International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:5, 2010

Development of Reliable Web-Based Laboratories for Developing Countries

Teyana S. Sapula and Damian D. Haule

***Abstract*—**In online context, the design and implementation of effective remote laboratories environment is highly challenging on account of hardware and software needs. This paper presents the remote laboratory software framework modified from ilab shared architecture (ISA). The ISA is a framework which enables students to remotely acccess and control experimental hardware using internet infrastructure. The need for remote laboratories came after experiencing problems imposed by traditional laboratories. Among them are: the high cost of laboratory equipment, scarcity of space, scarcity of technical personnel along with the restricted university budget creates a significant bottleneck on building required laboratory experiments. The solution to these problems is to build web-accessible laboratories. Remote laboratories allow students and educators to interact with real laboratory equipment located anywhere in the world at anytime. Recently, many universities and other educational institutions especially in third world countries rely on simulations because they do not afford the experimental equipment they require to their students. Remote laboratories enable users to get real data from real-time hand-on experiments. To implement many remote laboratories, the system architecture should be flexible, understandable and easy to implement, so that different laboratories with different hardware can be deployed easily. The modifications were made to enable developers to add more equipment in ISA framework and to attract the new developers to develop many online laboratories.

***Keywords*—**Batched, ISA, labserver, servicebroker.

I. Introduction

REMOTE laboratory means online experimentation on real processes. Contrary to simulations, which rely on mathematical models, remote laboratories deal with real signals. Laboratory experiments provide students with practical experience that help them better understanding the theory taught in classes. However, normal learners and distant learners often don't have access to such equipment. This is because traditional laboratory instruments are usually expensive such that many educational institutions cannot afford the instruments they require for their students. Sometimes students are overcrowding in laboratory sessions. In addition, laboratory personnel need to be hired to operate the facilities, thus imposing additional costs.

By providing remote access to laboratory equipment to students, the problem of costly traditional laboratories can be overcome by sharing the few laboratory resources available [1].

Authors are with Department of Electrical and Computer System Engineering. University of Dar Es Salaam .Box 35131 Dar Es Salaam, Tanzania. e-mail:teyana@udsm.ac.tz,. e-mail:dddhaule@udsm.ac.tz

Developments of such laboratories are useful in developing countries where funds for education resources are hardly available.. The remote laboratory architecture from MIT termed iLabs are remote laboratories developed to address the weakness 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 [2].

II. Types of Remote Laboratory in ilab Shared Architecture

[2] Divides online laboratory experiments into three categories: batched, interactive and sensor experiments. Each has different characteristics and requirements for the online laboratory. In this paper, the batched architecture is extensively explained and it has three tiers:

Batched experiments: The experiment, which is entirely defined before experiment starts. It is termed as "system of data processing where information is collected into batches before being processed by the computer in one machine to run". Experiment requests can be inserted to a queue and then device can run those one by one. Benefit of this type of experiments become evident especially when the experiment takes long time to conduct and if the system has results saving and retrieving feature as in that case user is not necessary to be online during running the experiment. Multiple principles can be used to schedule experiments, for example first in first out (FIFO) method, shortest job next method (SJN) as well different types of other priority methods.

Interactive experiments: defines interactive processing as opposite to batch: computer mode that allows the user to enter commands, or programs or data and receive immediate responses. System allows user to monitor and control some parameters dynamically during experiment running. This experiment type requires user to be online while running the experiment.

Sensor experiments: User does not have any influence to the measured phenomena, but may choose which data streams to subscribe. She/he monitors and analyses the continuous data streams of the sensor.

Describing Experimental specifications

The batched shared ilab shared architecture consists of three experimental specifications [3], [4], and [5]. These specifications are lab configuration, Lab specifications and ExperimentalResult. The function of these specifications is pass information between client and labserver. This is done

through client/server communication framework, which encodes the specific lab information between client and labserver. Here the XML technology is used to encode the information to be transmitted as plain text.

LabConfiguration: created by the labserver. It contains information about the name of experiment, list of components that make up the experiment and their names and pixel location, the location of the image file, etc. This information enables the student/user to configure the input of the specified experiment.

