on the ELVIS hardware. This code can be compiled into a shared library (DLL) that provides an interface for calling the LabView code from other programming languages such as C++ and Visual Basic. Hence in this way the LabView code can be called as if it were an independent program by other running programs.

The ELVIS iLab code is arranged in the following module structures that are described on the next page.

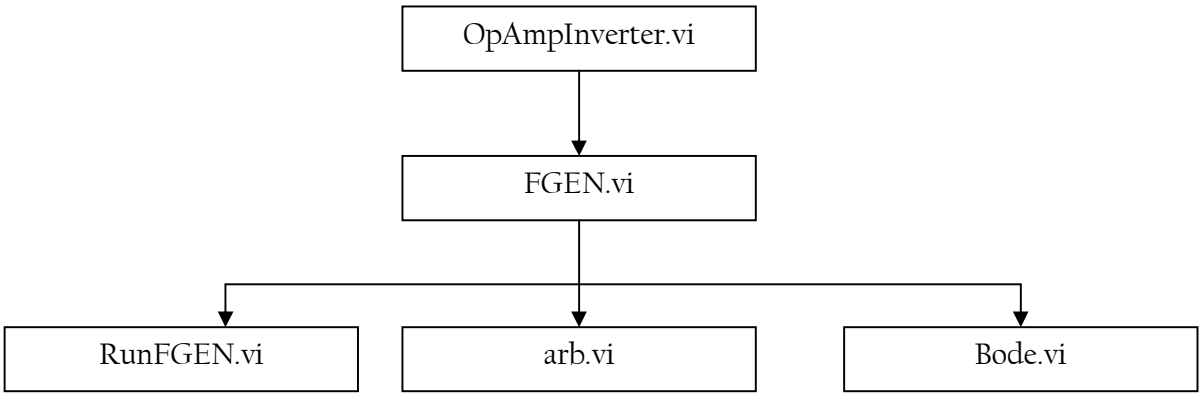


Figure 3-2: The LabView VI hierarchy.

OpAmpInverter.vi

This is the main entry point into the LabView code. This is the first class called by the compiled DLL from the Visual Basic Execution code in the lab server shown in Figure 3-1. This module just passes user parameters to the underlying FGEN.vi.

FGEN.vi

This is the VI that calls the various other hardware instruments. It calls the function generator and DAQ card (RunFGEN.vi), arbitrary waveform generator (arb.vi) and the bode analyzer (Bode.vi). Only one of the RunFGEN.vi and Bode.vi are run because the bode analyzer uses the function generator to output sine waves of different frequencies to measure the frequency response of the circuit.

RunFGEN.vi

Runs the function generator to produce the waveform requested by the user and controls the DAQ card to sample the required analog wave channels.

arb.vi

Runs the arbitrary waveform generator. This generates the required waveform on the DAC0 and DAC1 pins of the ELVIS. The waveform can be a predefined wave like a Square, Triangular, Sine, or Sawtooth wave. The feature also allows other flexible ways of defining a waveform either by a formula or by reading a file that has the data values for the waveform.

Bode.vi

Runs the Bode Analyzer feature of the ELVIS. This takes in the start and stop frequencies from the user and runs the function generator hardware to sweep the sine waves.

Both the Bode.vi and arb.vi files employ Express VIs that are provided in LabView to run the different ELVIS functionalities. For the arbitrary waveform generator (arb.vi) we wanted to add additional input capabilities to the ELVIS architecture. The ELVIS provides the DAC0 and DAC1 pins for generating two independent arbitrary waveforms. The capabilities supported for generating arbitrary waveforms are:

1) Square wave
2) Sine wave
3) Sawtooth wave
4) Triangular wave
5) Generating a wave from a formula
6) Generating a wave from a data file

For generating the Square, Sine, Sawtooth and Triangular waveforms LabView provides VIs that, given the wave parameters, calculate the waveforms and sample them according at the desired sampling rate. The parameters include: frequency, amplitude, phase and offset. An additional parameter for the square wave is the duty cycle. There are also parameters for the sampling rate and number of samples to be taken from the simulated wave. The variables representing these input parameters need to be created and connected to the appropriate waveform VI.

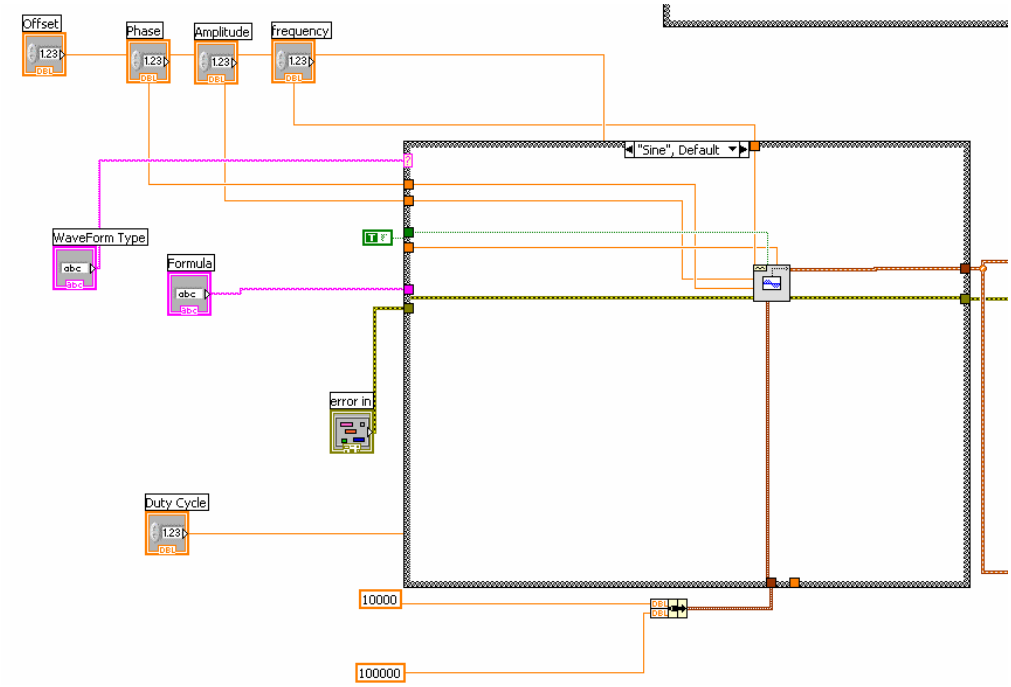


Figure 3-3: Part of the ARB VI showing the sine wave simulator and the input parameters it takes.

To generate the waveform from a formula there is also a VI that accepts a formula in MatLab syntax and parses the formula to generate the desired waveform.

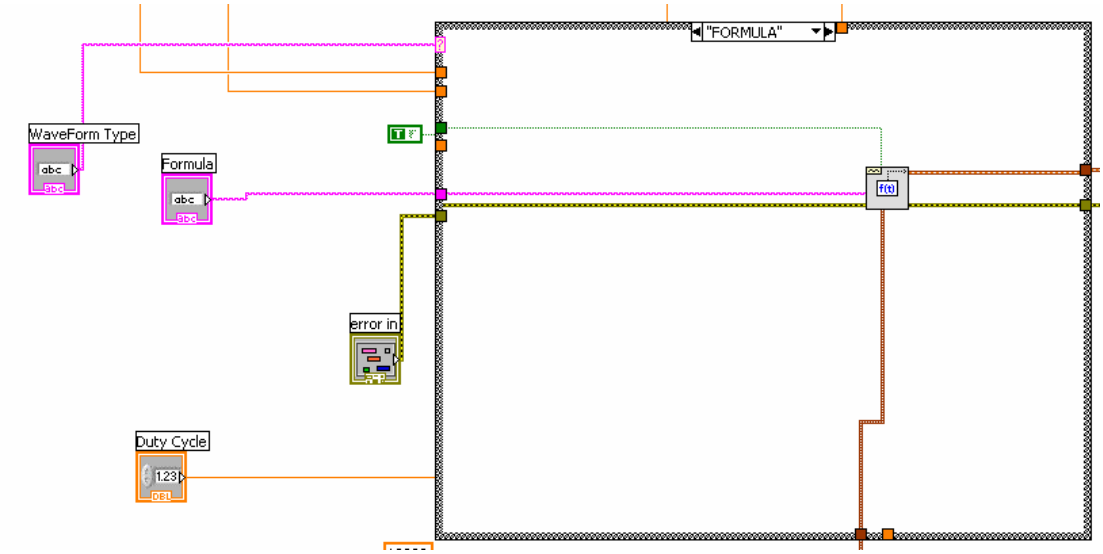


Figure 3-4: The formula generator VI.

To generate the waveform from a file, LabView provides the “Build Waveform” VI that takes an array of y values and a time difference (dt) and generates a wave from them (Figure 3-5). The array of y values is generated by parsing a comma delimited list of waveform values that is received from the user through the DLL wrapper class. This list is generated from a text file that the user loads in the client. All the above options for running the arbitrary waveform generator are put into a case statement that uses a string as a selector to determine which option to use.

