Lab Station for Remote Measurement and Control in Teaching Real-Time Embedded Systems and Software Engineering

Carlos Daboin* and Janusz Zalewski*

*Florida Gulf Coast University, Ft. Myers, FL 33965, USA
(Tel: +1 239-590-7317; e-mail: zalewski@fgcu.edu)

Abstract: This paper presents an approach to the implementation of a web-based lab station for teaching embedded systems, real-time systems, and other related courses in programs, such as software engineering, computer science or engineering, electrical engineering, mechatronics, robotics, etc. The lab involves not only data acquisition and control access to remote devices but also a capability of remote software development, uploading and debugging. The specific device used to create a lab station in this project is a remotely controlled eBox 2300 with measurement and control capabilities, running Windows Embedded 6.0.

Keywords: Embedded systems, real-time systems, software tools, software engineering education, robotics, microcontrollers, FPGA.

1. INTRODUCTION

With current, rapid technological advances, courses in disciplines such as real-time systems, embedded systems, and corresponding parts of the Software Engineering curriculum (LeBlanc and Sobel, 2004) require significant changes. In parallel with respective curricular changes in the discipline, one particular associated problem has to be addressed urgently, that of the contents of the labs.

In a traditionally oriented real-time software engineering lab, students are exposed to higher levels of software development, such as requirements specification and architectural or detailed design, with the use of corresponding tools, for example, DOORS for requirements and IBM Rational Rose or Telelogic Rhapsody for design (Fig. 1, Kornecki and Zalewski 2006). At the implementation level, however, it is rather rare (with a few exceptions, Kornecki et al. 2000) that a significant number of tools are used. There seem to be two reasons for this situation: first, a rather chronic lack of appropriate equipment in academic environments, and second, the need for its continuous maintenance, both factors contributing to a relatively high cost of creating and using a corresponding lab.

Due to recent technological progress, the situation gets even worse, because most of the new embedded devices, developed for and installed in industries such as telecommunications, aviation, railways, medicine, etc., require a remote, Internet-based access. And, as far as software engineering is concerned, it is not only the remote access to device’s operation, which is the most critical, but the ability to remotely develop software for these devices, with a capability of remotely uploading and debugging it.

This paper shows how the respective educational challenges can be addressed, and outlines one particular project that tries to resolve some of the related issues. The paper is structured as follows: Section 2 gives the background by introducing the reader to the web-based real-time software engineering lab at Florida Gulf Coast University (FGCU), Section 3 discusses general issues about the specific remotely controlled lab station, based on the Windows Embedded kernel, Section 4 provides some details of the implementation, and Section 5 derives some conclusions.

2. BACKGROUND

Although multiple papers on remote labs in other disciplines have appeared in recent years, for example in Control Engineering (Duan et al. 2005), including several surveys (Gravier et al. 2008; Howard et al. 2008; Ma and Nickerson 2006), there is a relative lack of respective literature referring
to the use of such labs in software engineering courses. A few papers were published on software development and experimentation in embedded systems (Callaghan et al. 2006; Yin and Sun 2008), however, there is essential lack of those that discuss issues related to remote software development.

Real-time embedded devices are typically either computerized sensors supplying measurement data to the remote server or more advanced microcontrollers deriving signals, either to move some mechanical parts or take other control actions upon a command received from a remote user. These devices are more and more commonly accessible over Internet, not only to receive operational instructions from, and send data to the clients, but also for a variety of other functions, such as checking the device’s status, diagnosing malfunctions, etc.

Correspondingly, the approach to design such labs in the academic environment must be significantly different than in other engineering or science disciplines. While most of the web-based labs in science and engineering do not require remotely done changes in software running on the remote devices, in software engineering this is the primary focus of lab activities, in addition, for example, to software development and performance assessment.

![Fig. 2. General concept of web access to the lab.](image)

A detailed presentation of a current status of the web-based real-time software engineering lab at FGCU, with an architecture shown in Fig. 2, has been recently given in (Zalewski 2009). In this section, only a general background on current developments is presented, with several examples of labs already in operation.