LabSpecifications: created by the client. It contains the parameters configured by the user and send them to labserver for processing. *ExperimentalResult*: When the experiment is done, the results are packed into the labserver and send back to the client.

III. System Architecture

A. Addition made to Labserver software

The development and implementation of remote laboratories is a very tedious and costly task. A central challenge is to develop a starting point to allow the rapid, easy and flexible creation of remote laboratories. To access many different devices, a high level of abstraction from the developer's point of view must be reached. The presented *framework* simplifies the integration of multiple instruments into an ilab framework.

The labserver is the core of the iLab-shared architecture. It houses the hardware equipment in which the experiments are created and communicate with the equipment to run a particular set of experimental parameters. In addition, it communicates with the client through the the service broker. When the client submits the experiment, the execution takes place. When the experiment has been successfully completed, the labserver notifies the servicebroker that the results are ready for retrieval. The labserver hosts the ELVIS drivers in the presented framework, which are able to interact direct with the NI ELVIS instruments such as function generator, oscilloscope, etc using driver software. The modular can communicate as well as other non-ELVIS instruments.

The labserver has two main parts: LabServer LabVIEW and LabServer Visual Basic. Labview interact direct to hardware. The labserver Visual Basic contains eight components. But the main components are:

(a) Experiment_Engine

It is a stand-alone execution engine which contain the main() subroutine, that checks the database for any queued experiments. If there is any, it de-queues it. The experiment_engine loops through specific database table for new run requests and then forwards them to the wrapper class to call the specific LabView shared library (dll) node. In addition, it takes care of the communications to the real laboratory devices. Furthermore, when experiment_engine calls New() function of the wrapper class, it gets an instance of wrapper class, which can be used as a factory to create a needed type of wrapper by calling the getWrapper function by giving experiment type as a parameter (this can be obtained from setup, as it is created before this call). Now when we call RunExperiment() on the experiment engine, the method is forwarded to the correspondent child wrapper e.g. to TransienAnalyse in case the experiment type is TransientAnalyse.

In the first version of the Educational Laboratories Virtual Instrumentation Suite (ELVIS) weblab [6], the experiment specification xml was parsed and validated in the experiment specification engine and then handled to a specific wrapper to be sent to the laboratory device. In the new model, the setup and wrapper are separated from the experiment_engine as it will be explained in setup and wrapper sections.

(b) Setup

Consist of one or more available terminals (instruments). This class is a parent, created to ease the adding of new experiment setups to the system. It consists of one or more terminals that are specialized using inheritance to a specific terminal or instrument. This project contains three classes:

(i) The *Setup* class

This has setups structure that obtains the terminals from the experiment specification xml. This class has the method parseXMLSpec() which forwards terminal dependent parts of the xml to base terminal class to create a new terminal with the experiment specification parameters chosen by the client (assumes that the experiment specification parameters are validated in validation engine before parsing to setup). The setup will determine whether the experiment being run is time response or frequency response by looking at the parameters send by the client. The result also will look at the parameters used at the input and the data array in the waveform.

(ii) Terminal class

This has the base terminal, contains the fuctions that are common to all terminals. It contains createNewTerminal method to create specific terminal. This class act as factory for creating different types of terminals based on the type of terminal given on parameters that are setupID where this particular terminal belongs, instrumentTypeStr that is the type of terminal to be created. It returns the new child class terminal (for example TerminalBODE class).

(iii) *TerminalBODE* (for example)

This is a class to represent the specific instrument and its parameters. It inherits from Terminal class and so all functions and class variables in terminal class are to be used also here. This class contains a method that creates a new specific terminal corresponding to the given experiment type. For the TerminalBODE the specific parameters to this terminal are waveformType, amplitude, frequency and offset.