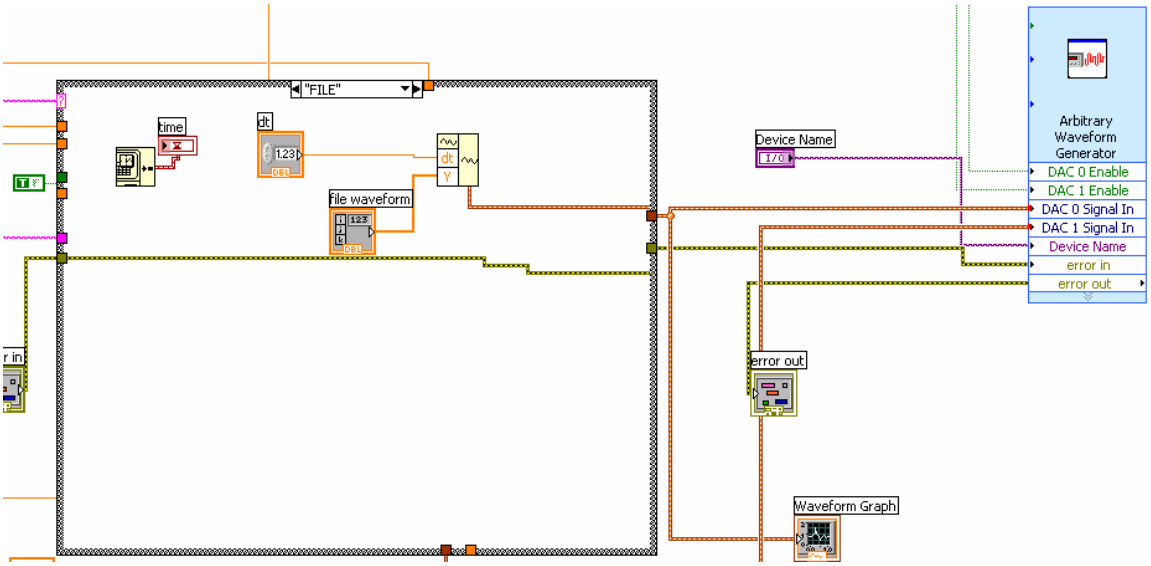


Figure 3-5: Generating a waveform from an array of y values.

.

For the Bode Analyzer VI (Bode.vi), the Express VI only requires three parameters: the start frequency, stop frequency and the steps per decade.

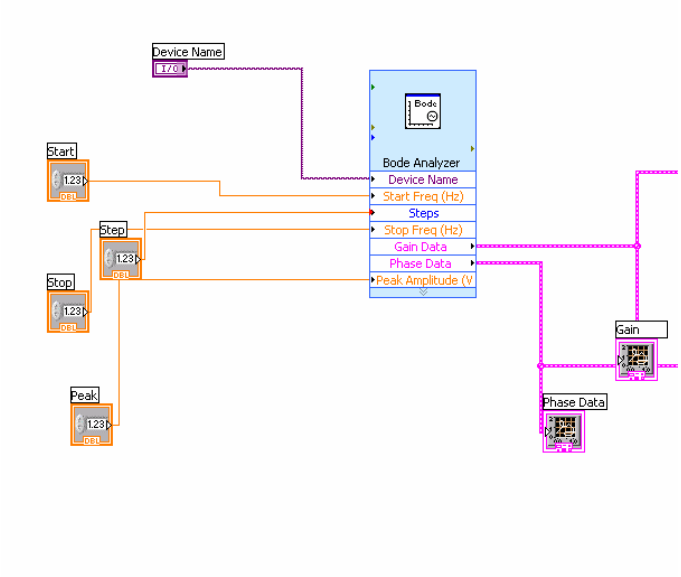


Figure 3-6: The bode analyzer VI developed using the Express VI to run frequency analysis in a certain defined frequency range.

I placed the ARB VI (arb.vi) in the FGEN.vi file to be run in a new thread. In this way the FGEN.vi file calls both the RUNFGEN.vi file to run the function generator and the arb.vi file to run the arbitrary waveform generator. This way the function generator and the arbitrary waveform generator can be run simultaneously. For the Bode Analyzer VI (Bode.vi) I placed the VI in a case statement so that only one of RUNFGEN.vi or Bode.vi is run as they both use the function generator hardware.

There are a limited number of parameters that can be exposed via the LabView DLL. This means that the Visual Basic Execution code in the lab server can only use a limited number of parameters to call the LabView DLL. For this reason I added a parser in LabView for each of the functionality. The parser parses a string to extract the different parameter values. This way for each instrument I only pass one string from the lab server which is then parsed in LabView to get the required parameters for running the instrument VI.

3.1.2 Experiment Engine

The experiment engine runs in the background at all times while the lab server is operational. At specified intervals, the experiment engine checks the experiment execution queue for new jobs (Figure 3-8). If one is available, the experiment engine de-queues the job. The main class file in the experiment engine is the Module1.vb file. This file has a M*ain()* subroutine that does the following:

1) De-queues submitted jobs from the SQL server. This is done at the following line:

   `strDBQuery = "SELECT dbo.qm_CheckQueue();"`

2) Once the job is de-queued the method call

   `ParseExperimentSpec(strExpSpec)`is made to parse the Experiment Specification received from the client.

   The ParseExperimentSpec method parses the experiment specification (an XML file that contains the experiment parameters chosen by the user). The method stores the values for different instrument parameters in a table called the `functInfoTable.` For example the code below extracts the frequency value specified for the function generator waveform and stores it in the table:

```
'load frequency value
 tempXPath = "/terminal/function/frequency"
 tempNode = xmlTemp.SelectSingleNode(tempXPath)
functInfoTable(instrumentConstant, FUNCT_FREQUENCY) =
```

Figure 3-7: Experiment Specification Parser.

3) This method calls the `RunExperiment()` method in the `Inverter` class defined in the *OpAmpInverter* project with the parameters values stored in the `functInfoTable` table. The *OpAmpInverter* project calls the DLL to run the experiment on the ELVIS hardware. The results are returned to the experiment

engine in an array of data for graphs that will be displayed to the user. The data points are then put into an XML file called the "Experiment Results" and sent back to the client for display to the user. Finally the experiment engine triggers a notification (via the *Notify()* method) to the ServiceBroker saying the data is ready [14].

I modified the experiment engine to handle the parsing of the parameters of the new instrument. This involved changing the XML parser in the ParseExperimentSpec method and extending the table that stores the parsed values. For the ARB feature, the two ARB channels are represented as distinct instruments so that they can be run independently. The new values are parsed from the Experiment Specification XML file and passed onto to the *OpAmpInverter* DLL wrapper.

The experiment engine also determines whether the experiment being run is a time domain or frequency domain experiment. This is done based on whether the FGEN or the BODE instrument is used. This is because only one of these instruments can be used at a time. This information is also used when sending results back to the client. If the experiment is a time domain experiment the x-axis values are time intervals otherwise they are frequency values of the sine waves used during the frequency sweep. The y-values are also different because they are FGEN or ARB waveforms in the time domain case and magnitude and phase values in the frequency domain case.

3.1.3 DLL wrapper (OpAmpInverter)

This is a Visual Basic wrapper class that imports the LabView DLL and calls it with the parameters passed from the experiment engine (Figure 3-8). The main class file

in this project is the *OpAmpInverter.vb* file that defines the `Inverter` class. The `RunExperiment()` method in the `Inverter` class calls the compiled LabView DLL with the specified parameters. The DLL runs the experiment on the ELVIS hardware. Once the experiment is run, it returns from the `Inverter` class back to the `runExperiment()` method in the experiment engine The DLL returns an interleaved array of the output data back from the LabView code. This array is deinterleaved in the `RunExperiment()` method of the Inverter class, which returns this data back to the `runExperiment()` method of the experiment engine.