2.1 General Objectives

Following the basic assumptions of building the lab, outlined in Section 1 and illustrated in Fig. 1, the major objective is to have several remote lab units that would allow experimentation with the use of several low-level implementation tools, at the following three layers:

- programming languages
- real-time operating system kernels
- target hardware platforms.

To maximally benefit the experiments, the use of industry-strength tools is recommended. The programming language layer should involve languages most commonly used in real-time embedded systems, that is, C/C++ and C#, Java, possibly Ada, LabVIEW, and a hardware design language, such as VHDL or Verilog. Real-time kernels should include those most successful and commonly spread around industries, possibly such as VxWorks, Windows Embedded, and a game platform. Among an array of hardware devices on the market, one can include those, which are of particular importance to industry, or those that have specific educational values and can be purchased with academic discounts. Platforms, such as PowerPC, multicore CPUs, Atmel microcontrollers, wireless sensors, and FPGA boards, have been selected for this project.

On some platforms, such as the majority of wireless networks and LabVIEW, web services are included as their natural component. This functionality simplifies writing embedded web applications and it was taken into account when considering specific application topics for the projects. Reports on LabVIEW and wireless projects connectivity are the subject of future papers; here we only concentrate on applications, which provide a challenge for remote code development, uploading and debugging.

Currently, there are almost a dozen projects developed at FGCU for a web-based lab. The selection was driven by the need to cover the entire spectrum of embedded applications, including:

- remote programming of a microcontroller with a data acquisition device
- remote development of a multiuser web-based computer game
- remote programming of FPGA boards
- remote development of testing software for web-based access of data acquisition devices
- remote programming of an embedded computer for data acquisition and control.

In the following, some of these projects are outlined to give the reader a better idea of the scale and focus of the lab developments.

2.2 Overview of Selected Projects

Remote programming of an ATMEAL microcontroller. The purpose of this work is to provide a sample project that will introduce students to the use of microcontrollers in embedded systems. An Atmel AVR® STK500 Microcontroller Starter Kit is used as a teaching vehicle. A simple external device, in this case a temperature sensor connected to an I2C bus, is used to establish a basic data acquisition (DAQ) system, with remote access to the microcontroller for software development (see Figure 3).

In this sort of project, the student learns an entire array of concepts and respective skills related to microcontroller’s architecture, operation, and its connectivity with external devices, which include low-level C programming and software development in the AVR Studio environment. What is im-
important from the point of view of a remote lab are the web-based functions, which involve:

- developing web interface (client GUI) for remote access
- preparing the server host for web access
- programming the board from a remote location
- uploading programs for web-based access to the device
- testing the remote operation.

Just like in the microcontroller project, in the learning process, students acquire several concepts and skills related to remote software development for embedded systems, typically: developing web-based GUI for remote access, programming and testing the board from a remote location, and perhaps most importantly, combining experiences with different programming technologies: Java, C# and ASP.NET.

Remote programming of FPGA boards. The purpose of this remote web lab is to have the user develop their VHDL code locally and then upload it to the board via a website to see the operation of a real FPGA board with a webcam (see Fig. 5). The FPGA web lab assumes the user will have knowledge of VHDL and have their VHDL code ready before starting the operations.

Remote development of data acquisition testing software. In several application domains, such as high-energy physics experiments, automated data acquisition systems have to be monitored frequently for continuous operation. Taking reliable readings by certain sensors and scientific instruments is crucial to understanding the investigated phenomena. If this function is not reliable, the results are often of limited usefulness. Thus, the capability of remote monitoring and control in real time is crucially important.

Dasavouret and Nogiec (2003) proposed a solution to create a toolkit, which gives technicians the ability to see the status of the developer page and read information on how to use the system. A part of the basic user interface illustrated in Fig. 4.

Fig. 3. User interface for remote access of the Atmel.

Remote development of a multiuser web game. The purpose of this project is to develop a software system that permits users to play a web game or develop a web-based game console remotely using web access.

