First Look at an Internet-enabled Embedded Systems Design Course

J.W. Bruce and Jordan Goulder
Mississippi State University

Abstract

The proliferation of Internet access has drastically changed the way people do business, recreate, learn, and do their daily tasks. It is widely believed that the Internet and its successors will be called upon to enable interactivity between devices that today are mundane. These interconnected devices are commonly called Internet appliances. Specifically, an Internet appliance is

> a machine designed for a specific function that also has a built-in Web-enabled computer. Internet appliances include small devices created especially for e-mail and Web surfing, as well as such diverse products as personal digital assistants (PDAs), smart phones, Web TV, and Web-enabled refrigerators and microwaves.

Mississippi State University's has recently revised its undergraduate computer engineering (CPE) program with input from alumni and advisory employers. The CPE program will have a focus on embedded computer systems. Embedded systems form a rich application source through which the CPE education can be made relevant. Embedded computer systems are a timely subject that is immediately useful to students in their senior design projects. Furthermore, a large number of our CPE graduates currently use or design embedded computer systems in their jobs. With the availability of low-cost, designer-friendly Internet connectivity, the design course is centered on the design, prototyping, and debugging of an embedded systems for internet appliances. The target application of the first offering is a personal weather station web server.

Evolving from an earlier course on embedded systems that are more traditional, or "free-standing" [1]-[2], this new course relies on cooperative, team-based learning and design, and seamlessly resumes where prerequisite courses ended. Design in the course requires formalized hardware and software design inspections [2]-[4]. The design inspections serve as a convenient time for software product measures to be collected. The quantitative measures document the nature, origin, and other vital characteristics of each design defect and are frequently used in industry [5] [6]. Finally, the design practices described in this paper help students to develop teaming and communication skills that are often neglected by traditional engineering curricula.

1 Introduction

In [1], the author presents a team-based progressive embedded systems design course that, in addition to providing the technical embedded systems knowledge, develops team and communication skills in situations emulative of industry. The course was a success by many accounts; however, student teams abandoned sound design practices in attempt to meet the demanding 16-week "time-to-market" constraint. Team members produced defect-riddled

designs and the design schedules slipped due to an unproductive test-redesign-test development cycle.

In [2], the course was retooled to use a lightweight design process based loosely on proven software engineering standards [3]-[5] which detects defects during design. This development process has been used with success in the subsequent offerings of the design course in [1]. The resulting student designs are typically on time and of high quality. Furthermore, students report satisfaction with the experience, because of both the visible results at course end and the perceived relevance of the process that they used.

The course described in [1] and [2] has made a visible impact on the computer engineering program at Mississippi State University (MSU). Computer engineering student projects in the capstone design course have greater complexity and are of higher quality compared to previous years. One obvious disadvantage to the course is that prerequisite requirements necessitate that the course is taken very late in the program, usually the penultimate or final semester. While the impact on the capstone design course is clear, the faculty was confident that capstone student projects would be improved further if the embedded systems concepts can be applied earlier. Moreover, changing technology constraints precipitated a redesign of the junior level introductory microprocessor course at MSU. The faculty decided that the first microprocessor course should focus on small microcontrollers, low-level programming, and basic hardware interfacing – much of the technical subject concepts of the course described in [1] and [2].

With fundamental embedded systems content being taught in the junior level course, the senior level embedded systems course is free to concentrate more on system concepts and integration issues that are more common in engineering practice and industry. The abstract system issues are better suited to the team-based design and industry-based standards described in [1] and [2]. In addition, the new course could be structured in such a way that computer science and software engineering students are more easily integrated into student teams. (The design project in [1] was sufficiently low-level that CS and SE students lacked the necessary circuits and electronics background.) The new course needed to examine embedded systems concepts at a systems level, build on the "new" early experiences in the first microprocessors course (taken by all EE, CPE, CS, and SE students), and be timely and relevant to current engineering practice. It was decided that the course focus on a rapidly growing requirement in many new embedded systems of significance – internet capability.

The nucleus of the course was forming. The course would employ the successful teaming and communication skill growth in [1] and use the standards-based development process in [2]. The embedded systems project would be a practical embedded system that included internet capabilities and would effectively integrate computer engineering students (more hardware focused) with computer science and software engineering students (more software and networking focused). With a diverse collection of students, the project needed to require only a minimal amount of additional expert experience. Since everyone has a rudimentary knowledge of weather and meteorological phenomena, it was decided that the team would design, test, and deploy an internet weather station. The remainder of this paper describes the structure of the internet weather station and discussion of the course's first offering in the Fall of 2004.

2 Internet Weather Station

The internet weather station provides current weather conditions via the Internet and keeps archives of past weather. A small Java-powered web server handles data archiving and dynamic web page generation. A microcontroller subsystem controls and interprets weather sensor data. The two embedded computers communicate via a two wire serial interface Inter-Integrated Circuit (I^2C) network [7] with the web server acting as master and the microcontroller serving as the slave. Figure 1 shows the system diagram with all components and their connectivity. The next two sections will describe the architecture of these two microcontroller and the web service subsystems.