(c) Wrapper

This class provides a wrapper around the Labview dlls that

communicate with the ELVIS board to run the experiment. In our effort of reducing the dependence of experiment_engine on the setup type to choosing of the correct wrapper, we generalized wrapper class and then specialized different wrapper by using inheritance. This way we were able to provide understandable way to add more setups to the system and then by using factory method to create wrappers. We finally made the experiment engine independent of specific setups or terminals. The wrapper runs an experiment in the setup (assumes that setup is validated before passing to this function). It returns an arrayList with the waveform values generated by running the experiment. In addition, wrapper contains classes depend on the functions to be done. At the time of writing this paper, two functions are created. These are *ACAnalyze.vb* for Time Response Analysis and ACAnalyze.vb for Frequency Response Analysis. The RunExperiment() method in each class is called from the experiment engine. This method calls the runExperiment() method in the PInvoke class that imports the LabView DLL with the parameters passed from the experiment engine. The DLL returns an interleaved array of the output data back from the LabView code. The array of results are returned back through the results() and give it to the setup method getResultXML().

(d) WebLab Data Managers

The WebLab Data Managers component serves as the primary interface between the Data Persistence Layer of the Lab Server. On the Web Server side, the data managers contain methods defining certain well known interactions with the Data Persistence (Database) Layer and are referenced by the other components within the web server process space.

(e) Experiment Setups

The labserver administrative interface is an ASP website where experiments are created. It interacts direct with the SQL database.

(f) ResourcePermissionManager

Contains the Visual Basic code for database. In particular, the SQL methods are exposed here as a VB.NET component utilized by an ASP.NET page or web service method.

(g) Validation_engine

This is the first thing that 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 parseXMLSpec() method in the setup to extract the experiment parameters and checks these values against the values stored in the database.

B. Addition made to Lab client software

The client is where the students/users specify the parameters to be used in the experiment. It is a Java Applet launched from the service broker. It uses Simple Object Access Protocol (SOAP) to communicate with the servicebroker. When launched, the client is initiated through the GraphicalApplet class and the SBServer class is instantiated in the Applet class. The init() method in GraphicalApplet class create the new WebLabClient. When this happens, the *LabConfiguration()* method in WebLabClient is called. This method fetches the LabConfigurationXML file in the labserver database through the service broker *getlabconfiguration()* SOAP call in the SBServer class. The fetched labconfiguration xml is then parsed to the labconfiguration class method *parseXMLConfiguration().* From parsing the lab configuration, a list of terminals is created. Each instrument has instrument type, instrument number, x-pixel location, y-pixel location and label to identify where the terminal is located in the setup image.

From a list of terminal, a setup is created which represent the current setup. The setup has setupID, name, description, imageURL, and a list of terminals that are present in that experiment. Once the lab configuration has been parsed, the *setup* is stored in the *ExperimentSpecification thesetup* field and the instrument eg *FGEN*, *SCOPE* or *BODE* are created from *terminal* information and stored in the *instruments* vector in the terminal class. Then the *MainFrame* draws the main client elements including buttons and menu bars. The *MainFrame* then calls the *SchematicPanel* and *ResultsPanel*, which draws the axes for plotting later. The *SchematicPanel* uses the 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. When the user clicks on any instrument labels, the dialog box appears. Each instrument has its own dialog box for input parameters user enters.

When the user clicks the Run button, the ExperimentSpecificationXML document is created in the ExperimentSpecification class. This experiment specification is sent to the labserver via the execute SOAP call in the SBServer class. The experiment is submitted to the labserver and the execution is taking place. When the experiment is completed, the labserver send the notification to the client via the service broker. The *RetrieveResult* SOAP request is used to get the *ExperimentResultsXML* from the labserver database. The *parseXMLExperimentResult()* method in ExperimentResult class is used to parse the ExperimentResultXML file.

