AC 2009-2378: TEACHING SOFTWARE DEVELOPMENT FOR MODERN REAL-TIME DATA ACQUISITION AND CONTROL

Janusz Zalewski, Florida Gulf Coast University

Janusz Zalewski is a professor of Computer Science and Engineering in the School of Engineering at Florida Gulf Coast University University. His research interests, in addition to software engineering education, include real-time safety-critical systems.

Andrew Kornecki, Embry-Riddle Aeronautical University

Andrew Kornecki is a professor in the Department of Computer and Software Engineering at Embry-Riddle Aeronautical University. His research interests, in addition to software engineering education, include real-time safety-critical systems.

Jerzy Nogiec, Fermi National Accelerator Lab

Jerzy Nogiec is the Software Development and Support Group Leader at Fermi National Accelerator Laboratory and an adjunct professor of Computer Science at the Illinois Institute of Technology. His research interests, in addition to software engineering education, include distributed systems and data acquisition systems.

Teaching Software Development for Modern Real-Time Data Acquisition and Control

Abstract

Modern data acquisition and control systems, in the most demanding real-time applications, such as sensor networks, flight control systems, accelerator control, road vehicle control, and others, are all distributed and for proper operation require very different programming techniques than traditional systems. Typical software engineering curricula rarely include respective methodologies of software development for such systems. If they do, their courses mostly concentrate on the specification and design of software for distributed systems, but stop short of including thorough treatment of implementation and testing issues.

The current work builds upon previous experiences of the authors and involves projects in teaching software development for distributed real-time data acquisition and control systems, with focus on implementation and testing. In particular, a process of organizing and using a web-based HTTP server for educational purposes in remote testing and operation of software based on VxWorks and Windows CE platforms is described. The concept has been adopted from work done at the Fermi National Accelerator Lab.

Introduction

Typical software engineering curricula rarely include methodologies of software development for the most demanding real-time applications, such as sensor networks, flight control systems, accelerator control, road vehicle control, and others, which are all distributed and for proper operation require very different programming techniques than traditional systems. If there are curricula that do this, their respective courses mostly concentrate on the specification and design of software for distributed embedded real-time systems, but stop short of including thorough treatment of implementation and testing issues[1)]. The major reason for this seems to be primarily the difficulties with acquiring, operating and maintaining appropriate hardware and system software, which normally require knowledge of a device architecture combined with low-level programming techniques and significant attention paid to technical support, which is rarely available in typical programs at the college level.

In an effort to provide students with such knowledge, several universities are using equipment donations to offer courses supporting the missing component. In many cases the only college level education on embedded real-time systems can be obtained from electrical engineering or hardware-focused computer engineering programs. The students and faculty have good grasp of the designed system hardware but too often the software component of the system is of marginal quality. The faculty and students do not have enough background and experience to produce quality software. The truth is, however, that the software is responsible for most of the system functionality. Software became a major component of the modern systems and the quality of software, more than that of the hardware, is the critical element of the system success.

Any software developer for industries dealing with real-time systems has to be familiar with techniques and tools that would allow him to stay competitive. Thus, the job of the educators is to enhance the curriculum by including in it those techniques and tools of real-time software development that are currently used in industry. Based on their experiences with real-time data acquisition and control projects in avionics and high-energy physics, the authors developed and previously published a suite of experiments for teaching implementation and testing of distributed software, including topics such as semaphores, message queues, scheduling, priority inversion, device drivers and real-time kernels[2,3)]. The implementation platform involves typical single board computers based on PowerPC architecture and VxWorks real-time kernel with its integrated development environments: Tornado and Workbench.

The objective of current work is to build upon previous work and experience and enhance the educational process by expanding experiments in teaching software development to projects in distributed real-time data acquisition and control systems, with focus on implementation and testing. In particular, a process of organizing and using a web-based HTTP server for educational purposes in remote testing of software based on aforementioned platforms is described. The concept has been adopted from work done at the Fermi National Accelerator Lab[4)].

The paper is organized as follows. First, we present an overview of traditional exercises used in teaching the implementation of real-time systems. Then, we address a web-based server approach for remote testing of data acquisition and control systems with the VxWorks real-time operating system kernel, and finally apply some of these concepts to the development of embedded systems with Windows CE.

Previous Work: Typical Programming Exercises with a Real-Time Kernel

In previous work[3)], we have shown how concepts, such as timing, multitasking, synchronization and scheduling, priority inversion and interrupt handling can be introduced via exercises and experimentation, using a VxWorks real-time kernel and its associated environment. To achieve respective goals, a dedicated real-time laboratory supporting the development of real-time software has been created as the necessary component of the modern curriculum. Such laboratory has to include a platform for development of embedded systems in a host-target environment. The students shall have access to a development environment supporting all lifecycle including requirements, design, implementation, and testing. As a critical element of the lab the students must be able to develop code on a host and download, debug and test it on the target. The target run-time system must have characteristics of a real-time operating system supporting concurrency, synchronization and communication primitives, interrupt handling, pre-emptive scheduling and deterministic behavior.