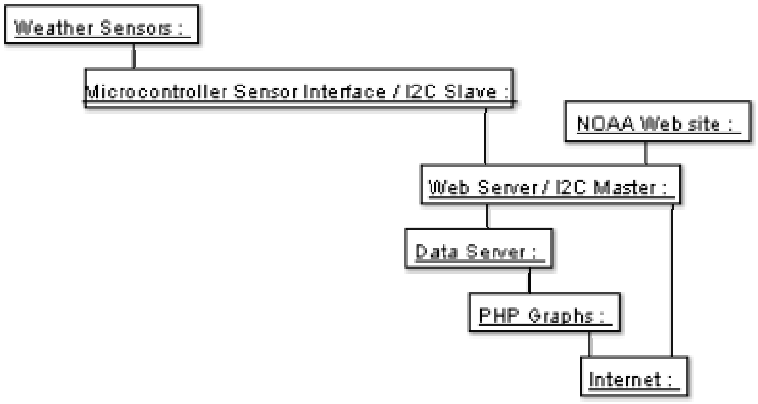

Figure 1 Internet weather station system diagram

2.1 Microcontroller Subsystem

Low-level sensor control and interfacing is performed by the microcontroller. Sensor control requires accurate timing and is not compatible with the small web server and the unknown and variable delays introduced by TCP/IP networks. Student teams were instructed to use the Microchip PIC18F242 microprocessor to control and measure weather sensor data. The course requires the same lab parts kit purchased by the students in the earlier, junior-level microprocessor course. The microcontroller subsystem has four major responsibilities:

- Read the raw sensor outputs
- Average the sensor outputs
- Calculate the meaningful sensor values from the averaged data
- Respond to sensor value requests from the web server

Development for the microcontroller is done with the same tools as the earlier course. However, students are required to follow the development, design review, and documentation process described in [2].

2.1.1 Reading the Sensors

A calibrated, industrial quality weather sensor package (EnviroMonitor from Davis Instruments) was secured and mounted on the roof of the four story Simrall Electrical and Computer

Engineering building at MSU. Sensors are multiplexed and transmitted via an eight-conductor Cat5/RJ45 cable to the microprocessors laboratory on the third floor.

The sensor signals represent six quantities in three signal categories: analog, digital closure, and frequency. The weather sensor signals are described in Table 1. Analog sensors create a voltage output corresponding to their quantity, wind direction, air temperature, and wind direction compass heading. Closure signals represent discrete weather events (anemometer revolution, rain drops) by temporarily closing a contact switch. Switch closure pulls a logic-high voltage (5V) to logic-low (ground). Frequency signals represent a weather quantity (humidity) by a square wave frequency. The microcontroller generates a control signal used by the sensors to multiplex two weather sensor signals on the same cable conductor.

Sensor	**RJ45 conductor signal**	**Type**	**Control state**
Wind direction	2	Analog	Low
Wind speed	4	Closure	Low
Temperature	7	Analog	High
Solar Radiance	2	Analog	High
Rainfall	5	Closure	----
Humidity	8	Frequency	High
Signal	**RJ45 conductor signal**	**Notes**	
Control	1	Low=0V, High=5.0V	
Sensor voltage supply	3	Requires at least 6.5V w.r.t. ground	
Ground	6		

Table 1. Sensors and sensor cable specification

The wind direction sensor creates an output voltage between 0 and 2.5 V that changes linearly as the anemometer revolves. Due north (0º) is 0V, east (90º) is 0.625V, south (180º) is 1.25V, and west (270º) is 1.875V. Wind direction is measured by the microcontroller's internal analog-to-digital converter (ADC). Wind speed is represented by switch closures on every anemometer revolution. The switch closure signals the microcontroller via its input capture timer that records the period between closure which is directly related to wind speed.

Temperature is represented by an analog voltage between zero and 2.5V. The temperature sensor is a thermistor placed in parallel with a 42.4kΩ resistor. The resistor and thermistor parallel combination is in series with another 42.4 kΩ resistor connected to the 5V sensor reference. The control signals switches the thermistor into the voltage divider. The changing thermistor resistance changes the value of the lower voltage divider resistance. The resulting temperature-voltage relationship is nonlinear and is shown in Figure 2.

Solar radiance measures the solar energy incident on wavelength selective photodiode on the sensor set. The solar radiance sensor generates a voltage between 0 and 2.5 V. The relationship between voltage and solar radiance is linear with 1.67mV representing $1W/m^2$.

Rainfall is measured by a rainfall collection bucket. The rainfall sensor momentarily closes a closure switch for each 0.01" of rainfall. The rainfall signal is obviously asynchronous to all

other events and is transmitted on a dedicated conductor. Each rainfall closure event signals the microcontroller via an external interrupt.