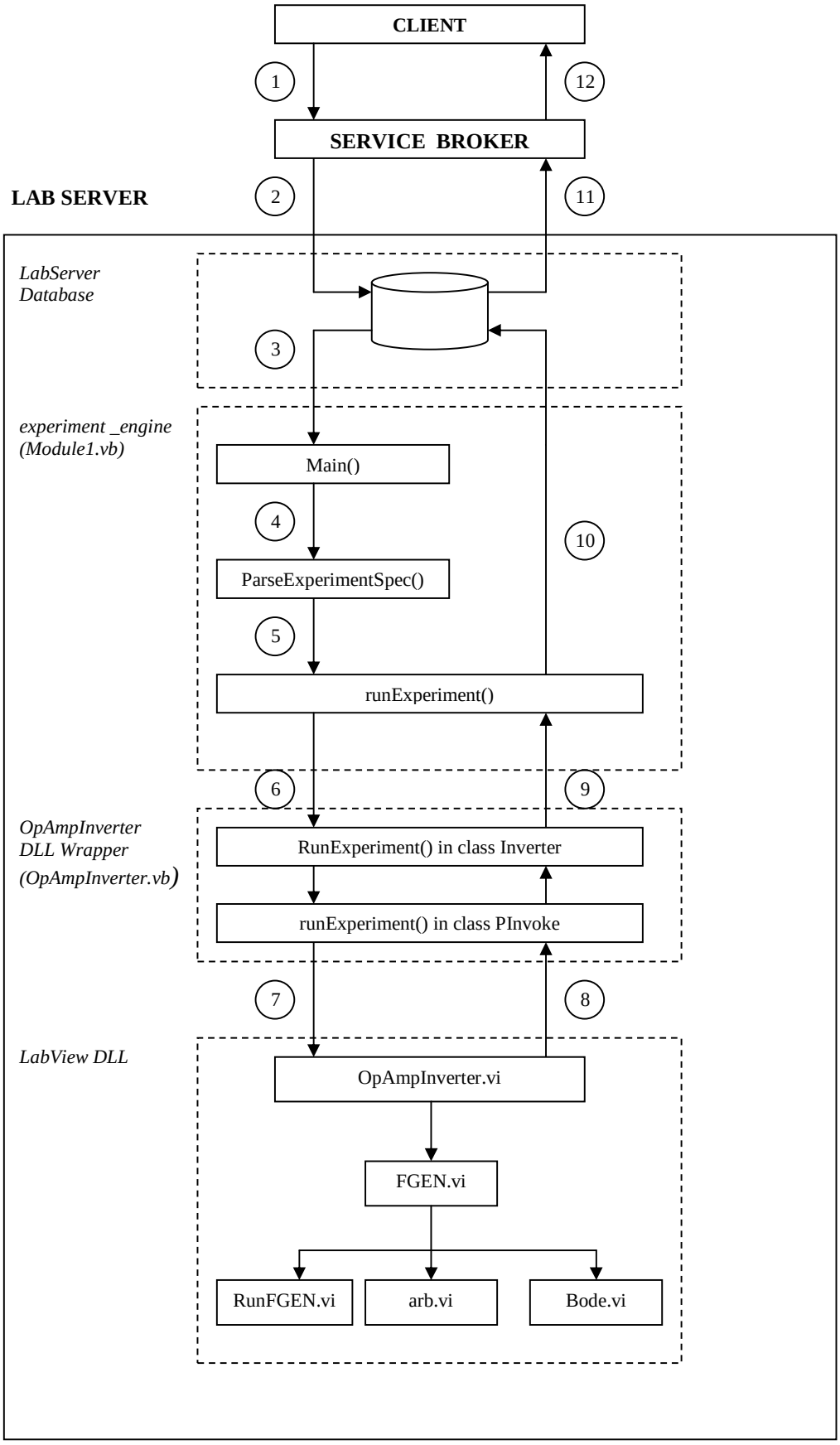
In the DLL wrapper I condensed all the parameters for each instrument into one string. Each ARB channel has 8 parameters and the BODE has 3 parameters that need to be passed to the LabView DLL. However the LabView DLL can only take a maximum of 28 parameters. Passing the parameters separately would have left only 5 more parameters open for future use. By passing a string for each instrument the number of parameters required is only 3 instead of 19; saving 16 parameters for future use.

I also added an extra sampling channel in LabView that returns the output waveform generated at the ARB channel. LabView returns an interleaved array of all the sampled waveforms. Hence the code in the DLL wrapper has to be changed to de-interleave the extra waveform.

3.1.4 Validation engine

This module is called before the job is queued for execution. It checks whether the inputs specified by the user meets the specification set by the designer of the experiment when setting up the assignment. It works the same way as the

`ParseExperimentSpec()` method in the experiment engine to extract the experiment parameters and checks these values against values stored in the database. Hence the changes made to the validation engine are similar to the changes made to the parser in the experiment engine. I also added constraint checks for the ARB and BODE instruments. Some examples of the ARB constraints include maximum voltage or maximum frequency. For the BODE I checked to make sure that the start and stop frequencies were between 1 Hz and 35 KHz, which are the extremes allowed by the ELVIS hardware.



1) The client sends the "execute" SOAP request to the ServiceBroker with the "Experiment Specification" XML file.
2) A Web service call is made to the appropriate Lab Server and the experiment specification is stored in the Lab Server Database after validation.
3) The experiment engine in its Main() method de-queues the experiment in a FIFO manner and fetches the stored experiment specification.
4) The experiment specification is passed on to the ParseExperimentSpec() method in the execution engine where the XML file is parsed and parameters stored in a table.
5) The runExperiment() method is called. It extracts the parameters from the table stored during parsing.
6) The RunExperiment() method in the Inverter class is called using the parameters extracted this in turn calls the runExperiment() method in the PInvoke class with the same parameters.
7) The PInvoke class imports the compiled LabView DLL and runs the DLL with the parameters when called. The entry point into the LabView code is the OpAmpInverter VI. This runs the experiment on the ELVIS platform.
8) The LabView DLL finishes execution and returns and array of output data to the Inverter class.
9) The Inverter class de-interleaves the output array and returns a 2D array of results to the experiment engine.
10) The experiment engine forms the "Exp Results" XML file and stores it in the database and sends a notification to the Service Broker.
11) & 12) The Service Broker fetches the Exp Result file from DB and forwards it to the client.

Figure 3-8: The execution cycle showing details of the lab server.

3.1.5 Lab Server Administrative Interface and Database

The lab server administration interface is an ASP website where you can create new experiment setups. It interacts directly with the lab server SQL database. To reflect the new ARB and BODE functionalities the business logic of the ASP website was changed to reflect the new instruments and their constraints. The database was also changed to store the new instruments and constraints.

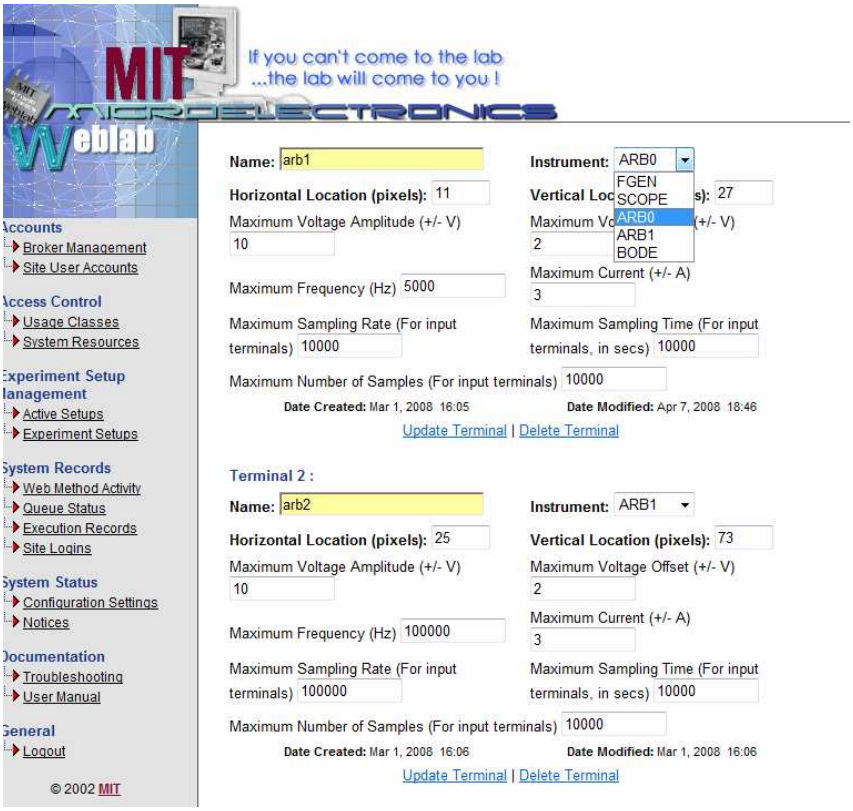


Figure 3--9: The lab server administration interface where the creator of an experiment can choose what terminals and instruments to use in the experiment and specify constraints for those instruments.

When the instructor creates a new experiment they can add the ARB0 or ARB1 instruments (ARB for the DAC0 and DAC1 channel respectively) or the BODE

instrument. For the ARB they can specify constraint parameters like maximum voltage, maximum frequency or maximum offset to prevent damage to the circuit components.

When the instrument is added to the experiment the `AddSetupTerminal` stored procedure is called that adds the new terminal to the `SetupTerminalConfig` table in the lab server database. The stored procedure is changed to pass the constraints of the ARB and BODE instruments and the table was changed to include columns for the new constraints. The entries in the `SetupTerminalConfig` are used to create the Lab Configuration document in the lab server when the client is started.

3.2 Client

The client is where the user can specify the parameters that will be used in the experiment. It is a Java Applet that is launched from the Service Broker. It uses Simple Object Access Protocol (SOAP) requests to interact with the Service Broker to send and receive information. To understand the changes involved in the client it is important to look at the steps involved in the running the client.

3.2.1 The main flow of information in the client

(italics represent Java class names)

When opened, the client is initiated through the *GraphicalApplet* class and a *SBServer* class is instantiated in the applet class. This *SBServer* class represents the Service Broker the applet was downloaded from. The init() method in the *GraphicalApplet* class then creates a new *WebLabClient*. When this happens the

loadLabConfiguration() method in *WebLabClient* is called. This method fetches the Lab Configuration XML File (see Section 2.1.4) from the lab server through the specified Service Broker via the getLabConfiguration() SOAP call in the *SBServer* class [15].