Both Embry Riddle Aeronautical University and the Florida Gulf Coast University created appropriate labs allowing their undergraduate and graduate students to develop expertise in host-target real-time software development. The selected development environments include Tornado and Workbench based stations running VxWorks real-time kernel, thanks to the software grants from Wind River Systems Academic Program.

Once such environment is in place, the challenge is to create an infrastructure to allow students easy access to software development and experimentation, and include the lab experiments in the course structure to teach students basic real-time concepts.

To facilitate the learning process, a basic series of laboratory experiments to be performed by the students has been created. The sequence addresses the issues of timing, multi-tasking, shared resources and locking, communication, signals and interrupts, and scheduling. As the operating system plays an important role in developing real-time software, the experiments focus on using the kernel primitives by the application programs. The experiments were designed to be completed by a student during a single semester, or during a course of independent study, while learning the appropriate theory component in the classroom.

Each lab experiment contains the following sections: a) introduction, b) objectives, c) description, d) example program, e) procedures, f) follow-on experiment, and f) additional information.

The introduction section gives a brief description of the experiment and how it applies to a real-world situation. The objective describes what should be learned from the experiment. The description section touches the theory behind the real-time concept – the topic of the lab experiment. It also explains the terms and operating system constructs used to implement the example program. The example program is a fully functional program that the student can compile and run. The program demonstrates the real-time concept - the objective of the experiment. The follow-on is to be completed by the student. The student may either modify the existing program or write a new one.

By experimenting with the demo and modifying the program, students gain experience in dealing with a real-time environment. At the same time, the design and coding time is reduced since most of the code is already written. Student can focus on experimenting and learning "how does it work."

A laboratory report must be submitted with each experiment. The report has a pre-defined format. The report presents the results of running both the example program and the student's modified/derived program. Finally, the report requires an analysis. In this way, the student not only has to complete the experiments, but also analyze them critically.

In addition to the lab explorations, after learning the basic real-time concepts, the students engage in a small team project to apply the acquired knowledge and skills. In the past we had a few projects implemented on the VxWorks platforms. All projects resulted in prototypes implemented on VME VxWorks target with user interface on a remote Unix workstation (using a GUI builder or Java applet). Some examples of such projects included a real-time data acquisition and control system, a real-time GUI implementation, and others.

These software development projects went one step further than the lab experiment. The main objective of such projects was to present the students, working normally in teams, with a challenge to develop a prototype software system for a specific application. For example, for the data acquisition system, the task was to use a VMI/VME-4514A analog I/O board and VMI/VME-2532 digital I/O board with the processing on VME Motorola target board running VxWorks, to use a PT-326 process trainer and accomplish a close-loop proportional control. The data acquisition implemented via sample sequencing of operations used for software design purposes is presented in Fig. 1, as a sequence diagram developed by a student team in a design phase.