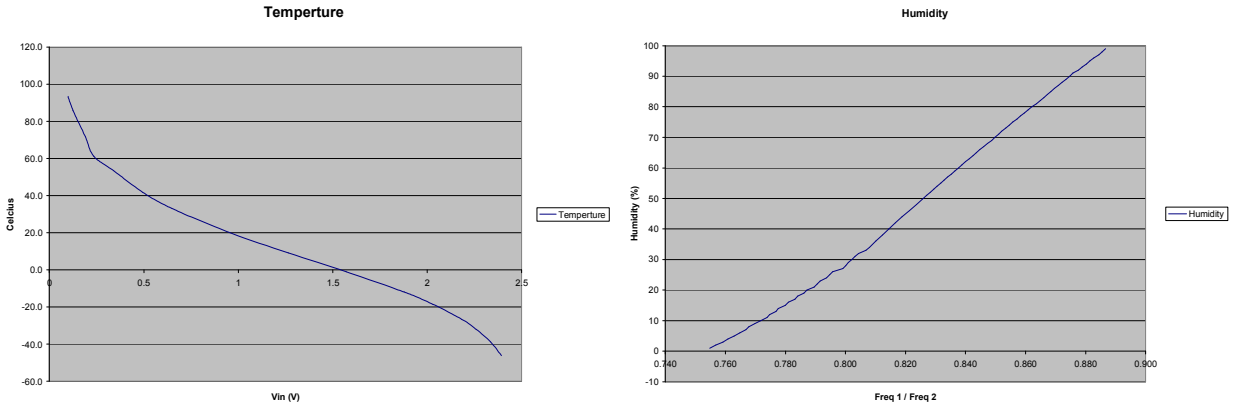

Figure 2 Temperature and humidity transfer curve

Relative humidity is measured with a variable capacitor the change with humidity. The variable capacitor is adjusts the frequency of a square wave oscillator. To account for temperature-dependent effects on the oscillator frequency, a reference square wave is generated on conductor 8 when the sensor control signal is low. Relative humidity is determined by the ratio of the two square wave frequencies removing the common temperature dependent effects. Both frequencies are measured with the microcontroller's input capture timer modules. Figure 2shows the relationship between frequency ratio and relative humidity.

2.1.2 Averaging and Calculating the Values

The sensor control signal is generated by the microcontroller. Each signal quantity with the exception of rainfall is measured repeatedly over during a 1-3 second interval and an average value is calculated. The low pass filtering characteristic of the averaging operation removes signal noise picked up by the 60m sensor cable. Sensor values are stored into circular buffers and averaged during the period when other sensors are being read. Average values are used in the calculations performed when the web server requests a weather sensor value from the microcontroller.

2.1.3 Sending the Values

The microcontroller acts as an I2C slave that responds to requests from the web server (I2C master). I2C communication can be initiated at any time. The web server sends a special command byte that is interpreted by the microcontroller, which then performs the actions that correspond to the command.

2.1.4 Microcontroller Subsystem Architecture

The operation of the microcontroller software is best modeled with state machines and activity flow diagrams. Figure 3 and Figure 4 show the notation used in the paper for state chart and activity flow diagram notations, respectively.