The Lab Configuration is then parsed by the parseXMLLabConfiguration() method in the *LabConfiguration* class. From parsing the Lab Configuration a list of *Terminals* are created. A *Terminal* has an instrumentType (for example *Instrument*.FGEN_TYPE or *Instrument*.SCOPE_TYPE) and an instrumentNumber, a label, an xPixelLocation and yPixelLocation to identify where the terminal is located in the setup image.

From the list of Terminals a *Setup* is created which represents the current experiment. Setup has a setupID, a name, a description, an imageURL, and an ordered list of Terminals that are present in the experiment.

Once the Lab Configuration has been parsed, the *Setup* is stored in the *ExperimentSpecification* `theSetup` field and the *Instruments* (*FGEN*, *ARB*, *SCOPE*, *BODE*) are created from the *Terminal* information and stored in the `instruments` vector in the same class. Then the *MainFrame* draws the main client elements including the buttons and the menu bars. The *MainFrame* then calls the *SchematicPanel* and the *ResultsPanel* which draws the axes for plotting later.

The *SchematicPanel* uses setup stored in the `theSetup` in the *ExperimentSpecification* to draw the image of the experiment and the corresponding *InstrumentLabel* for the instruments in the setups.

The experiment is ready to be run. When the user clicks on any of the instrument labels the instrument dialog box appears. Each instrument has its own dialog box that

users can use to specify parameters of the instruments. For example frequency, amplitude etc for the *FGEN* instrument.

Each *Instrument* has a *SourceFunction* associated with it. This *SourceFunction* (*WAVEFORMFunction* in the case of *FGEN)* is changed when the user specifies values in the dialog box (*FGENDialog* for the case of *FGEN*).

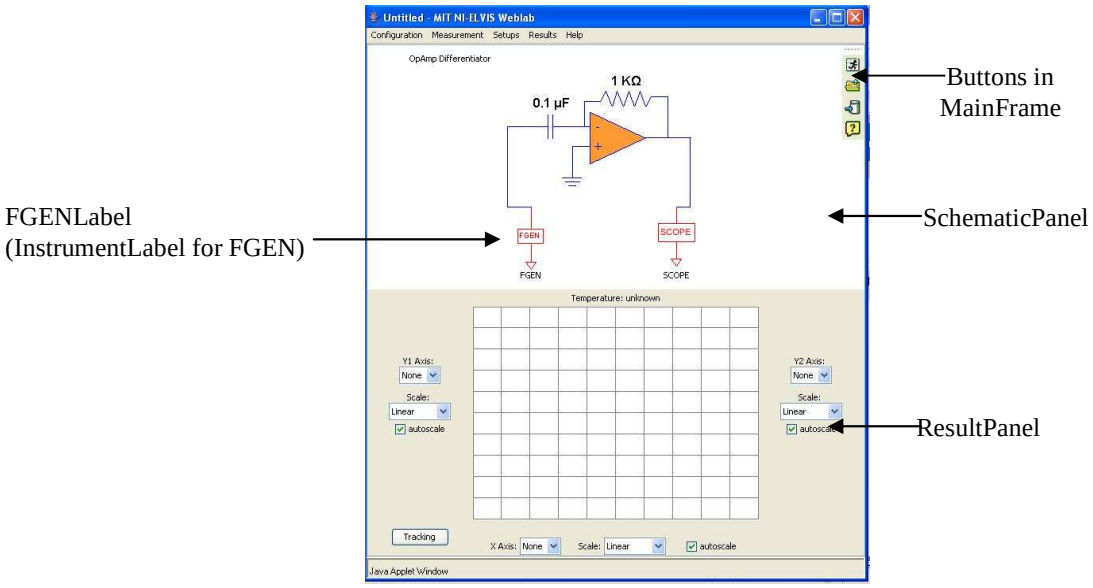


Figure 3-10: Different parts of the Java Client

When the user clicks on the 'Run' button, the Experiment Specification XML document is created by the ExperimentSpecification class. This Experiment Specification is sent to the lab server via the execute SOAP call in the *SBServer* class. The job is now submitted to the lab server and the events in Figure 3-8 take place. When the experiment finishes executing the lab server sends a notification to the client. The RetrieveResult SOAP request is then used to get the Experiment Result XML file from the lab server. The parseXMLExperimentResult method in *ExperimentResult* is used to parse the

Experiment Result XML file. The data is displayed using the *Axis*, *ConnectPattern*, *Graph* and *Grid* classes in the graphing package.

3.2.2 Changes made to the client

The first step in adding the new functionalities was creating an instrument class for each feature. For the Arbitrary Waveform generator (ARB) and Bode Analyser (BODE) feature a new class was created that extends the Instrument class.

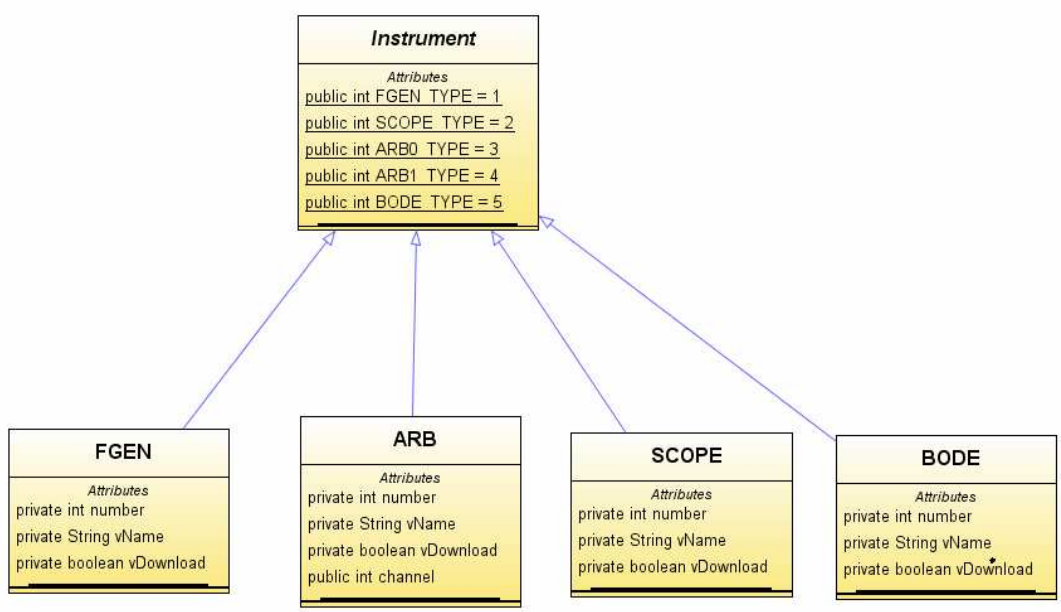


Figure 3-11: Each feature on the ELVIS is an Instrument. All features inherit from the Instruments

Then I made the source function class for the new instruments by extending the *SourceFunction* class. The source function (Figure 3-12) for the instrument stores the information parameters for your instrument. In my case the ARBFunction is the function for the ARB instrument and stores information like the waveform type selected by the user and the parameters like frequency, amplitude, phase etc associated with the

waveform. If the user wants to plot a formula it also stores the wave function. Finally, to use a file for generating a wave, there is a parser that parses the file loaded and extracts the dt value and the associated y values that are stored in the `wavefilevalue` field. The BODEFunction similarly has the fields to store the start, stop frequencies and the steps per decade to be used for the bode analysis.

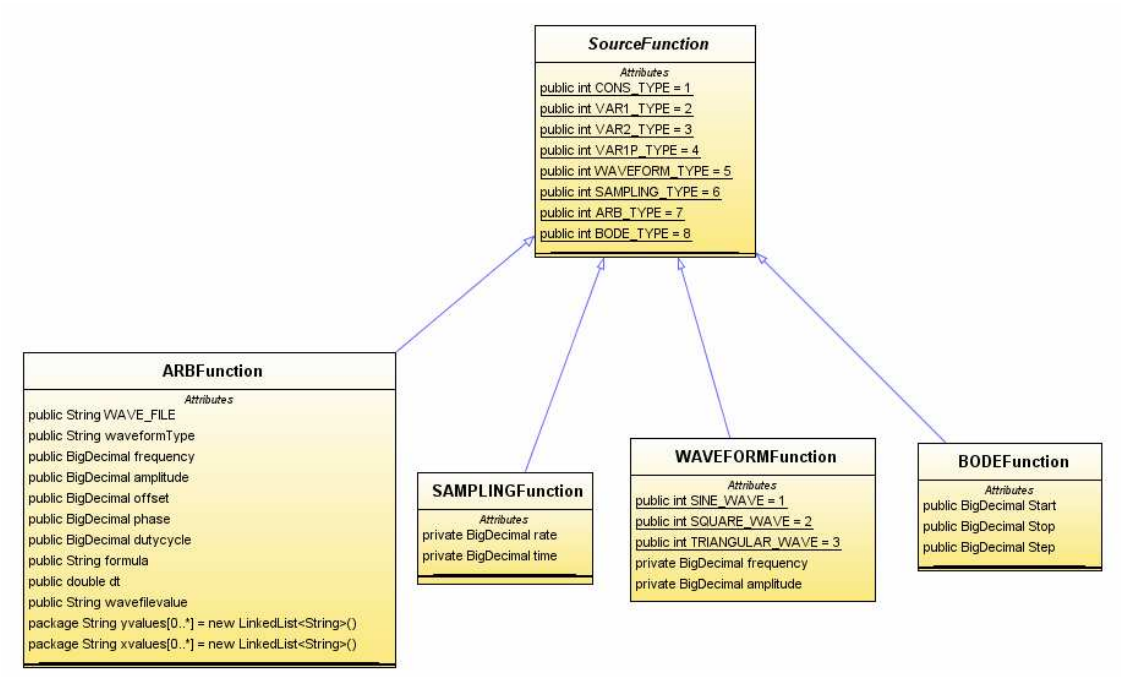


Figure 3-12: Associated with each instrument is a function. Functions for each instrument are sub classes of the SourceFunction class.