A HYDRA game development kit has been selected for this project, which uses a multicore Parallax Propeller CPU with eight 32-bit RISC processors, with a variety of ports available for external access. Users are able to use a web browser to play the game using arrow keys, upload new games via the developer page and read information on how to use the system. A part of the basic user interface illustrated in Fig. 4.

Fig. 4. Part of a developer’s interface to the web game.

The learning process combines two aspects: strengthening the knowledge of digital circuits, as acquired in a digital systems course, and integrating it with new experiences in VHDL software development via the web, using a professional hardware development tool. The double role of the project allows students to build elements of knowledge required in modern development of embedded systems, based on the convergence of hardware and software.

Remote development of data acquisition testing software. In several application domains, such as high-energy physics experiments, automated data acquisition systems have to be monitored frequently for continuous operation. Taking reliable readings by certain sensors and scientific instruments is crucial to understanding the investigated phenomena. If this function is not reliable, the results are often of limited usefulness. Thus, the capability of remote monitoring and control in real time is crucially important.

Dasavouret and Nogiec (2003) proposed a solution to create a toolkit, which gives technicians the ability to see the status
of DAQ devices quickly through any computer with an Internet connection. This function is accomplished in a configuration shown in Fig. 6, with an embedded computer running a VxWorks real-time kernel (Zalewski et al. 2009). The educational value of this project lies in combining low-level programming aspects (VxWorks real-time kernel and PowerPC architecture) with developing web access to the device for testing (socket programming and website development).

Fig. 6. Single board computer used as a lab station running VxWorks kernel.

3. REMOTE CONTROL IN WINDOWS EMBEDDED

3.1 General Background

The objective of this project is twofold: first, to teach how to develop software for a remote device (in this case, remotely controlled camera and temperature sensor), controlled by an embedded computer running Windows Embedded CE kernel, and second, to learn how this can be done with a remote access from a host workstation to the target computer.

The system has several hardware components, as shown on a system’s network diagram in Figure 7: a PC host and an embedded target eBox 2300 (based on a Vortex86 system-on-a-chip microprocessor) running a server that accesses two USB devices - a camera with servo motor and a sensor. The client application can run anywhere on any computer with the Internet connection.

3.2 eBox 2300 Hardware and Software

The eBox 2300 (Fig. 8) is a thin client, a low-cost computer box devoid of CD-ROM players, diskette drives, and expansion slots. Thus, a thin client accesses programs and data from a server instead of storing them locally, and the software that performs the majority of its operations reside on a server rather than the local eBox computer. The eBox 2300 can be also used as a server for an application that makes the eBox 2300 accessible to clients/users in a LAN or WAN.

Fig. 7. System’s network diagram for remote control.

Such an application is normally a client/server application based on C#, and can have functions to transmit and receive measurement and control data using sockets.

Fig. 8. Embedded device eBox 2300: rear view.

When eBox 2300 plays a role of a server, its main use is to host applications controlling sensors and other devices connected to it. These applications run under the Windows CE 6.0. This OS is designed and customized based on the needs of the server application. The OS and server application are developed on more powerful and faster host computer. Then, a Windows 6.0 CE image and the C# server application are deployed onto the eBox 2300 (the target).

4. LAB STATION IMPLEMENTATION

Implementing the lab station is done in four steps, gradually expanding on the complexity of the project, to show the students various aspects of software development for Internet aware embedded applications. The four steps involve: the initial DAQ-only application, its expansion to the robot control, robot control with a game controller, and creating remote software updates.

4.1 Initial Project

The initial project is to use the eBox for DAQ functions only. This involves reading a temperature sensor and stream-
ing live video captured from a USB camera connected to the eBox. The assumption is that once the eBox 2300 server application is operational, the clients over the Internet may request and receive the temperature and the streaming video from eBox 2300 and display it on the remote computers. The client application may be deployed anywhere on the host that runs Windows XP, because the .NET framework is needed to run such C# client. After the client application is downloaded, the user is able to connect to the server running on the eBox 2300, read the temperature and see the live video streamed from the remote location.