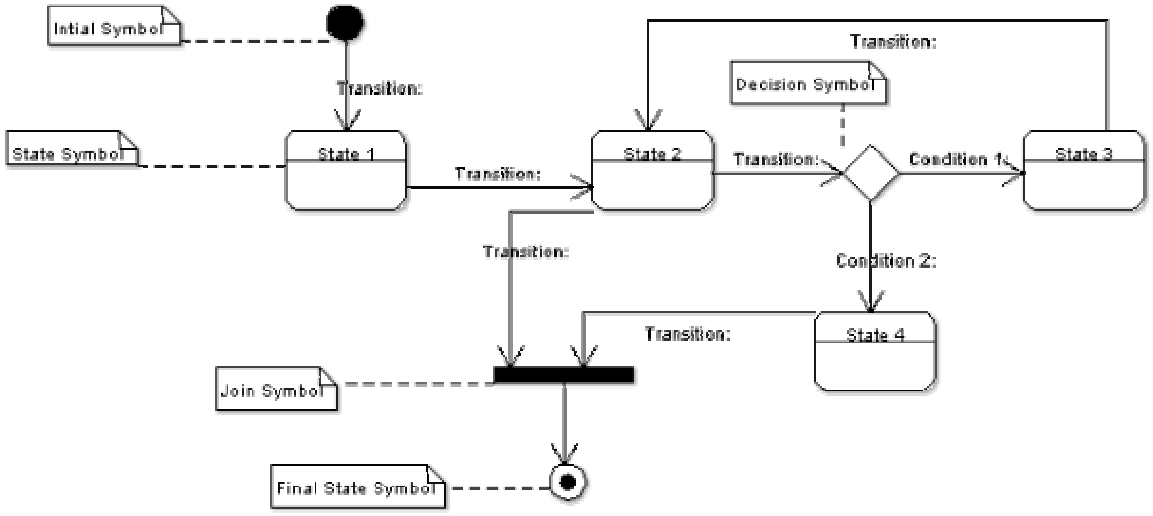

Figure 3 State Chart Diagram Notation

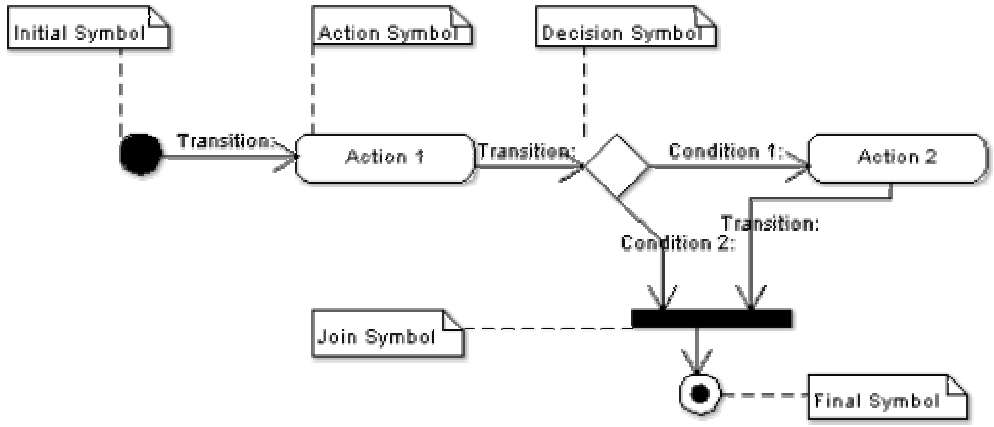

Figure 4 Activity Flow Diagram Notation

Figure 5 shows the state chart for the overall architecture of the microcontroller subsystem. Sensor data averaging and I2C processing are handled in the main state. The system ISR handles the control signal and the A/D conversions.

Interrupt service routines provide timely response during sensor data collection. The rainfall ISR counts the number of rainfall closures. The wind speed ISR measures the period of the

anemometer wind speed closure sensor. The humidity ISR measures the frequency of the relative humidity sensor square wave output. The sensor ISRs are small, providing the minimum processing required servicing the request and obtaining the needed data. More computationally intensive data processing (table lookup, unit conversion, averaging, data extrema storage) is done in the "main" loop.

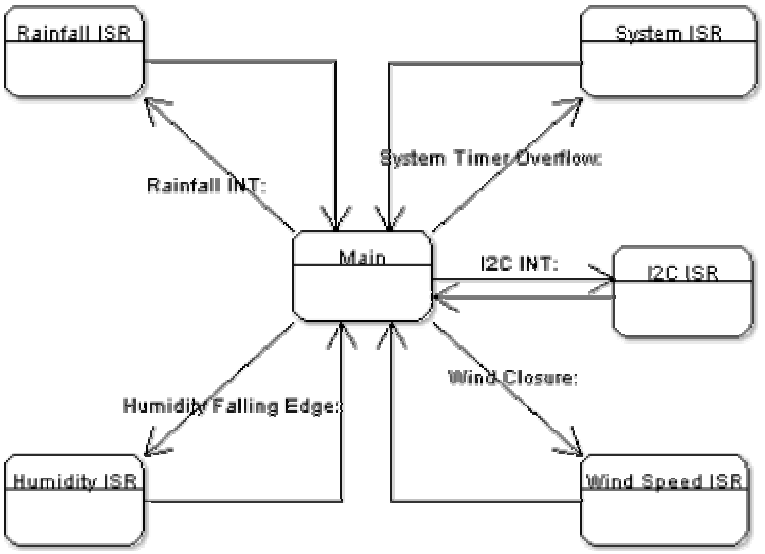

Figure 5 Overall Architecture State Chart

The I2C ISR responds to the I2C requests from I2C masters. I2C slave response requires compliance with the I2C specification and has multiple states. Figure 6 shows the I2C transfer ISR activity flow diagram.

2.2 Web Server Subsystem

The web server in Figure 1 is the subsystem that coordinates the operations of the internet weather station. The design specification document specified the Java-based web server with standard network protocols (HTTP, FTP, DHCP, and SMTP). The network protocols give the web server ample communications options and the Java language and run-time environment support modular programming and intellectual property reuse. The weather station uses an internet processor platform based on the TINI specification [8].