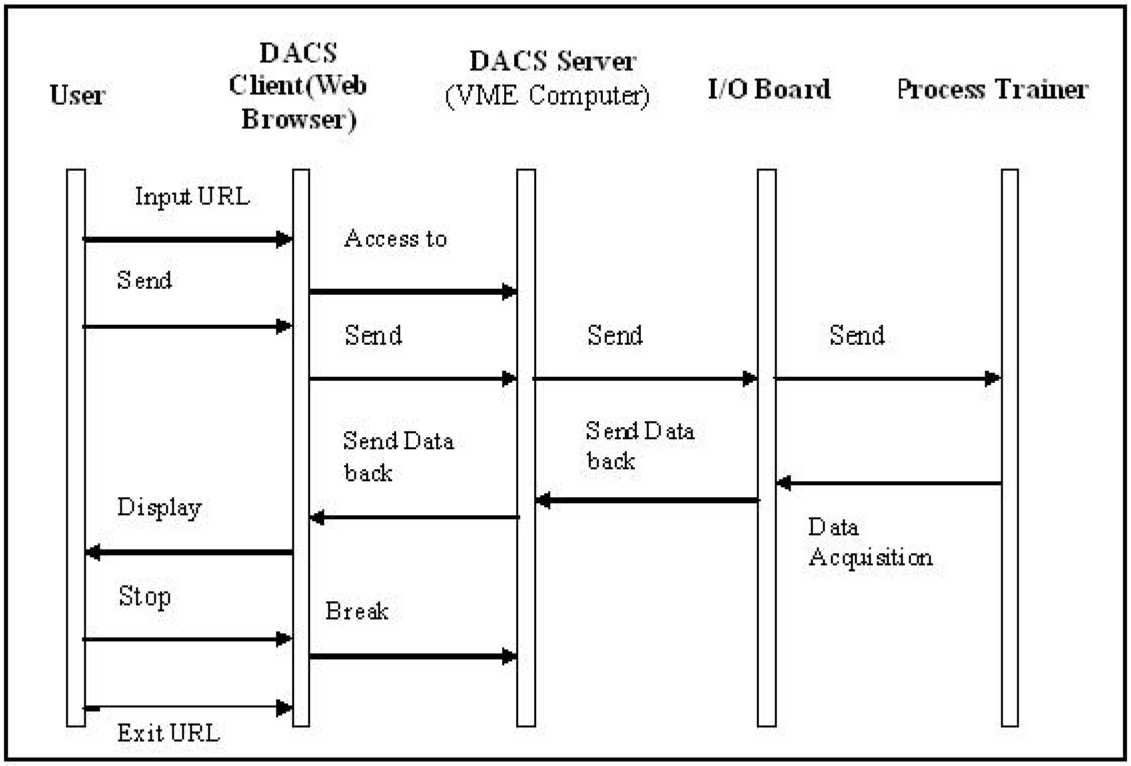


Figure 1. Sequence Diagram for Data Acquisition and Control Experiment

The experiments outlined above, as well as simple team projects, although meeting their individual objectives, are not sufficient for comprehensive education of software engineers who would like to be competitive on the job market in developing real-time embedded data acquisition and control applications. There are two reasons for this situation. First, the complexity of embedded devices grew dramatically over the last decade and new tools became available to deal with it. Secondly, availability of the web dramatically changed the architectures and accessibility of embedded systems, so that remote access to distributed devices and experiments has to be provided on a standard basis[5)]. Thus, the schools and educators who provide graduates to respective industries have to consider changing and expanding their curricula towards including the latest technologies and teaching corresponding competencies and skills. In the next two sections we are trying to address some of these issues.

Web-based Access to Experiments and System Operation

When using automated data acquisition and control systems, one wants to monitor them frequently for continuation of operation. This is important in practice, because data acquisition's (DAQ) whole purpose is to continuously take reliable readings using specific external instruments. If this function is off even slightly, the results in real-time safety-critical applications could be considered completely worthless. That is why emerging issues have to be resolved as quickly as possible. One proposed solution[4)] is to create a type of toolkit which gives technicians the ability to see the status of DAQ devices quickly through any computer with an Internet connection.

In the solution outlined in[4)], the authors have a host web server which collects information from a set of data acquisition devices that have their own specialized web servers installed on them. The host server takes the information from the devices and publishes it to a viewable webpage, so that it can be accessed from a computer with an Internet connection. This is emulated in a class project with the VxWorks based Wind River single board computer (SBC) being the DAQ device (Fig. 2), and a Windows based desktop being the host server.

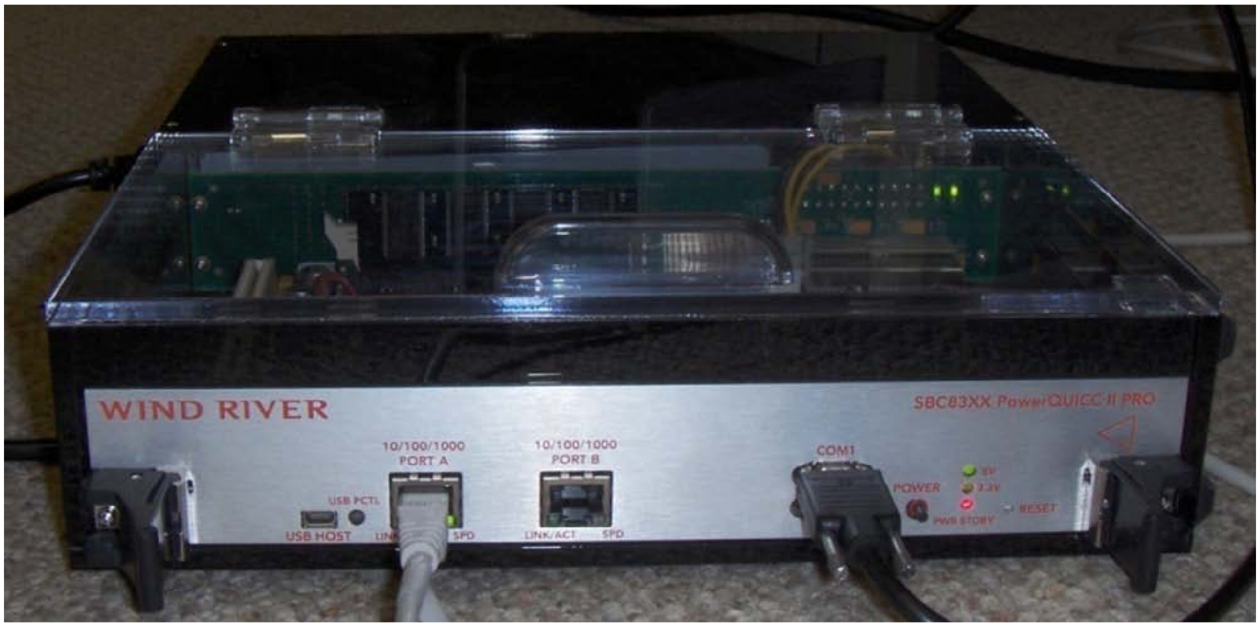


Figure 2. Wind River Single Board Computer Used for Monitoring.