Next I designed the user interface dialog boxes that will allow the user to input the desired parameters for the ARB and BODE instrument. For the BODE instrument the user needs to specify the start and stop frequency of the sweep and the increment steps per decade for the sweep.

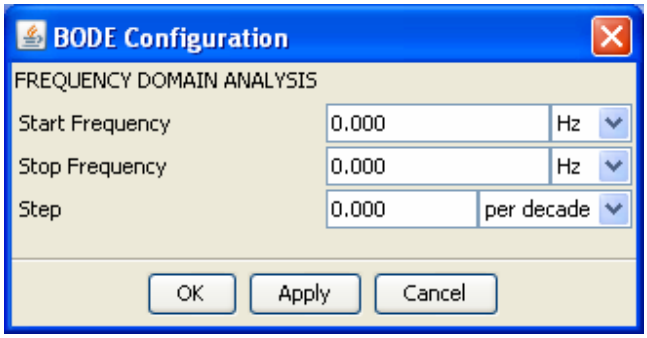


Figure 3-13: The dialog box to specify parameters for the bode analysis.

For the ARB instrument, the user can either select from predefined waveforms, choose to enter a formula or load a wave data file.

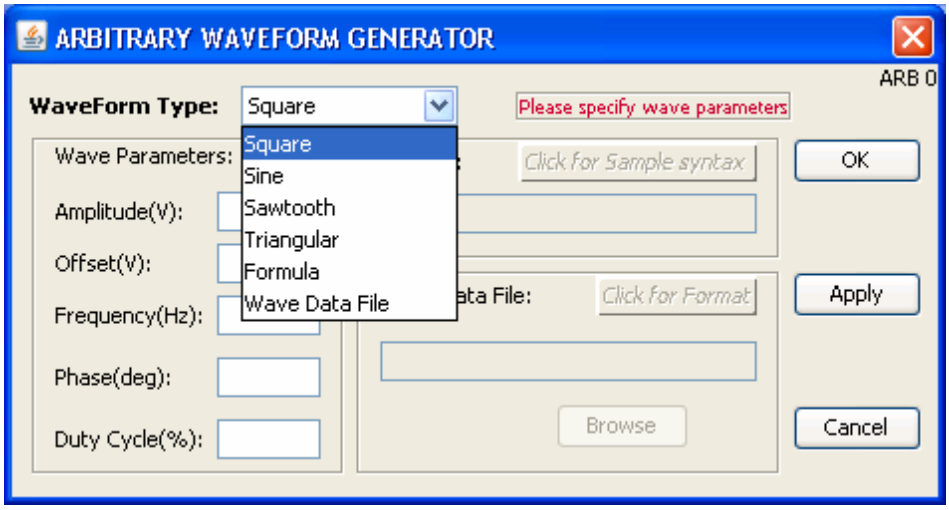


Figure 3-14: In the ARB dialog box users can choose the type of waveform they would like to use and specify parameters for them. They can also enter a formula or load a file to specify the wave form.

When the user chooses from predefined waveforms, they can specify the parameters that they would like to use for that waveform. Otherwise they can enter a formula or load a comma delimited file to specify the waveform. The formula accepted by the client has to be in MatLab syntax. This formula is passed from the lab server to the

LabView DLL, where it is parsed to form the wave. A wave file can be created using MS Excel and saving the file as *CSV* file.

To handle the drawing of the new experiment instruments, the *LabConfiguration* class needs to be changed to handle the parsing of the new instruments. Hand in hand with this, new labels with appropriate images for each instrument need to be made. To pass the user parameters for the ARB and BODE instruments to the lab server the Experiment Specification XML needs to be changed. Additional tags are added to the XML document for the new instruments and the same XML tags are used to parse the specification in the lab server.

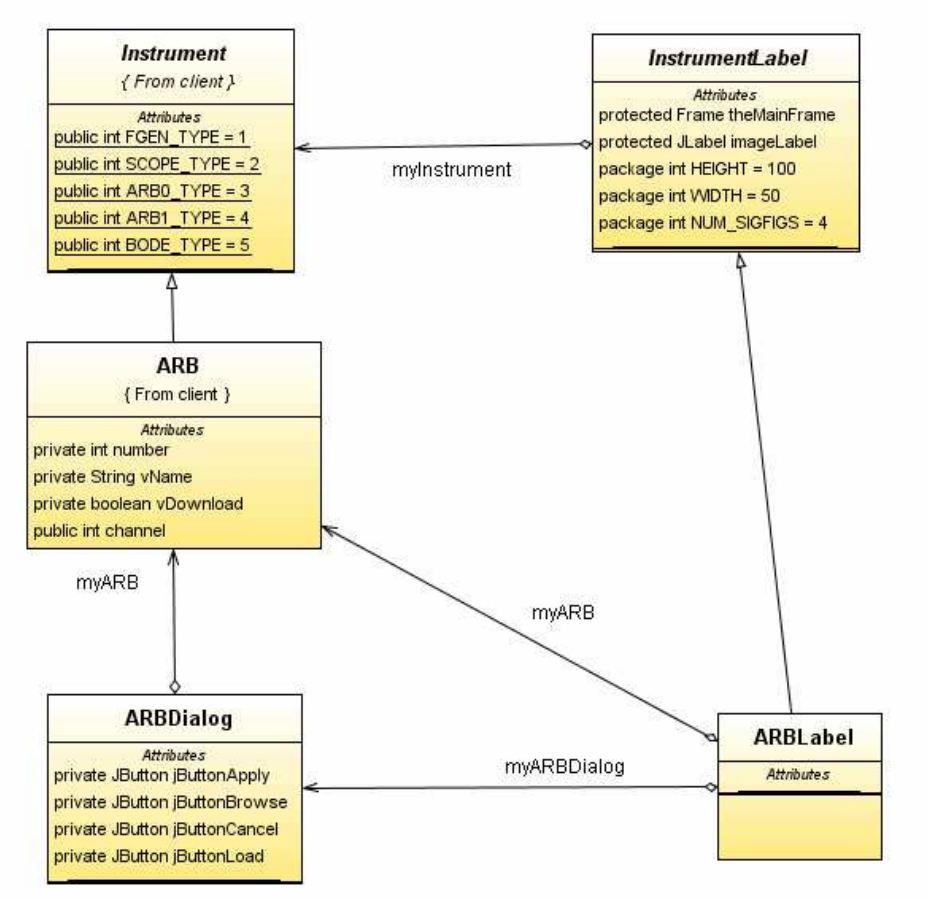


Figure 3-15 The UML diagram shows how the ARB instrument is related to the ARBDialog class that draws the dialog box and the ARBLabel class.

3.3 Testing using new circuits

To test the two new ARB channels, I decided to build an adder circuit. This was not possible in the ELVIS version 1.0 because it only had one input channel. I used a sine wave and square wave of the same frequencies in each of the input channels. I also tested the ARB feature with a wave file that forms a wave that is unique. To test the BODE feature I built three Sallen-Key filters to test the BODE feature.