Client

The client is where the students/users specify the parameters to be used in the experiment. It is a Java Applet launched from the service broker. It uses Simple Object Access Protocol (SOAP) to communicate with the service Broker. When launched, the client is initiated through the GraphicalApplet class and the SBServer class is instantiated in the Applet class. The init() method in GraphicalApplet class create the new WebLabClient. When this happens, the *LabConfiguration()*

method in WebLabClient is called. This method fetches the LabConfigurationXML file in the labserver database through the service broker *getlabconfiguration()* SOAP call in the SBServer class. The fetched labconfiguration xml is then parsed to the labconfiguration class method *parseXMLConfiguration()*. From parsing the lab configuration, a list of terminals is created. Each instrument has intrument type, instrument number, x-pixel location, y-pixel location and label to identify where the terminal is located in the setup image. From a list of terminal, a setup is created which represent the current setup. The setup has setupID, name, description, imageURL, and a list of terminals that are present in that experiment. Once the lab configuration has been parsed, the *setup* is stored in the *ExperimentSpecification thesetup* field and the instrument eg *FGEN* and *SCOPE* are created from *terminal* information and stored in the *instruments* vector in the terminal class. Then the *MainFrame* draws the main client elements including buttons and menu bars. The *MainFrame* then calls the *SchematicPanel* and *ResultsPanel*, which draws the axes for plotting later.

The *SchematicPanel* uses the 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.

IV. Sample Experiment: Frequency Domain Analysis (ACAnalyze)

The Bode Analyzer is used to display frequency response and the corresponding phase angle (Bode Plot) of the circuit. The magnitude against frequency and phase angle against frequency are obtained by making use of sweep feature of function generator and analogue input capability of DAQ device. To perform the frequency response analysis, the Bode Analyzer instrument is required.

The Bode Analyze was not included in ilab ELVIS version 1.0 framework. So Bode Analyzer is first added to ilab framework and then the frequency analysis of the single stage CE, RC coupled amplifier is performed. To add the Bode Analyze to the ilab framework, certain areas have to be changed to accommodate the amplitude, start frequency, stop frequency and steps parameters. These areas include experiment_engine, experiment wrapper, experiment_setups, ResourcePermissionManager, updating the SQL Database, updating the how the Experiment Specification and Experiment Result XML documents are written and parsed, updating the client code such that it will gather and send the appropriate experiment parameters. I will explain in more detail of the changes made in the following sections. The best place to start development is in the labview. The best way to add a new feature is to start by creating a new VI for it. Most of the features on the ELVIS have an associated Express VI, and this is usually the best and easiest way to interact with the hardware.

The frequency response of a single stage amplifier in labview has two stages: BodeAnalyze.vi and AcAnalyzer.vi as shown in figure 1 and figure 2 respectively.

Labview

(a) BodeAnalyze.vi

BodeAnalyzer.vi is first created in the LabView. This vi runs Bode Analyzer parameters from the client and the function generator hardware to sweep the sine waves. The BodeAnalyzer.vi utilizes the Bode Analyzer VI Express that is provided in Labview to run the ELVIS functionalities. This vi is then put in ACAnalyze.vi as a subvi.

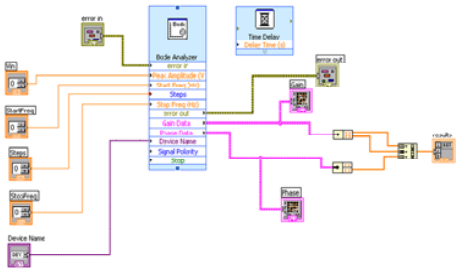

Fig. 1: The BodeAnalyze.vi

(b) ACAnalyze.vi

The ACAnalyze.vi is the entry point to the labview from the DLL. This enables the parameters from the client to get to BodeAnalyze.vi, which runs the experiment on the ELVIS board. When the experiment is done, the results are collected in the result waveform in the BodeAnalyze.vi and passes back to ACAnalyze.vi to DLL.

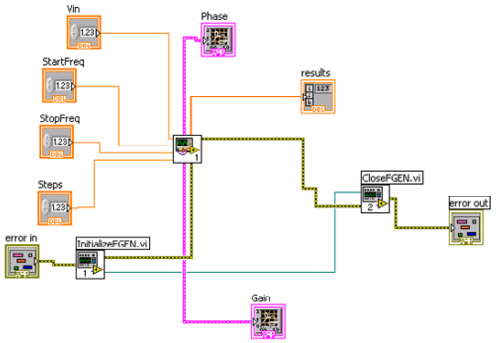

Fig. 2: The ACAnalyze.vi

Labserver experiment_engine

It calls the setup, which has parseXMLSpec method and parses the experiment specification. Then the setup calls the terminal that creates the terminal from the specific terminal class, TerminalBODE.vb to get the specific parameters of that instrument. The RunExperiment() method in the *ACAnalyze* 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