The software requirement specification document is particularly designed to define the boundaries of the VxWorks real-time kernel connectivity and functionality. A sample of student developed functional requirements are listed below:

- The web toolkit shall consist of a set of HTML pages displayed in a web browser, specialized web servers, test handlers, and support utilities.
- The system shall be comprised of a central web server used for selecting a test target, and specialized target web servers running on tested or monitored computers.

- The software shall allow the user to select a system, then a computer to test, and finally a device or monitoring program.
- The tested computer shall be selectable either by its name or its function.
- The web server shall execute a proper test handler and send a reply page back to the requestor.
- The application shall receive updated information from DAQ devices after a request.
- The software shall verify that all data received from DAQ devices are valid.
- The software shall verify if inputs from the user are valid.
- The software shall output the requested DAQ devices data.
- The software shall replace the existing DAQ devices data with new data as received.
- The system shall provide an error message if invalid data are requested.
- The system shall provide an error message if it is unable to find a DAQ device.

The basic software design is very simple and shown in Figure 3. It is composed of modules to request and receive data from the clients and publish the data to the web server when prompted by the user. The host computer is only used to update/maintain the system.

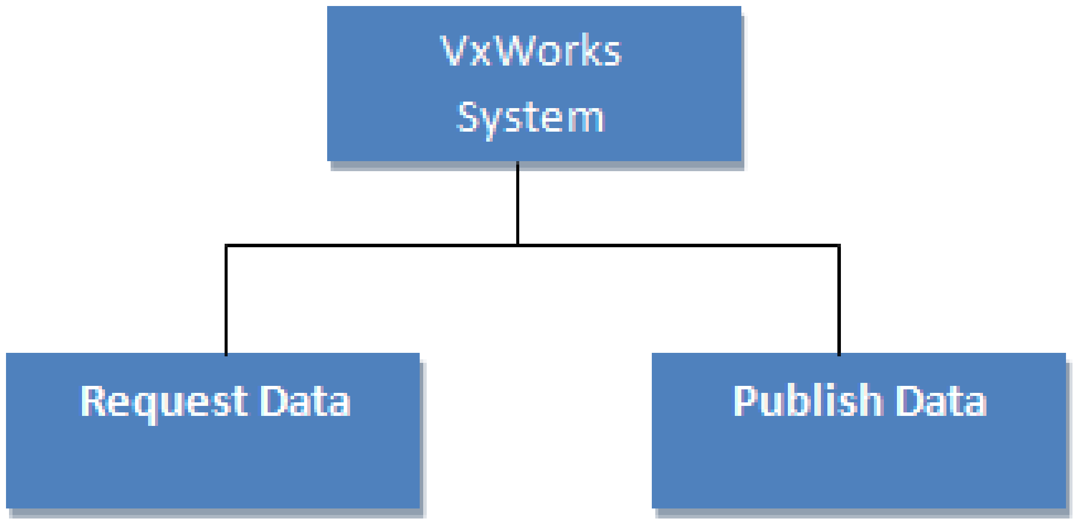


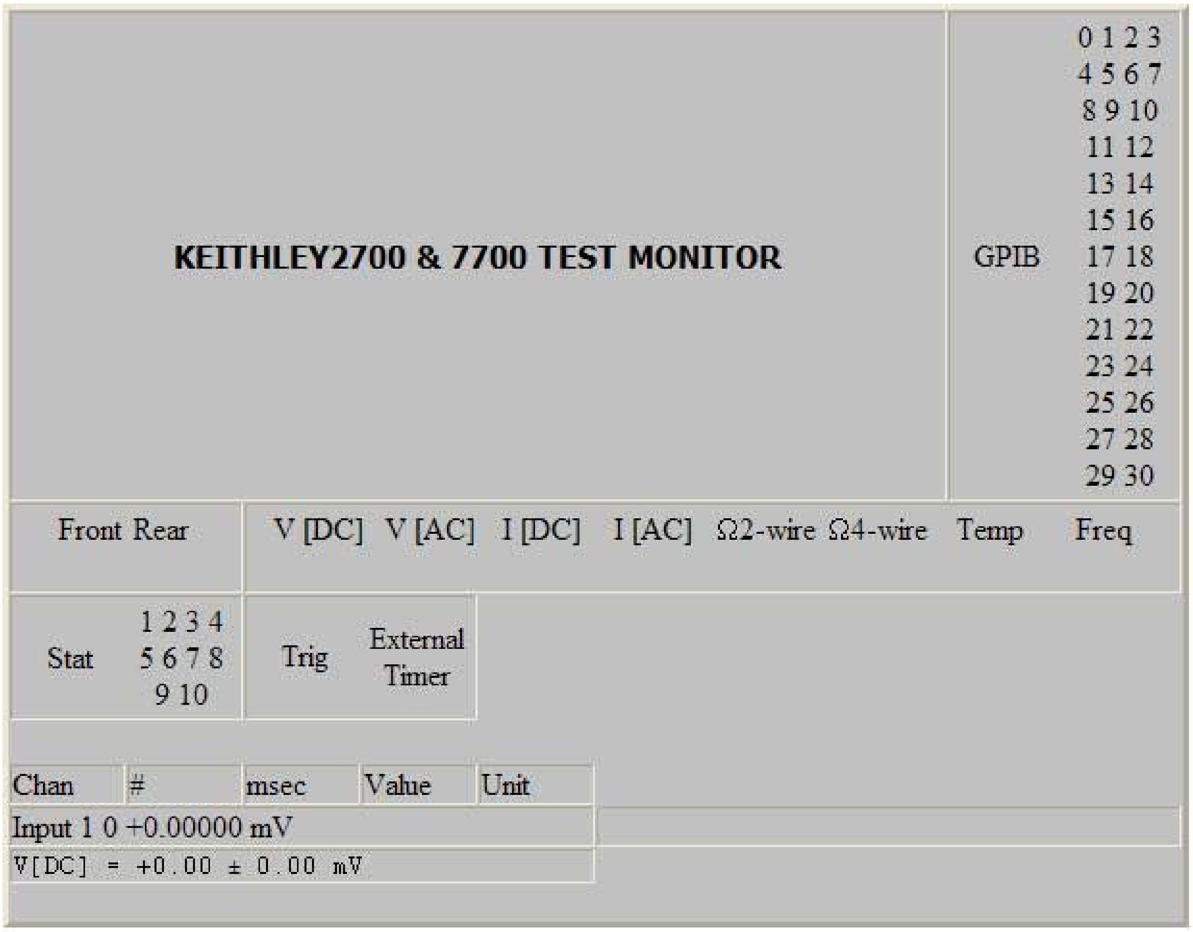
Figure 3. Hierarchical Decomposition Diagram