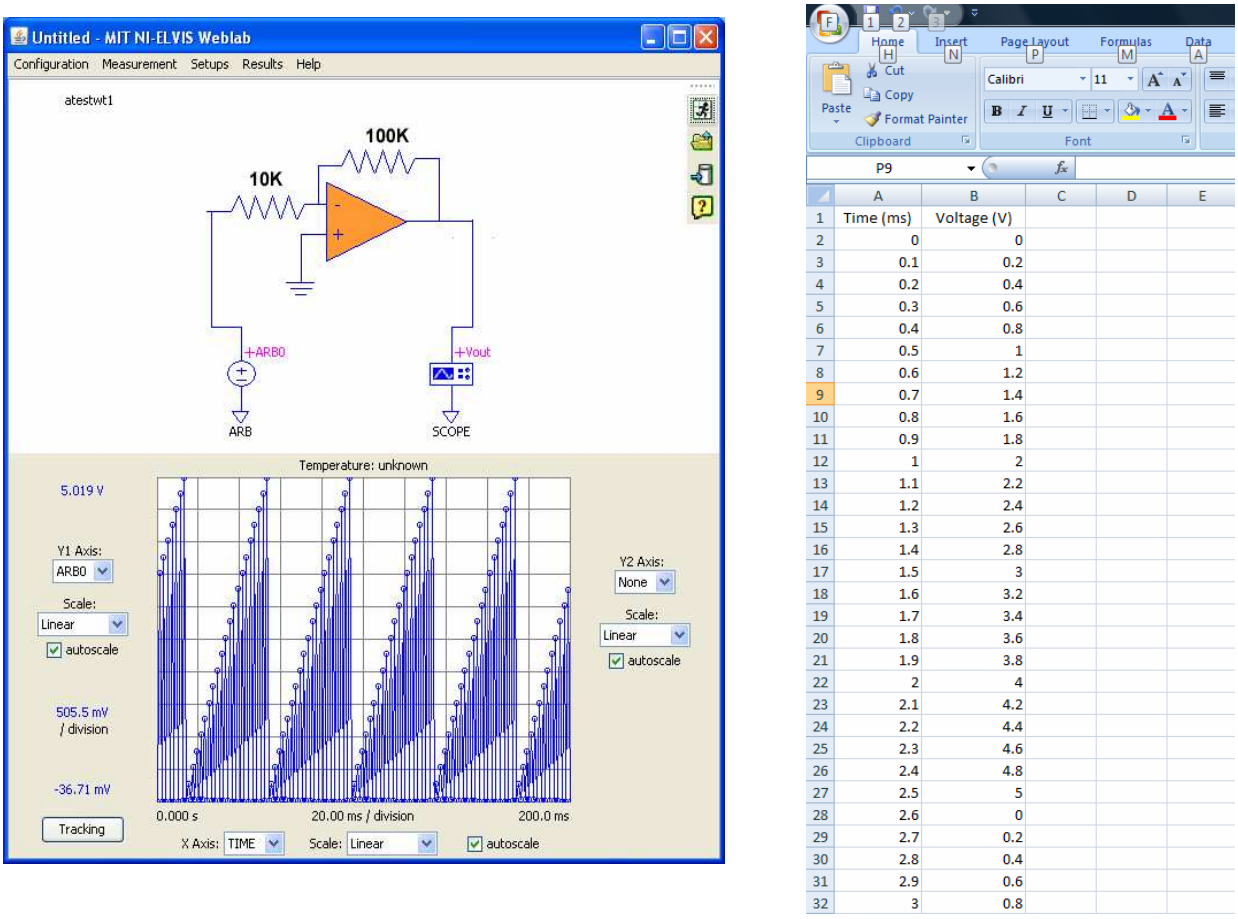
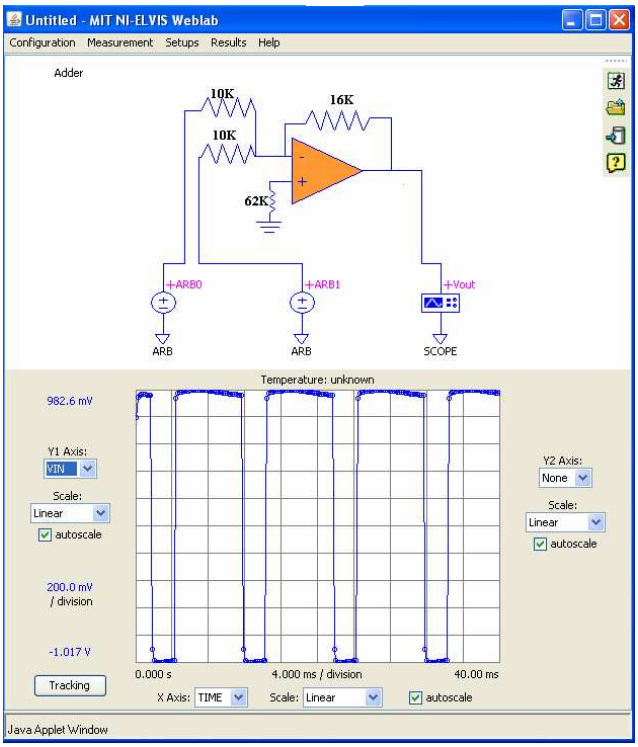
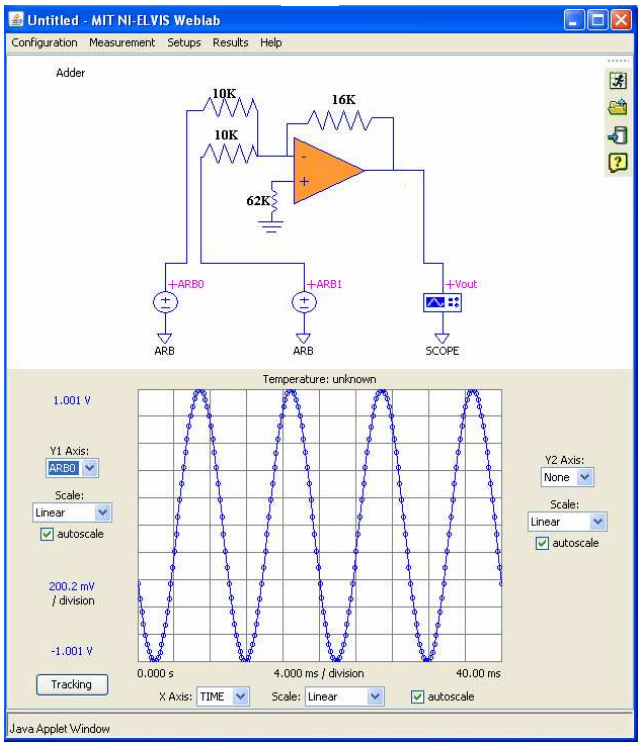


Figure 3-16: This shows the waveform generated by loading a .csv file that was created in MS Excel. It generates a unique waveform that won't be found in any pre-defined library and for which a formula doesn't exist.

A



B



C

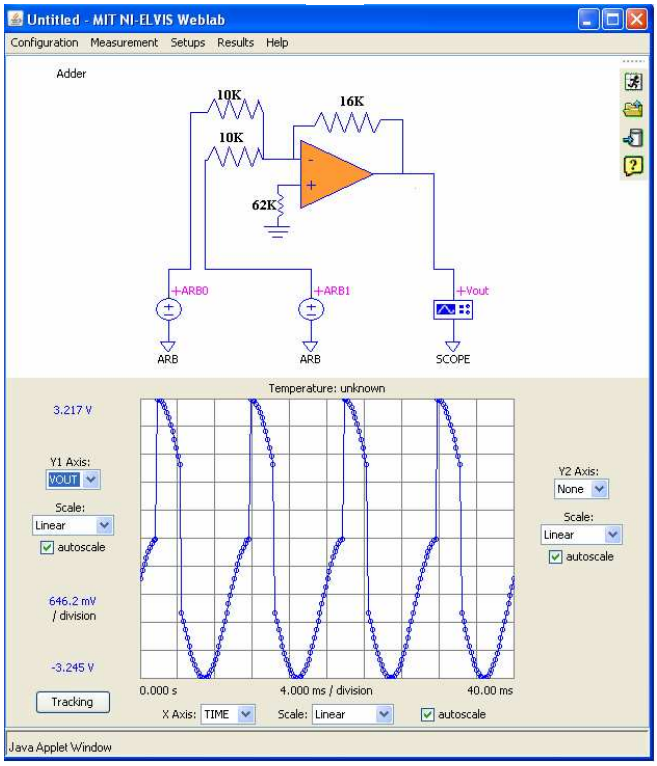
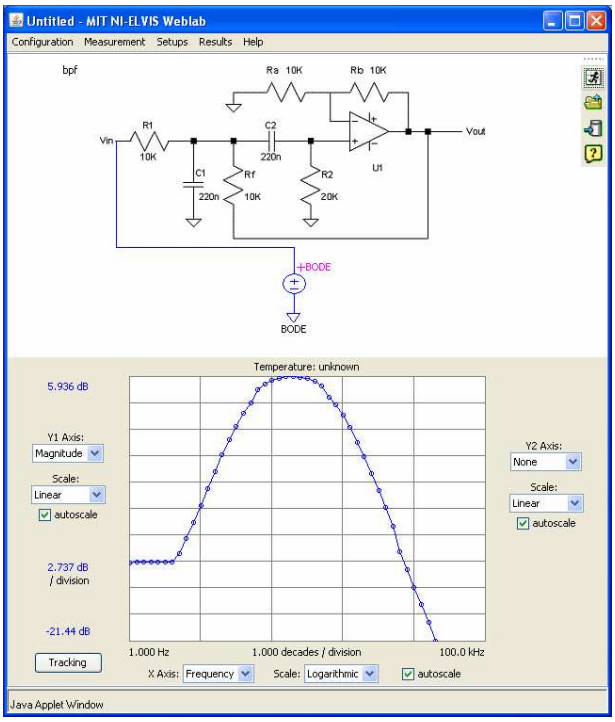
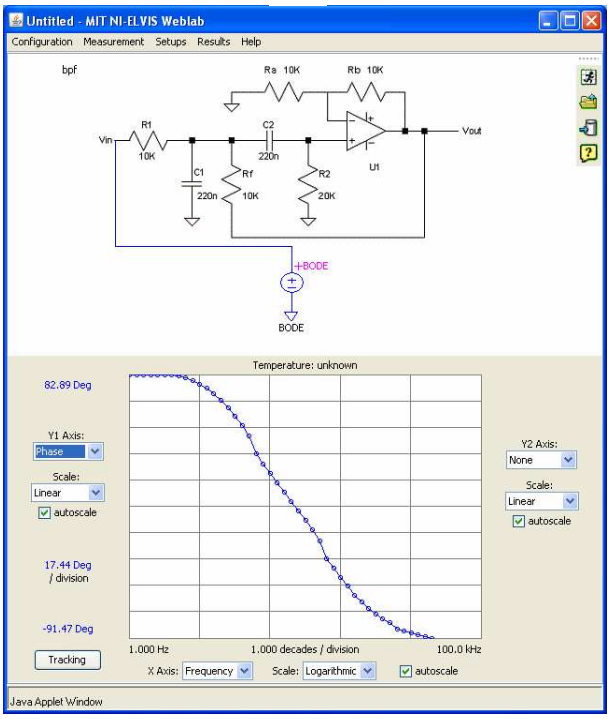