The web server subsystem had three main responsibilities:

- retrieve weather sensor data from the microcontroller subsystem,
- log weather data for use in long-term weather archives, and
- create and serve dynamic web pages displaying weather data over the internet

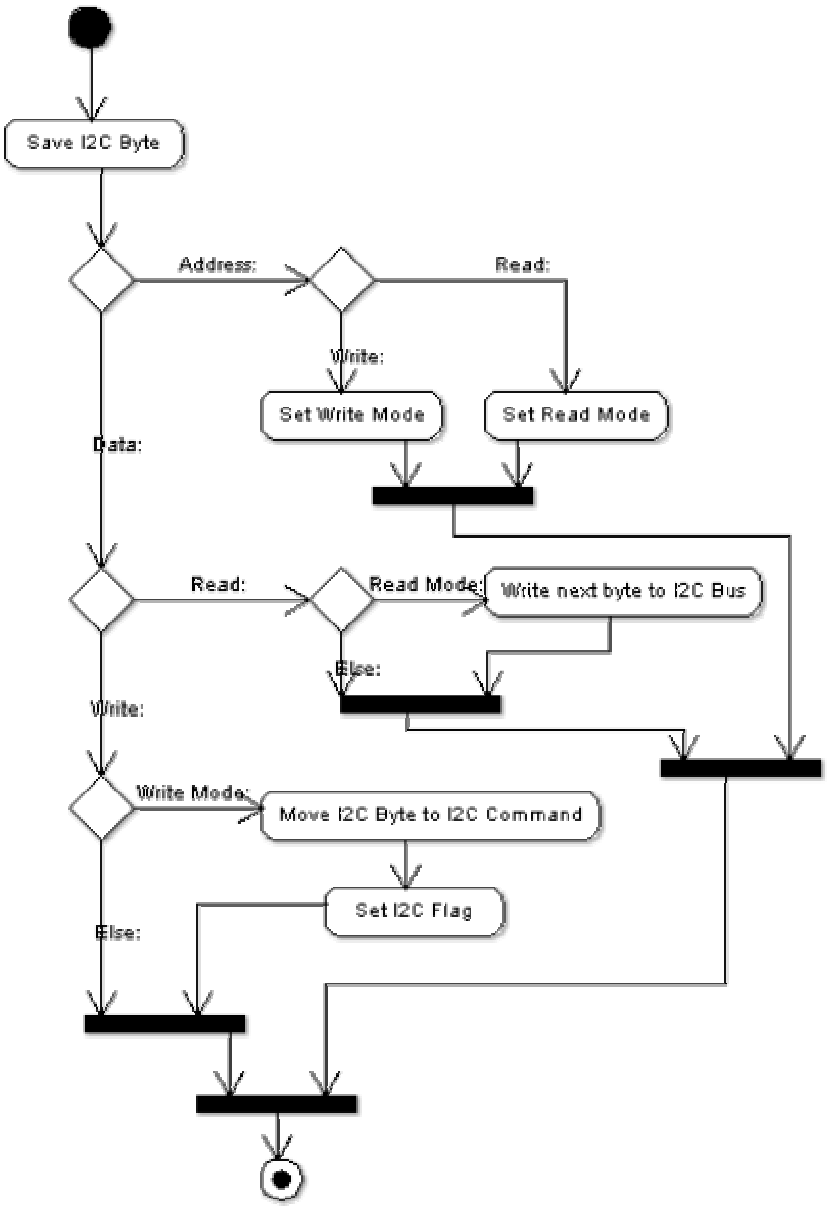

Figure 6 I2C ISR Activity Flow Diagram

The following sections will describe the different components of the web server subsystem and then show the architecture of the subsystem.

2.2.1 Web server Modules

The web server runs five different processes to perform its responsibilities.

Weather Sensors –The microcontroller subsystem was encapsulated by a module called Weather Sensors in the web server subsystem. This module implements a weather sensor interface given

to students in the design specification. These objects offer a layer of abstraction between the web server and the microcontroller subsystems and allow the web server to communicate with the microcontroller.

Weather Logger – The Weather Logger module periodically retrieves the weather data from the Weather Sensors module and sends it to the Weather Data Store where all the weather data is stored.

Weather Data Store – The Weather Data Store module handles the storing of all the weather data. Any module that needs to save or retrieve weather data must use this module.

Weather Servelet – The weather servelets dynamically generates a web page that contains the current weather data from the Weather Data Store module.

Data Uploader – The data uploader module is used by the Weather Data Store when the amount of data on the web server gets too large for the web server to store. The data uploader will attempt to send the archived weather data to a larger (more traditional) web server on the LAN. The data uploader will continue to store weather data and attempt to upload data until it is successful.

2.2.2 Web Server Subsystem Architecture

The web server subsystem architecture is broken down into the shared data style and a hybrid of the uses and generalization styles. The shared data style shows how the system has one main location for data that is shared among the rest of the modules. We've created the hybrid style to show how the different web server modules implement use, inheritance and implementation. Figure 7 and Figure 8 show the notation used for the shared data style and our hybrid style, respectively.