The basic implementation of the server program is a portable web server intended to be used for local application. The web server supports CGI with both POST and GET methods and is intended to be a deployment platform. Only files located in this directory and its subdirectories can be served to the user.

The web server program running under VxWorks can be used to enable querying RTDB (Real-Time Data Base) from web browsers. The web browser can be used also as a front end for various CGI programs allowing for development of various diagnostic systems, calibration applications, hardware tests, and resources monitoring applications.

The set of environment variables defined by the webserver program is limited to only a few values. The VxWorks server executes all CGI programs as functions called in the server context, which may cause the server to malfunction in cases when CGI programs are misbehaving. Therefore, all CGI programs should be entirely tested before deploying them to the VxWorks platform.

The server is currently implemented as a single-threaded program. Therefore, one should avoid CGI programs that have undetermined or long execution times. Such programs could block the server for a longer than allowed time. A sample user interface for a typical data acquisition device connected via a GPIB bus is shown in Figure 4. An example of a sample code for a simple CGI program for the VxWorks server is shown in Appendix 1.



```
Input 1 0 +0.00000 mV
V[DC] = +0.00 ± 0.00 mV
```

Figure 4. Sample Data Acquisition Device Accessible via the Server.

In summary, this project plays three different and complementary roles in the course. First, it verifies the student's knowledge of typical real-time programming constructs used on a development platform for a real-time embedded target device. Second, a student learns the remote development, with an embedded target located on a network as a remote device. Finally, the student gets familiarity with the operation of data acquisition and control devices in a distributed environment, typical to many modern industrial and research applications.

Remote Measurement and Control with the eBox 2300 Thin Client

While the VxWorks kernel and its development platform have dominated the most critical hard real-time applications, Windows CE is a widespread platform used in soft real-time systems. This puts a pressure on educators in the area of embedded data acquisition and control systems to include projects that would expose students to related issues, comparable to that encountered in more critical systems.

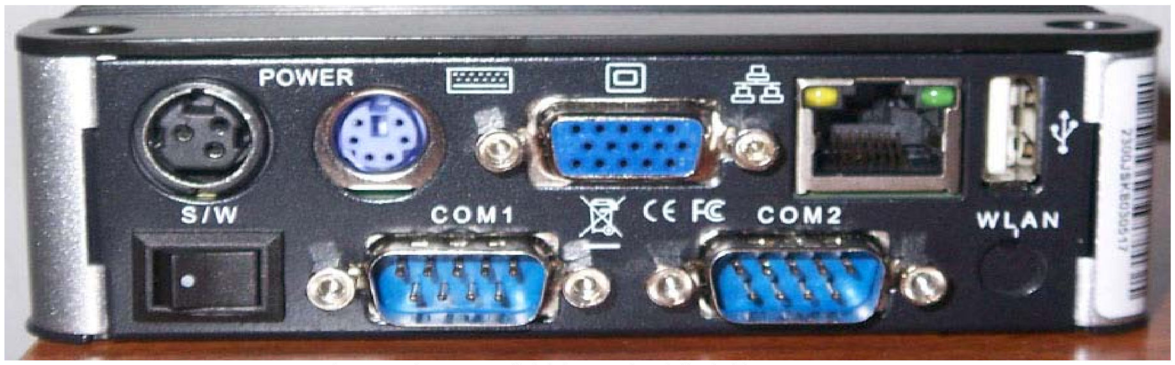


Figure 5. eBox 2300 Embedded Computer.