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

Setup

This class has the method parseXMLSpec. This method parses the experiment specification (an XML file that contains amplitude, start frequency, stop frequency and steps parameters chosen by the client). It parses the XML parts from validation engine and delegates each parsing of each terminal to terminal class. The parsed data elements are loaded into class variables for processing by other private and internal methods. The result will look at the parameters used at the input and the data array in the waveform. Then, the gain and phase of the single stage amplifier are determined. For the gain, the x-axis is Frequency in log scale and y-axis is the magnitude in dB in linear scale. For the phase measurement, x-axis is Frequency in log scale and y-axis is the magnitude in degrees in linear scale.

Wrapper

This class provides a wrapper around the Labview dlls that communicate with the ELVIS board for the AC Analyze experiment. It runs an experiment in the setup with parameters amplitude, start frequency, stop frequency and steps. It returns an arrayList with the waveform values generated by running the experiment.

validation_engine

This is the first thing that 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 parseXMLSpec() method in the setup to extract the experiment parameters and checks these values against the values stored in the database.

Database

The changes made in the database are in the the *Setups* Table, the *experiment_type* field is added and *AddSetupTerminal*, the start frequency, stop frequency and steps are added. These are changed to accommodate amplitude, the start frequency, stop frequency and the steps.

The Lab Server administration pages changed so that the new functionality is available to be seen by whoever is making the labs. This can be done by modifying *experiment-setups.aspx* page. Things that should be changed here include adding the new instrument to the drop down lists of available instrument and adding constraint fields for the new instrument.

ResourcePermissionManager

This class file contains the Visual Basic code for WebLabServicesLS database. In particular, the SQL methods are exposed here as a VB.NET component utilized by an ASP.NET page or web service method. The individual methods contain input validation code and invocation of the appropriate SQL stored procedure. Example below shows the validation of bode analyzer parameters.

Client

The first step in adding a new functionality was creating a Bode class. This class extends the Instrument class. An instrument identifier number should be added in the Instrument class for the new instrument.

Then, add the function for the new instrument by extending the SourceFunction class. The function for new instrument stores the information parameters for the instrument. In this case the ACAnalyzeFunction is the function for the BODE instrument and stores information like the waveform type selected by the user and the parameters like amplitude, start, stop, steps, gain and phase, etc are associated with the waveform. The *ACAnalyzeFunction* similarly has the fields to store the start, stop frequencies and the steps per decade to be used for the bode analysis.

After that, next step is to make the instrument label for the sweep feature. This extends from the *InstrumentLabel* class. This class has a chooseImageName() method that specifies the name of the image to be used for the instrument. You will need to draw an image for the instrument and put the image in the 'img' folder and put the name of the image in the method. You can also add a case for your instrument in the updateToolTip() method in the InstrumentLabel class to show a tool tip. Next thing was to create the dialog user interface that will be used to modify the instrument. This will appear when the label for that instrument is placed. The FGENDialog class was added.

To handle the parsing of the lab configuration the a case is added for the instrument in the parseXMLLabConfiguration() method in the LabConfiguration class. This just assigns a type to the instrument so that it is recognized when the instrument in created in ExperimentSpecification class.

When the bode analyzer is added to NI ELVIS code, the client launches the single stage amplifier with the bode analyzer parameters which are amplitude, start, stop and steps. The results are the gain in dB against frequency in logarithm scale and phase in degrees against frequency in logarithm as shown in Fig.3 (a), (b).

