CPSC 483 – Computer System Design

Coff-e-mail Final Report

May 10, 2004

Don McGee
Eric Peden
Payton Quackenbush
Zack Roman

Table of Contents

Introduction

The Computer Science Department of Texas A&M University needs a convenient way to keep tabs on their faculty lounge coffee maker. The coffee maker is a popular device among the department, shared by so many people that it is currently inconvenient to get coffee. There is no way to know how old the coffee is, and remotely, no way to know whether or how much coffee is left in the machine, or when someone has just brewed a fresh pot. It would also be convenient to generate usage statistics of the machine, and be able to remotely notify coffee drinkers that fresh coffee is ready, or that the machine is empty and needs a new pot to be brewed. These statistics could also allow for justification of a secondary or more powerful coffee maker.

The Coff-e-mail group has developed a solution for this problem. We have built an embedded system that will monitor the coffee maker through various sensors to determine when coffee is brewing (or not brewing), when someone is drawing coffee, and approximately how much coffee is left in the machine. All of the information collected is logged and analyzed, generating data and graphical statistics available on a web server. The system also includes an email notification utility to notify members of the list when the pot is full and when the pot is empty.

Problem Background

The general scope of this project is computer engineering research, encompassing embedded systems, sensor interfacing, computer networking, human-computer interaction, and web programming. More specifically, the background is developing a system to perform device monitoring, collect statistics, perform remote notification, and produce different representations of statistical data to users in easily viewable forms. A specific area that was addressed was determining the level of a liquid in a clear site tube using matched infrared emitters and detectors.

Needs Statement

The coffee maker in the third floor break room of the Bright Building did not have an easy way of monitoring the status of its coffee: when was it last brewed, how much coffee is left, or how long the coffee will last. Previously, the only way to determine the amount of coffee left in the maker was to physically walk to the coffee maker and look at the site

glass on the front of the machine. This may not seem like it is that big of a task, but if you are on the fifth floor of the building, the task can be fairly time consuming. There was a need to have a means of remotely monitoring the coffee maker, and being notified of various status events that occur with the coffee maker. This required a monitoring system to determine the level of coffee left in the maker, a way of estimating the amount of coffee that a person takes from the machine, and a camera that takes pictures of the users' coffee mugs as they are dispensing coffee from the machine.

Goal

The goal of Coff-e-mail is to develop an embedded system to monitor various aspects of a coffee maker, and perform remote notification through email of events captured.

Objectives

The objectives of Coff-e-mail include:

- The ability to remotely monitor coffee consumption online, including a statistical history. Graphs should be generated to display usage, along with daily, weekly, and monthly aggregates. The areas to be tracked are cups of coffee poured, times coffee is brewed, and the level of coffee.
- Notification of fresh coffee (the brewing), through emails, to coffee drinkers.
- Snapshots of the latest users' mugs, and a history of them.
- Low-cost final prototype.

Literature Survey

Web-enabled devices have typically been thought of as "hobbyist" projects, and as a result most of the technical literature deals with developing new protocols and hardware for "smart appliances" and data acquisition devices (DAQs) rather than hooking web servers up to existing devices. [Che02] and [Sch01] deal with smart appliances; both articles discuss communication protocol issues. Because of our need for a web server, we chose instead to use TCP/IP for communication rather than a custom protocol, but [Sch01] provided a number of useful guidelines for building custom protocols for devices that must stay out of the users' way. [Spa99] discusses an autonomous data-collection device that must deal with event-logging issues we also expect to encounter, such as time-stamping and on-the-fly summaries of collected data.

[Oul01] details the development of a web server-on-a-chip. The product, Webchip, was one we were not aware of; after reading this paper and looking at the chip's documentation, it looked promising. [Yea03] discusses the development of a power-monitoring device. The device captures a variety of data points (temperature, voltage, current) and logs them on-board. Web pages are created by a CGI which runs on a dedicated web server. Again, this is not the approach we decided to use, but sample Java code further demonstrates logging techniques. Also, although the paper does not say what hardware was used, analysis of one of the screenshots suggests that the TINI controller provided network services to the device. Finally, [Wit98] provides an in-depth analysis of an old (circa 1998) embedded web server with attached web cam. It provides dated but real-world example of how to interface with a camera and alternate methods for creating dynamic web pages.