The eBox 2300 is a device that can run under Windows CE and serve as a useful tool for teaching embedded systems. It is known as a thin client that can be used as an embedded computer to control remote experiments accessible to clients/users via the network (Fig. 5). The application described in this section is a client/server software based on C#, and its function is to transmit video, temperatures, and rotation commands using sockets. The data transmission is used for surveillance and control of a remote site from a remote computer.

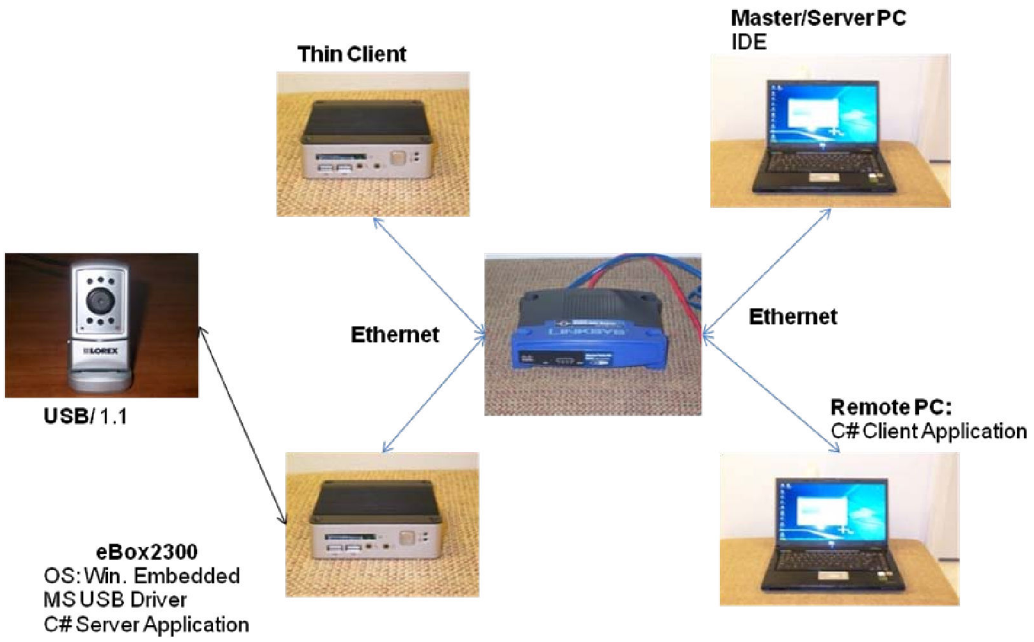


Figure 6. Configuration of the eBox 2300 Client/Server Application.

The server application's main use is to stream live video capture from a USB camera connected to eBox 2300, send temperature readings read by a sensor, and send rotation commands to a servo attached to the USB camera to change the viewing angle. It runs under the Windows CE 6.0. The entire configuration is shown in Fig. 6.

The client application main use is to receive the streaming video from eBox 2300 and display it locally. It also displays the temperature readings taken by the temperature sensor located at the remote location. In addition, the client application sends rotation commands to rotate the servo motor axis attached to the USB camera. Thus, the eBox 2300 makes use of a video surveillance web camera, the temperature sensor, and a servo motor to track and control environmental changes on a remote site.

Software requirements are divided into input, output, processing, and non-functional requirements. A sample input requirement on the client commands looks as follows:

- Operations or commands input from the personal computer shall include:
    - Command for connecting to server application
    - Command for closing connection to server application
    - An input field to enter the thin client's remote IP
    - An input field to enter remote application's port number
    - Command for moving the server motor to the left, and to the right
    - Command for retrieving remote temperature readings.

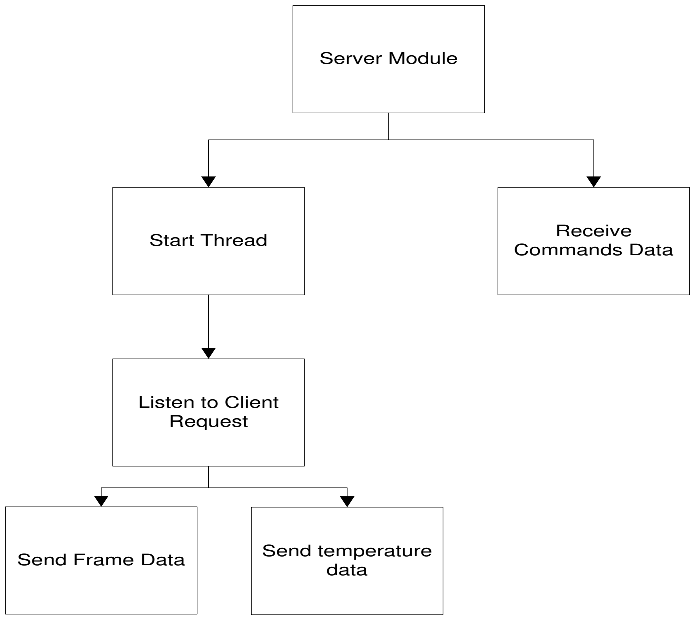


Figure 7. Server Module Structure Chart

A sample design of the server module is illustrated in Figure 7. The server application is composed of five submodules; start thread, listen, send frame data, send temperature data, and receive commands data. The video server application in this project has borrowed the design implementations from the Servo CAM project developed at Georgia Institute of Technology[6)] for the camera driver SDK (wrapper).