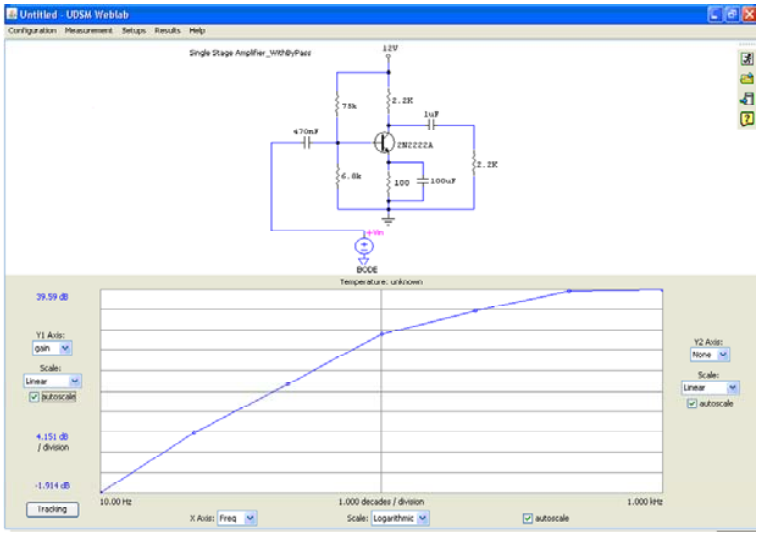

Fig. 3(a): The gain against frequency

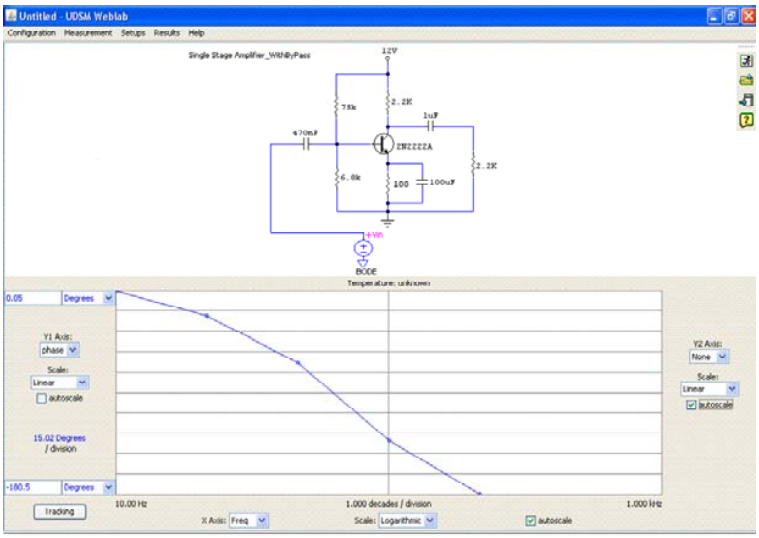

Fig. 3(b): The phase against frequency

V. Conclusion

The simplified architecture made to the ELVIS platform greatly increase flexibility of the batched ilab framework. The results obtained in web accessible of lab experiments is very close to traditional labs as the experiments are done in real equipment contrary to simulations which the experiments are done depending on mathematical models

References

[1] C.C. Ko, B. M Chen, J. Chen, Y. Zhuang and K. C. Tan, "*Development of a Web-Accessible laboratory for control experiments on a coupled tank apparatus*," IEEE Transactions on Education, vol. 44, no. 1, pp. 76–86, 2001.

[2] J.V. Harward, J. del Alamo, V.S. Choudhary, K. deLong, J. Hardison, S.R. Lerman, J. Northridge, D. Talavera, C. Wang, K. Yehia, D. Zych, " *iLab: A Scalable Architecture for Sharing Online Experiments*", International Conference on Engineering Education, Gainesville, Florida, 2004.

[3] P. Bailey, "*The online experiments shared architecture and the future of web based laboratory Experiments*", 2004.

[4] V.N. Gerardo, "*Design and Implementation of a Feedback Systems Web Laboratory Prototype*", AUP Final Report, MIT EECS, 2004.

[5] G. Viedma, J. D. Isaac and H. L., Kent, "*A Web-Based Linear-Systems iLab*"submitted to the 2005 American control conference.

[6] S. Gikandi, "*A Flexible Platform for Online Laboratory Experiments in Electrical Engineering*", Master of Engineering in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2006.