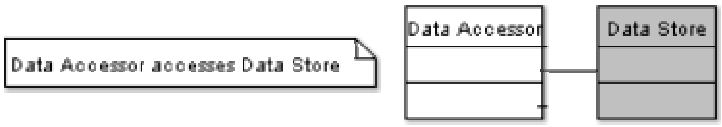

Figure 7 Shared Data Style Notation

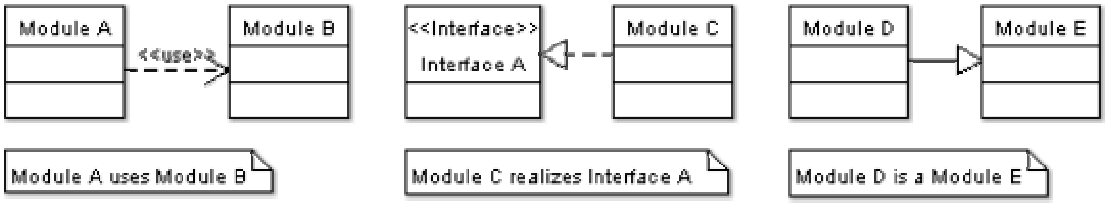

Figure 8 Hybrid Style Notation

Figure 9 shows the shared data diagram representation for the weather station web server subsystem. As is easily seen in Figure 9, all data accesses and saves are done through a common point, the Weather Data store.

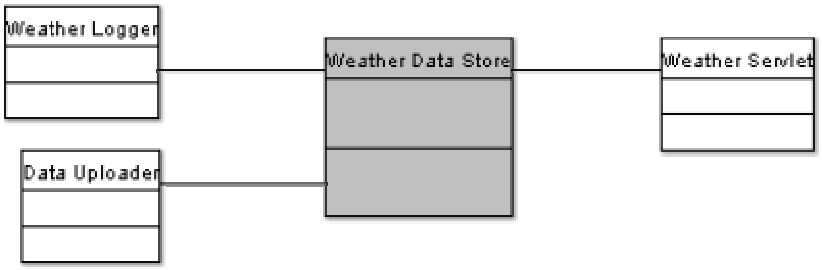

Figure 9 Shared Data Diagram for web server

Figure 10 shows the hybrid style diagram for the web server subsystem. The weather logger and HTML server+weather servelet run concurrently and both access weather data via the weather data store. The weather logger periodically uses the weather sensor objects to query the microcontroller subsystem and gather real-time weather data. The HTML server and the associated weather servlet are running and waiting for HTTP requests for web page weather information from users anywhere in the world. Upon a HTTP request via the system's URL, the weather servlet obtains the latest weather data samples from the weather data store to place dynamically into the weather page. Finally, the weather data store itself senses when its logged weather data has grown to a point where it should be upload to a more commodious server (the MSU ECE department web server). The data uploader performs this action. Because of network traffic or service, this request may require repeated tries. The weather data store continues storing weather data until the data uploader reports a successful data transfer.

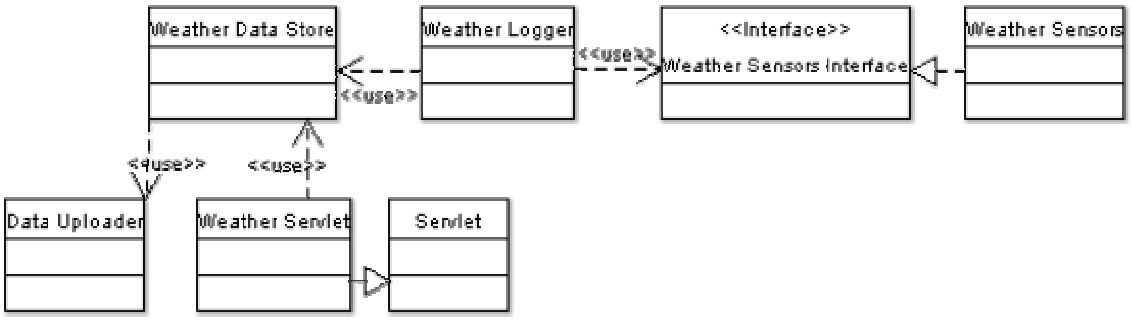

Figure 10 Hybrid Diagram for web server

3 Lab Structure

The lab component of the course is progressive, where each lab builds upon the successes of the previous labs. Teams worked in instructor-selected groups of three to four. Labs are one to two weeks in duration depending on the academic holiday calendar and lab complexity. This section briefly describes each lab turn.

3.1 Lab 1

The objective of the first lab is to get the main software tools and components that the students would be using installed. The software includes Java SDK, additional Java APIs, Web Server Firmware and development tools and libraries, and Eclipse Integrated Development Environment (IDE) with Concurrent Version Systems (CVS) revision control software and Ant (a platform-independent make utility based on Java) included. After the tools are installed, students work through tutorials and perform simple program builds to provide a firm foundation of the tools.

3.2 Lab 2

The objective of the second lab is to setup the web server hardware and to get a simple program compiled and running on the web server. The setup of the hardware includes: installing the latest web server firmware, intern et setup, user account setup. The students learn how to compile the code manually in order for them to get a good understanding of the tools they would be using in the future.