The challenge with developing an eBox application in this sort of environment is multi-fold. First, an operating system kernel has to be built and deployed to the eBox 2300 to allow the thin client to run applications and drivers. Second, specific drivers need to be deployed or installed on eBox that allow access to specific devices connected to it. In this particular project, communication has to be established between the eBox 2300 and the video capture device (USB camera). Third, to create C# applications for the eBox, an SDK (Software Development Kit) has to be built for the Visual Studio development environment. Finally, a client/server application that allows communication between remote PC’s and the eBox 2300 must be developed and deployed. Additionally, both the eBox 2300 server application has to be configured to stream the video being captured from the USB camera, and the client application on the remote PC has to be set to display video.

Fig. 9. Sample user interface for remote access.

Consequently, what the student learns in such a development project is not only programming (which in fact is a prerequisite for the course), but a whole array of concepts and skills necessary for remote development, as follows:

- developing an operating system image for the target, and deploying it on the target
- installing the necessary drivers on the target
- building the Software Development Kit on the host
- developing the target application in C# (this is the traditional programming part, to which most conventional labs are limited)
- deploying an application to the target, and
- developing an independent user interface for a client to access the target’s server.

A sample user interface of a client, a GUI that displays remote site information for the Internet clients, demonstrating the interaction of the client/server software components to generate the system output, is shown in Figure 9. The left-hand side of the GUI handles the connectivity with the camera, and allows moving the camera in horizontal directions as well as taking the snapshots. The right-hand side of the GUI is independent of the camera connectivity and is dedicated to handling the temperature readings, with separate connectivity and alarm setting.

4.2 Project Expansion

A more sophisticated project involves building a robotic arm, connecting it to the eBox 2300, and developing control software for its five degrees of freedom. The project requires the following non-trivial steps, some of them can be skipped depending on the nature of the course for which the arm is used:

- assembling the robotic arm (Fig. 10), composed of several dozen parts, and wiring its cable connections
- application development as described in Section 4.1, which includes building an operating system kernel and the SDK
- expansion towards including a more sophisticated robot control, for example with the use of an Xbox 360 gamepad controller accessory
- updating the robot control software and remotely uploading it to the eBox 2300

The possibility to remotely update the robot control software is particularly intriguing, but requires a significant amount of conceptual and development work. It is done automatically in the following steps:

- close the current application and delete its files
- connect to the ftp server and download the new files
- start the new application.

The GUI to control the robotic lab station with a gamepad is shown in Figure 11.

5. CONCLUSIONS

With industry moving with fast pace towards development of various sorts of embedded devices that are Internet aware, college education of software engineers and computer scientists faces a real challenge. Student projects for remote software development have to be included in the curricula, which require building resourceful and expensive labs – an objective that can be difficult to accomplish in academic environments.

In this view, FGCU’s Computer Science program is pursuing the creation of an embedded systems lab for a variety of applications and development platforms, with the objective to prepare graduates of programs in software engineering
and other computing disciplines to respond to current market demands. One particular lab station described in this paper allows students to learn concepts and skills in a cross-section of computing disciplines, including operating systems, programming languages, real-time computing and networking, and apply them in practice.

From the instructor’s perspective the organization of a lab presents a significant challenge, because of the time and effort needed to maintain continuous operation of devices and software involved. While technically feasible, the operation of the lab involves several serious additional administrative issues that have to be resolved, such as network access, lab scheduling, etc. The reader is welcome to test the operation of the lab station by downloading the application from the school website at: http://satnet.fgcu.edu/~cedaboin/ebox2300, and running it for the IP address 69.88.163.25

ACKNOWLEDGEMENTS

This work has been done under a grant from the National Science Foundation, Award No. DUE-0632729. Views expressed herein are not necessarily those of the NSF. The following students in FGCU Computer Science program are gratefully acknowledged for their contributions to the development of the lab: Vincent Rosa, Scott Marks, Olexiy Kovtunenko, Robby Porter, Joanne Sirois and Hank Grimmick.

REFERENCES