A sample implementation of a GUI is shown in Figure 8. It displays the real-time video surveillance of remote site location. The video is displayed over a picture box, that is activated when the user presses the connect button, and the remote host IP and port have been entered in the respective textbox. The GUI also displays temperature reading over a textbox every time the user presses the read temperature button. In addition, it allows the user to close the application at any time.

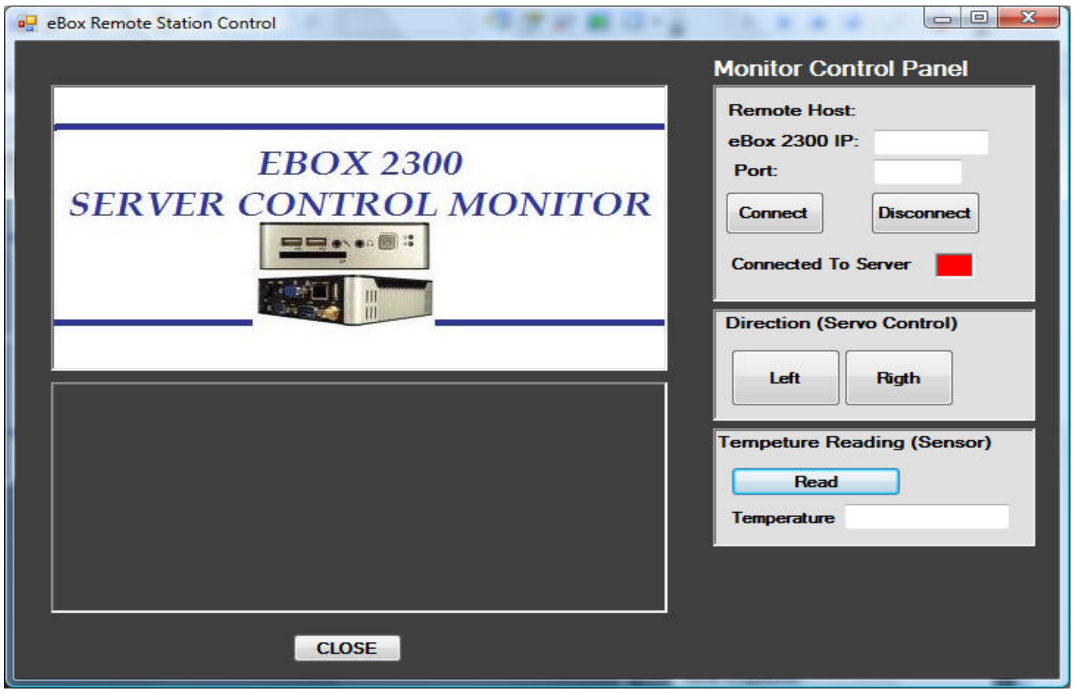


Figure 8. Sample Implementation of a Graphic User Interface

The client was developed in C# as a Windows application. The application uses asynchronous sockets to connect to remote video server running on the eBox 2300. The source code of the client is shown in the Appendix 2.

A limited example of dynamic testing of the client GUI application software involved the following procedure:

Inputs

- Remote eBox 2300 IP address in GUI's textbox entry.
- Remote application port number in GUI's textbox entry.
- Button event: pressing the Connect button on the GUI video server remote panel.

Expected Outputs

Connection to the video server application. The video being captured by the USB camera should be displayed over GUI's picture box, and it should be an image display of 15 fps.

Actual Results

- The client application communicated with the video server, after the IP, and port number were entered on its respective textboxes.
- After pressing the Connect button, the client GUI's picture box displayed the video being capture by the USB camera connected to the eBox.

Conclusion

With the growing complexity of real-time, embedded data acquisition and control applications, software engineers and educators are facing new challenges in being productive in the development of such systems. The problem is further exacerbated by the widespread use of Internet in connecting embedded devices, which generates additional issues.

The authors of this paper are concerned that the teaching methods, which were relatively advanced a decade ago, are no longer sufficient to educate graduates who would be competitive enough on the job market after graduation. Therefore a new significant step in education of real-time software engineers is proposed in this paper: introducing in the classroom environment such projects that would reflect current situation in professional software development.

Two platforms used in real-time embedded data acquisition and control systems have been introduced in senior design project courses: VxWorks and Windows CE for respective single board computers. Both projects were developed with a typical waterfall process model in mind, with four phases of development, including requirements specification, design, implementation and testing. The emphasis on implementation and testing, without neglecting the first two phases of the process, allowed students to apply knowledge of traditional real-time programming in applications of significantly higher complexity, involving networked development and operation.