Figure 3-17: Using the ARB feature to run an experiment on an adder circuit. A and B shows the two input waveforms and C shows the output of the adder circuit. Such a circuit with two inputs wasn't possible in ELVIS version 1.0.

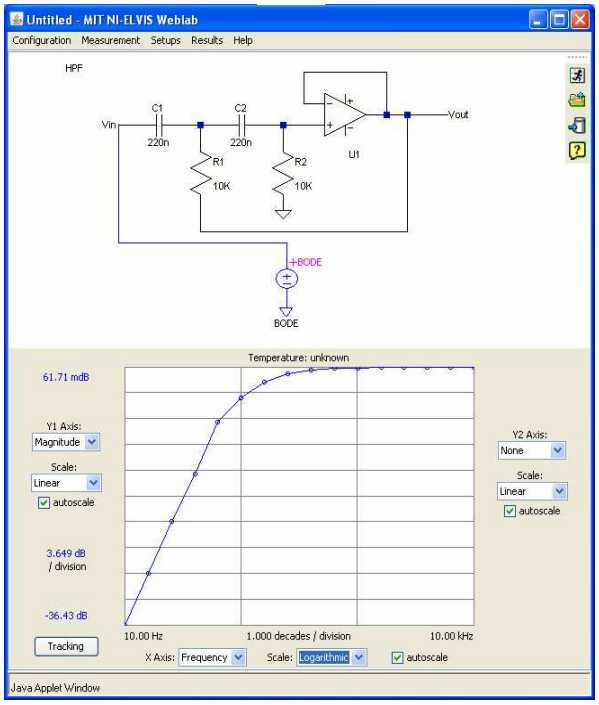
A



B



C



D

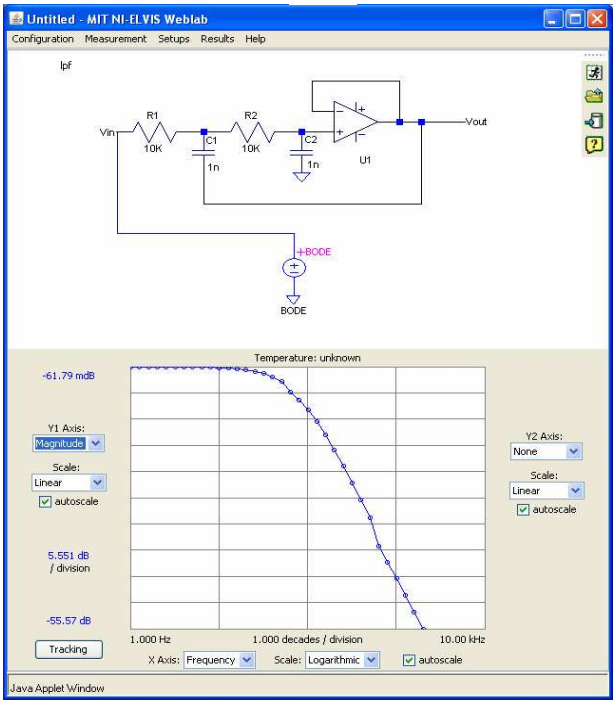


Figure 3-18: The various bode analyses done using the bode analyzer feature. A and B show the magnitude and phase plot of a Sallen-Key band pass filter. C and D show the magnitude response of a high pass and low pass filter respectively.

CHAPTER 4

Conclusions and Recommendations for Future Work

The work described in this thesis has improved the current ELVIS iLab version. However there are still other improvements that can make the ELVIS iLab a more versatile electrical engineering remote laboratory platform.

4.1 Conclusion

The improvements made to the ELVIS platform greatly increase the types of experiments that can be done on this platform. The arbitrary waveform generator enables the user to design circuit with up to three input channels. It also allows the designer of the experiment and the student to use a variety of waveforms and even to make their own waveforms using either a formula or a file with the wave values.

The bode analyzer feature will allow students to do frequency domain analysis of circuits. The ability to combine time domain and frequency domain analysis of the same circuit will allow students to getter better insight on what the circuit is doing.

4.2 Recommendation of future work

One of the main problems that I faced with the arbitrary waveform generator feature was the triggering of the oscilloscope to capture an arbitrary waveform. This is because the triggering was hard coded and was not a user defined option. The setting that we had encoded in the first version of the ELVIS required triggering on the rising edge

when the 0 V level was passed. However, this setting doesn't always work for the arbitrary waveform generator because the user can input a waveform that never crosses the 0 V threshold level. Such a waveform can be always less than zero or always greater than zero. Even if it does cross the zero, it may be much later than the start of the waveform meaning that the oscilloscope will not capture anything before the zero crossing.

This problem has recently been fixed by Jim Hardison who used the higher level Express VI to offer the user a choice of triggering modes. Hence they can choose the appropriate triggering type based on the waveform they are studying. Combining this with my version will greatly improve the arbitrary waveform generator feature.

Bryant Harrison was working on his thesis to develop ELVIS version 2.0 which added the capability of switching at the same time I was doing my research. Consolidating his version with mine will be a logical next step in ELVIS development. Doing this will enable students to be more creative by enabling experiments that require circuit design skills. For example one possibility is having students design a feedback controller for a system. Using ELVIS version 2.0 and 3.0, students can choose different components for their controller and check the frequency response of the system with their newly designed controller.

There still remain quite a few features on the ELVIS that need to be adapted for the iLab platform. Some of the features that should be exposed in future versions include the digital and device characterization features. NI ELVIS also just recently introduced a second version of the ELVIS platform (ELVIS II). The current ELVIS platform (ELVIS I) has high noise issues when using low voltages on the order of 50mV. When generating

a sine waveform of such amplitude in the function generator, there usually is an offset that causes the wave not to pass the zero crossing. It would be interesting to test the performance of the new hardware. The function generator of the new ELVIS platform can generate waveforms of higher frequencies than the current platform. The ELVIS I that can generate a maximum frequency of 250KHz while the ELVIS II can generate frequencies of up to 5MHz. The bode analyzer in the new ELVIS also has a bigger frequency range of measurement [16].

4.3 Progress at UDSM and MUK

A major component of the thesis work was collaborating with iLab teams at the University of Dar-es-Salaam and Makerere University of Kampala. We visited UDSM and MUK in January 2008 to help develop the iLab teams at those institutions and to integrate ELVIS experiments in the curriculum. At UDSM one of our main aims was to spread iLabs to other institutions in the country. We held a seminar attended by more than 50 participants from 6 institutions within the country. The seminar raised awareness about iLabs and the attendees showed great interest in the technology. During a second visit in June 2008, we visited two of the institutions that showed the most potential and encouraged them to start getting involved by using iLab experiment in their curriculum. The UDSM iLabs team is currently working on improving the lab server and adding the bode analyzer feature to it. The team also bought 2 more ELVIS platforms, and 5 students will start developing new experiments on this equipment.

Figure 4-1: Picture taken at the iLabs workshop at UDSM on 22nd January 2008.

At MUK we started training the new student team and helping them get started on their final year project. They learned LabView and ELVIS programming very quickly. When they were able to visit MIT in April, they completed most of the work for their final projects and started exploring adding the digital features of the ELVIS. During the summer an iLabs conference and workshop was held at MUK for all the partner universities.

More work remains to be done in training the teams there so that they can contribute to the ELVIS iLab platform by developing additional functionalities. We hope this joint development will then be shared and will encourage collaboration among the partner universities.