3.3 Lab 3

The objectives of the third lab are to introduce the students to some of the useful features of the IDE that they would be using, HTTP client capabilities of the web server, and data logging capabilities of the web server. The students learn how to use refactoring to more easily make global changes to their projects and to use the built-in Ant capabilities of Eclipse to compile and upload their project to the web server. The HTTP client classes that they develop start with a simple HTTP client that downloaded a web page's source and echoed it to the console. They then modify this object to download a page from National Oceanographic and Atmospheric Administration (NOAA) and parse the weather data from it that would be logged by another object. These objects are the foundations for objects developed in future labs.

3.4 Lab 4

The objectives of Lab 4 are to introduce to revision and control software, use of the I^2C bus by the web server (as master), and Java encapsulation. CVS allows the students to develop software as a group and to track of software revisions. The students learn how to use the web server as an I^2C master to communicate with dedicated component I^2C slaves (I^2C temperature sensor and EEPROM memory) through a sensor interface that they would soon design. Encapsulation is introduced to show the ability to encapsulate sensor peripherals into a Java class.

3.5 Lab 5

Lab 5 moves from the abstract levels of the web server to the low-level of the microcontroller and the physical sensor interface. The objectives of this lab are to setup the microcontroller and to get it to read one of the sensors. The students would be setting up the hardware that would later be interfaced with the web server. Implementing the temperature sensor reading gave the students a chance to begin setup up the software structure of the microcontroller sensor interface. (This and later microcontroller-based lab tasks move very rapidly because of the students' familiarity with the microcontroller and the development tools from the junior-level microprocessor course.)

3.6 Lab 6

The objective of Lab 6 is to implement and read the remaining weather sensors. Although seeming simple, the intricate timing and signal conditioning issues require considerable student effort to get all the sensors values functioning simultaneously. Upon the completion of Lab 6, the microcontroller sensor interface is very nearly complete and ready to communicate with the web server in a similar manner as the dedicated I^2C components in Lab 4.

3.7 Lab 7

Lab 7 requires the students to develop the microcontroller into an I2C standard conforming slaved device that communicates with the web server via the specified protocol. Microcontroller interface circuit is transferred from solderless breadboard to a professionally fabricated two-layer printed circuit board with soldermask and silkscreen. Upon the completion of Lab 7, the microcontroller sensor interface design is complete.

3.8 Lab 8

Starting in Lab 8, many of the skills and results of the previous seven labs come together to create the fully functioning system. The objectives of Lab 8 are to (i) install the web server software (Tynamo), (ii) encapsulate the microcontroller sensor interface, (iii) display current weather conditions via a dynamically generated web page, and (iv) log weather data.

Students use their Ant skills to build and upload the open source web server software Tynamo. They use their encapsulation knowledge to design a Java interface and implementation for the microcontroller sensor interface. The weather logger is modified from its Lab 3 form to use the new interface. Students generate dynamic web content with Java Servlets running with Tynamo.

3.9 Lab 9

The objectives of the final lab are to improve software functionality to handle unconnected sensors and data uploading to the “big” ECE department server. The students use the HTTP client class from lab 3 to get data from the Internet in the case of unconnected or missing sensors. They also design a class to upload data logs to an external server. After this lab, the weather station projects are complete. Student then created PHP scripts on the large department server to graphically display historical weather data from the archives upload by the “small” internet-capable embedded system

4 Conclusions

In order to meet the need for embedded systems engineers, faculty at MSU has created an embedded systems design course that requires students to build an internet-capable embedded system under conditions that emulate industry as much as possible. The student teams build a progressively more complex design using formal and documented design reviews and collect product measures to monitor their design performance. The design uses two different processors, each being used where it is most appropriate. The web server and the microcontroller together produced results that neither could have by themselves. The web server handles course data processing/logging and dynamic web page generation while the microcontroller tends to the

weather sensors' low-level interface. Development is done predominately in C (on the microcontroller) and Java (for the web server). Students must use skills and knowledge from at least a half-dozen prerequisite courses, while working on multi-disciplinary teams including CPE, EEs, SE, and CS majors.