Acknowledgements

This project has been funded in part by the National Science Foundation under a grant on Web-based Real-Time Software Engineering Lab, Award Number 0632729. Thanks go to FGCU Computer Science students, Joanne Sirois and Carlos Daboin, for their contributions to the VxWorks and Windows CE software development. Wind River Systems donation of software development platforms and VxWorks real-time kernel is gratefully acknowledged.

References

1. Zalewski J., "Cohesive Use of Commercial Tools in a Classroom," *Proc. 7th SEI Conf. on Software Engineering Education*, San Antonio, TX, January 5-7, 1994, J.L. Diaz-Herrera (Ed.), Springer-Verlag, Berlin, 1994, pp. 65-75
2. Kornecki, A., Zalewski J., "Real-Time Laboratory in a Computer Science/Engineering Program," *Proc. IEEE Workshop on Real-Time Systems Education*, IEEE Computer Society Press, Los Alamitos, CA, pp. 73-79, 1996.
3. Kornecki A., J. Zalewski, D. Eyassu, "Learning Real-Time Programming Concepts through VxWorks Lab Experiments," *Proc. 13th Conf. on Software Engineering Education*, Austin, Texas, March 6-8, 2000
4. Desavouret E., J. Nogiec. *Web Tools To Monitor And Debug DAQ Hardware*. Report FERMILAB-Conf-03/131, Fermi National Accelerator Lab, Batavia, Ill., June 2003
5. Shah R., "Device Server Technologies for Remote Monitoring of Devices," *ECE Magazine*, pp. 33-36, April 2007, http://www.embedded-control-europe.com/ece_magazine
6. Jarvis K, D. Kimn, J. Belisle. *ECE 4180 Final Project: ServoCAM*. Fall 2007, http://users.ece.gatech.edu/~hamblen/489X/F07PROJ/ServoCAM/

Appendix 1. Example Code of a Simple CGI Program for the VxWorks Server

```
/* Simple CGI program that returns the parameters sent to it.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef vxworks
int simplecgi(void)
#else
extern char **environ;
int main(int argc, char **argv)
#endif {
  int content_length = 0;
  char *p_request_method = NULL;
  char *p_user_input = NULL;
  char *p_content_length = NULL;

  printf("<html>\n"
         "<head>\n"
         "<title>CGI</title>\n"
         "</head>\n"
         "<body>\n"
         "<pre>\n");

  if((p_request_method=getenv("REQUEST_METHOD")) != NULL) {
          printf("REQUEST_METHOD = %s\n", p_request_method);
          printf("DOCUMENT_ROOT  = '%s'\n", getenv("DOCUMENT_ROOT"));
      if(strncmp(p_request_method, "GET", 3) == 0) {
          printf("CONTENT_LENGTH = %s\n", getenv("CONTENT_LENGTH"));
          printf("USER_INPUT     = '%s'\n", getenv("QUERY_STRING"));
      }
      else if(strncmp(p_request_method, "POST", 4) == 0) {
          if((p_content_length=getenv("CONTENT_LENGTH")) != NULL) {
              if(sscanf(p_content_length, "%d", &content_length) == 1){
                  printf("CONTENT_LENGTH = %d\n", content_length);
                  if((p_user_input=malloc(content_length+1)) != NULL){
                      if(fread(p_user_input, 1, content_length, stdin) > 0) {
                          p_user_input[content_length] = '\0';
                          printf("USER_INPUT     = '%s'\n", p_user_input);
                      }
                    free(p_user_input);
                  }
              }
          }
      }
  }
 printf("</pre>\n</body>\n<html>\n");
 return 0;
}
```

Appendix 2. C# Windows Remote Video Viewer Client Application Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.IO;
using System.Threading;

namespace VideoClientWin
{
    public partial class Form1 : Form
    {
        TcpClient client;
        delegate void PicHandler(Bitmap b);

        public Form1()
        {
            InitializeComponent();

            client = new TcpClient();
            client.ReceiveBufferSize = 2000000;
        }

        private void Go(IAsyncResult ia)
        {
            TcpClient tc = ia.AsyncState as TcpClient;

            while (!tc.Connected)
            {
                continue;
            }
            NetworkStream ns = tc.GetStream();
            BinaryReader br = new BinaryReader(ns);

            try
            {
                while (true)
                {
                    if (client.Available > 0)
                    {
                        int len = br.ReadInt32();

                        if (len > 100)
                        {
                            byte[] bt = br.ReadBytes(len);
                            MemoryStream ms = new MemoryStream(bt);
                            Bitmap b = new Bitmap(ms);
                            ms.Close();
                            this.Invoke(new PicHandler(UpdateImage), new
object[] { b });
                        }
                    }
                    Thread.Sleep(0);
                }
            }
            catch (Exception) { }
        }

        private void UpdateImage(Bitmap b)
        {
            this.pictureBox1.Image = b;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            string ip = textBox1.Text;
            int port = 0;
            bool works = int.TryParse(textBox2.Text, out port);
            if (works && !client.Connected)
            {
                AsyncCallback ac = new AsyncCallback(Go);
                client.BeginConnect(ip, port, ac, client);
            }
        }
    }
}
```