APPENDIX A

LabConfiguration.xml


```
<?xml version='1.0' encoding='utf-8' standalone='no'
?><!DOCTYPE labConfiguration SYSTEM
'http://web.mit.edu/weblab/xml/labConfiguration.dtd'><labCon
figuration lab='MIT ELVIS Weblab' specversion='0.1'><setup
id='7'><name>Adder</name><description>Op amp Adder
</description><imageURL>http://localhost/LabServer/images/se
tups/7adder.GIF</imageURL><terminal instrumentType='ARB0'
instrumentNumber='1'><label>arb1</label><pixelLocation><x>11
</x><y>27</y></pixelLocation></terminal><terminal
instrumentType='ARB1'
instrumentNumber='2'><label>arb2</label><pixelLocation><x>25
</x><y>73</y></pixelLocation></terminal><terminal
instrumentType='SCOPE'
instrumentNumber='3'><label>scope</label><pixelLocation><x>2
45</x><y>86</y></pixelLocation></terminal></setup><setup
id='13'><name>band pass filter</name><description>b
</description><imageURL>http://localhost/LabServer/images/se
tups/19bo5.GIF</imageURL><terminal instrumentType='BODE'
instrumentNumber='1'><label>bode</label><pixelLocation><x>17
</x><y>83</y></pixelLocation></terminal></setup></labConfigu
ration>
```

APPENDIX B

ExperimentSpecification.xml

A. BODE

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE experimentSpecification SYSTEM
"http://localhost/LabServer/xml/experimentSpecification.dtd"
><experimentSpecification lab="MIT NI-ELVIS Weblab"
specversion="0.1"><setupID>13</setupID><terminal
instrumentType="BODE" instrumentNumber="1"><vname
download="true">BODE</vname><function
type="BODE"><startfrequency>10.00</startfrequency><stopfrequ
ency>1000</stopfrequency><step>5.000</step></function></term
inal></experimentSpecification>
```

B. ARB

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE experimentSpecification SYSTEM
"http://localhost/LabServer/xml/experimentSpecification.dtd"
><experimentSpecification lab="MIT NI-ELVIS Weblab"
specversion="0.1"><setupID>7</setupID><terminal
```

instrumentType="**ARB0**" instrumentNumber="1"><vname download="true">ARB0</vname><function type="ARB"><waveformType>Square</waveformType><frequency>250.0</frequency><amplitude>1.0</amplitude><phase>0.0</phase><offset>0.0</offset><dutycycle>50.0</dutycycle><formula></formula><channel>BOTH</channel><dt>0.0</dt><filevalues>null</filevalues></function></terminal><terminal instrumentType="**ARB1**" instrumentNumber="2"><vname download="true">ARB1</vname><function type="ARB"><waveformType>Sine</waveformType><frequency>400.0</frequency><amplitude>1.0</amplitude><phase>0.0</phase><offset>0.0</offset><dutycycle>0</dutycycle><formula></formula><channel>BOTH</channel><dt>0.0</dt><filevalues>null</filevalues></function></terminal><terminal instrumentType="SCOPE" instrumentNumber="3"><vname download="true">Vout</vname><function type="SAMPLING"><samplingRate>20000</samplingRate><samplingTime>0.01000</samplingTime></function></terminal></experimentSpecification>

APPENDIX C

ExperimentResult.xml

A. BODE


```
<?xml version='1.0' encoding='utf-8' standalone='no'
?><!DOCTYPE experimentResult SYSTEM 'http://ilab-
labview.mit.edu/labserver/xml/experimentResult.dtd'><experim
entResult lab='MIT NI-ELVIS Weblab'
specversion='0.1'><datavector name='Frequency' units='Hz'>10
15.8489319246111 25.1188643150958 39.8107170553497
63.0957344480193 100 158.489319246111 251.188643150958
398.107170553497 630.957344480193 1000
</datavector><datavector name='Magnitude' units='dB'>-
8.43744638272253 -4.72643766642672 -1.13100822846422
1.87497449923435 4.56120907760289 5.51952781285724
5.92812725499974 6.10969595899165 5.50990339189745
4.14724738839594 3.72754745030727 </datavector><datavector
name='Phase' units='Deg'>73.2895958778259 69.4963263262885
63.6334972817331 51.0847210224226 31.1982910592505
16.9370339215086 5.60293878972823 -7.26607176216796 -
21.7011374320994 -38.7595993504763 -59.4737248206846
</datavector></experimentResult>
```

B. ARB


```
<?xml version='1.0' encoding='utf-8' standalone='no'
?><!DOCTYPE experimentResult SYSTEM 'http://ilab-
labview.mit.edu/labserver/xml/experimentResult.dtd'><experim
entResult lab='MIT NI-ELVIS Weblab'
specversion='0.1'><datavector name='TIME' units='s'>0 5E-05
0.0001 0.00015 0.0002 0.00025 0.0003 0.00035 0.0004 0.00045
0.0005 0.00055 0.0006 0.00065 0.0007 0.00075 0.0008 0.00085
0.0009 0.00095 0.001 0.00105 0.0011 0.00115 0.0012 0.00125
</datavector><datavector name='VIN'
units='V'>0.635299299652402 0.999653454555948
1.00029775904035 0.999653454555948 1.00094206352501
0.999975606798119 1.00094206352501 0.999331302313842
1.00158636800992 0.999009150071798 0.999975606798119
2.99278710075719 -0.486756969986187 </datavector><datavector
name='ARB0' units='V'>0.951330618926855 0.951330618926855
0.998364845587902 0.998364845587902 0.983223690288558 875 -
0.58855712679146 -0.587912821960877 -0.771217569058104 -
0.77089541656047 -0.905233022125623 -0.905555174691976 -
0.98222749049865 -0.981905337889957 -0.998657273785101 -
0.000306998656911655 0.249360970543194 0.248716666137523
0.482277007927486 0.481632703543914 0.684910738066098
0.684588585866899 0.844698232689047 0.844698232689047
0.951008466693817 </datavector></experimentResult>
```

BIBLIOGRAPHY

[1] iCampus: the MIT-Microsoft Alliance. iLab: Remote Online Laboratories. [Online] Retrieved April 29, 2008 from http://icampus.mit.edu/projects/ilabs.shtml

[2] J. Harward et. al. The iLab Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories. Proceedings of the IEEE, vol.96, no.6, pp.931-950, June 2008.

[3] Colton, C. K., Knight, M. Q., Khan, R., West, R., "A Web-Accessible Heat Exchanger Experiment", INNOVATIONS 2004: World Innovations in Engineering Education and Research, Win Aung, Robert Altenkirch, Tomas Cermak, Robin W. King, and Luis Manuel Sanchez Ruiz. Arlington, VA: iNEER, 2004, pp. 93-106.

[4] Talavera, D., "On-Line Laboratory for Remote Polymer Crystallization Experiments Using Optical Microscopy", MIT M.Eng. Thesis, 2003.

[5] Amaratunga, K., Sudarshan, R., “A Virtual Laboratory for Real-Time Monitoring of Civil Engineering Infrastructure”, presented at the International Conference on Engineering Education 2002, Manchester (UK), August 18-22, 2002.

[6] Hardison,J., DeLong,K., Bailey, P., Harward, V.J., “Deploying Interactive Remote Labs Using the iLab Shared Architecture. Frontiers in Education (FIE) Conference, Saratoga Springs, New York, October 22-25 2008.

[7] del Alamo, Jesus. Realizing the Potential of iLabs in sub-Sahara Africa. Steering Committee Meeting, June 24, 2005. Makerere University, Kampala, Uganda.

[8] National Instruments. NI Educational Laboratory Virtual Instrumentation Suite (NI ELVIS) .February 1, 2006. [Online] Retrieved March 3, 2008 from http://zone.ni.com/devzone/cda/tut/p/id/3711

[9] National Instrument ELVIS User Manual. [Online] Retrieved March 8, 2008 from www.ni.com/pdf/manuals/373363b.pdf

[10] Gikandi, Samuel. ELVIS iLab: A Flexible Platform for Online Laboratory Experiments in Electrical Engineering. Master of Engineering Thesis, September 2006, Massachusetts Institute of Technology.

[11] Harrison, Bryant .Expanding the Capabilities of the ELVIS iLab Using Component Switching. Master's of Engineering thesis, Massachusetts Institute of Technology, 2006.

[12] Harward, J. et al. iLab: A Scalable Architecture for Sharing Online Experiments. Whitepaper, International Conference on Engineering Education, October 16-21, 2004.

[13] S. Lerman and J. del Alamo, "iLabs Architecture Overview", MIT. [online]. Retrieved June 8, 2008 from http://icampus.mit.edu/iLabs/architecture/

[14] J. Harward. Service broker to lab server API. (Date downloaded: May 1, 2008). [Online] Retrieved June 23, 2008 from http://icampus.mit.edu/iLabs/Architecture/downloads/protectedfiles/Service%20Broker%20to%20Lab%20Server%20API.doc

[15] D. Zych. Lab client to service broker API. (Date downloaded: May 1, 2008). [Online]. Retrieved June 25, 2008 from http://icampus.mit.edu/iLabs/Architecture/downloads/protectedfiles/Client%20to%20Service%20Broker%20API%206.0.doc

[16] Integrated Suite of 12 Instruments for Interactive, Cost-Effective, Multidisciplinary, Hands-On Learning. July 17, 2008. [Online] Retrieved July 25, 2008 from http://www.ni.com/pdf/products/us/cat_nielvisii.pdf