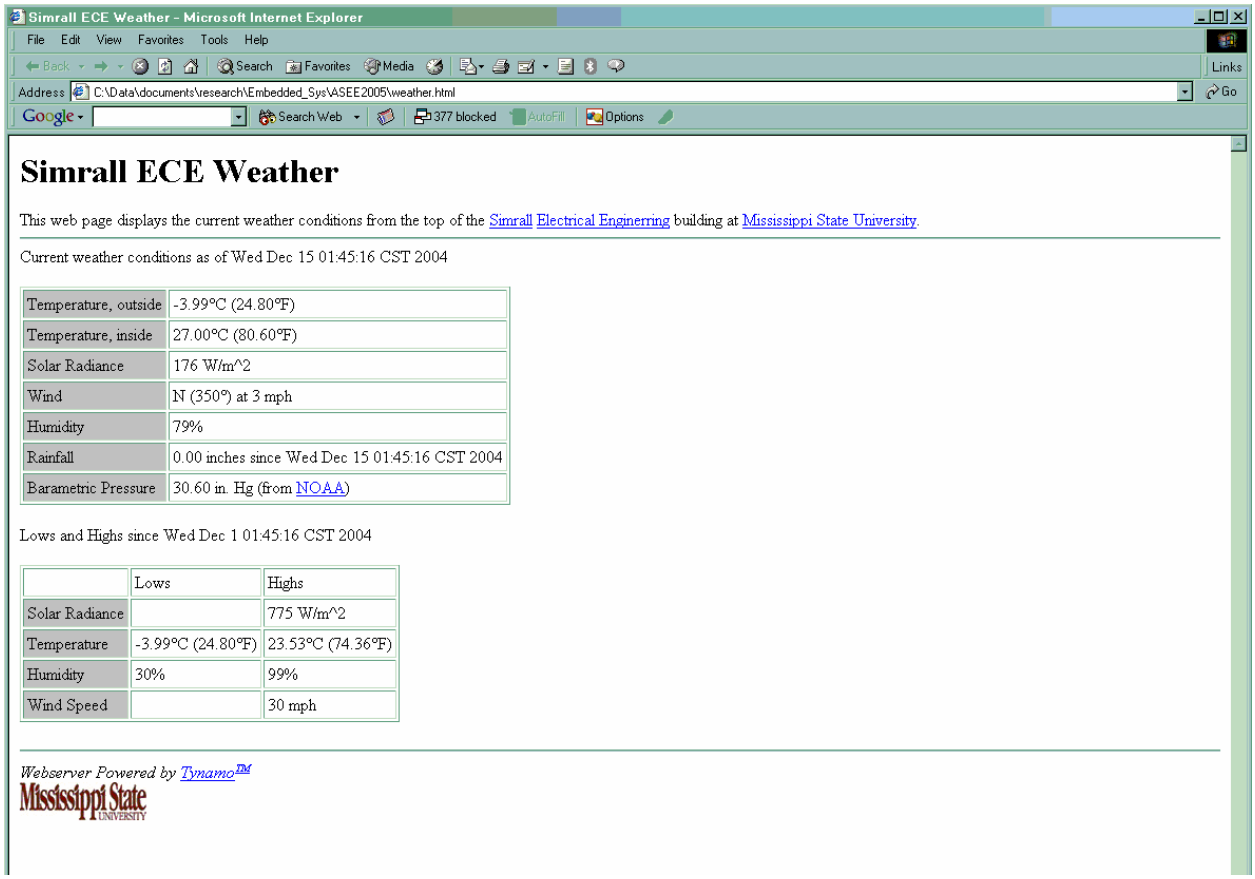

Figure 11 Dynamic web page served from internet-capable embedded system

The course described here has been offered once (Fall 2004) so statistically significant data is not yet available. Anecdotal evidence suggests that the very visible and broad extents of the project engaged students more than traditional "concept" oriented lab tasks. Furthermore, students perceived an increased relevance of their education. Quantitative assessment data will be collected in subsequent semesters and the findings will be reported at ASEE conferences in the coming years.

References

[1] J.W. Bruce, J.C. Harden, and R.B. Reese, "Cooperative and progressive design experience for embedded systems," *IEEE Trans. Educ.* vol. 47, no. 1, pp. 83-92, 2004.

[2] JW. Bruce, "Design Inspections and Software Product Metrics in an Embedded Systems Design Course", *Proc. ASEE Annual Meeting and Exposition*, 2004.

[3] IEEE Std. 1028-1997, IEEE Standard on Software Reviews. Section 6.

[4] T. Gilb and D. Graham, Software Inspections. Addison-Wesley, 1993.

[5] IEEE Std. 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.
[6] R.S. Pressman, *Software Engineering: A Practitioner's Approach 5/e*, Mc-Graw Hill, 2001.
[7] Philips Semiconductors, I2C specification version 2.1, Jan. 2000. [Available at http://www.semiconductors.philips.com/markets/mms/protocols/i2c/]
[8] D. Loomis, *TINI Specification and Developer's Guide*, Addison-Wesley, 2001. [Available at http://www.maxim-ic.com/products/tini/devguide.cfm]

Biographical Information

J.W. Bruce is with the Department of Electrical and Computer Engineering at Mississippi State University, where he is an Assistant Professor. Dr. Bruce teaches courses on embedded systems, VLSI, and systems-on-a-chip design and was named the Bagley College of Engineering Outstanding Engineering Educator in 2003. Dr. Bruce researches data converter architectures and embedded systems design.

Jordan Goulder received the B.S. degree from Mississippi State University in 2003, and expects to complete the M.S. degree at Mississippi State University in 2005; both in Computer Engineering Mr. Goulder research interests are embedded systems design.