Design Constraints

The design constraints for the Coff-e-mail are:

- Building a monitoring system for the coffee maker that does not require any permanent modifications to the machine in any way.
- Tracking useful information from the coffee maker, including the estimated coffee level and the amount of coffee taken by users.
- Creating a cost effective design. It would not be efficient to have a very expensive machine to monitor the department's coffee maker.
- Easily integrating into the department's computer network. It is desired for the system to communicate wirelessly with the network, but this proved prohibitively expensive. It was decided instead that the system could have a wired connection, but it would have to be as unobtrusive as possible. We identified an Ethernet to 802.11 bridge that will allow "after market" addition of wireless access, should the department (or the consumer) be willing to accept the additional cost.

Alternative Solutions Considered

For our project there were a couple of alternative solutions that we could have taken for different parts of the project.

Wired/Wireless

The first decision that we had to make was whether we could implement our project using wireless hardware or not. It would be nice for the project to be able to host the webpage and collect the statistics with a wireless microcontroller. This would alleviate the problem of having to run wires and having new Ethernet ports added. The main problem with using wireless communication is that in order to work with the computer science departments' system, we would have to use a VPN protocol. In order to do this we would have to write our own client for the microcontroller we were going to use. Another problem with the wireless microcontrollers is that they are almost twice as expensive as their wired counterparts.

The wired solution had the benefit that the microcontroller was a lot cheaper. The wired microcontroller would also be much easier to integrate into the department's computer system because there is not a complicated protocol like VPN that would have to be used. The disadvantage of using the wired microcontroller is that a cord would have to be run to connect the system to the network. There is not currently a Ethernet port in the room with the coffee maker. The nearest port is in the room next to the coffee maker.

Due to the fact that the VPN protocol looked like it would be very complicated to implement on a wireless microcontroller, we decided to go with the wired route. We talked with Nolan Flowers and were told that we could run a cord to the nearest network port in the next room temporarily while we are testing our project. If the project is a success, a more permanent solution (such as adding a new network port next to the coffee maker) would be implemented.

Microcontroller

There were a few different microcontrollers we could have gone with. We already knew we wanted a wired microcontroller for the reasons discussed above. After deciding to go the wired route, we still had a few choices we needed to narrow down:

Table 1: Microcontrollers

	TINI390 1 MB	SitePlayer	SNAP TiltKit
RAM	1 MB SRAM (non-volatile)	512 B	8 MB DRAM
Flash Available	64 KB	32 KB	2 Mb
Programming Interface	Java 1.1.8, 8051 assembly	C, 8051 assembly	Java J2ME-CLDC
Cost	$134.00	$99.00	$167.00
Web Site	http://www.ibutton.com/TINI/hardware	http://www.siteplayer.com/	http://www.imsys.se/products/prodsnap.shtml

We decided to use the SNAP TiltKit for our microcontroller. The SitePlayer board was cheap, but it also did not have very much memory. The TINI390 board had more memory and it was also still fairly cheap. The problem we had with the TINI390 was that it was a little on the slow side because it simulated Java. We came up with one last microcontroller, the SNAP TiltKit. The SNAP TiltKit had plenty of memory and it is approximately 20 times faster than the TINI390 because it implements Java on a custom processor. The biggest problem with the SNAP TiltKit is that it is produced in Sweden and there are no American vendors of the product. However, we emailed the manufacturer of the SNAP TiltKit and we were assured that it would be no problem getting a microcontroller in a timely manner.

In the end we decided to go with the SNAP TiltKit. We liked this microcontroller because it is faster and it does not cost that much more than the other microcontrollers. The extra speed of the SNAP TiltKit will come in very useful for working with the camera data. We were also assured that we could get this microcontroller quickly, which is a must.

Camera

There were a few different cameras we could have gone with. Some of the different cameras we looked at are:

Table 2: Cameras

	CMUcam	VGA00AIT1	GameBoy Camera/ M64282FP
Operation Voltage	5VDC	?	5.0V
Operation Power	200mW(Active)	?	15mW
Video Output	Digital 8 bit	8 bit (JPEG Encoding)	Analog
Lens	OV6620, CMOS image sensor	VGA size CMOS image sensor	Mitsubishi M64282FP CMOS Image Sensor
Color/B&W	Color	Color	B&W
Resolution/Pixels	80x143	640x480 or 80x60	128x128
Vendor	seattlerobotics.com	usbdeveloper.com	Used Part
Price	$55	$50	~$20

The GameBoy Camera was a very cheap solution; the main problem was that it only has an analog interface. This analog signal would have to be converted to a digital signal in order to work with our system. Another problem with the GameBoy camera was that it is only a B&W Camera. This was not really that big of a problem because the images we are taking are only of coffee mugs.

The VGA00AIT1 looked to be a very useful camera because it could take low resolution images and convert them into a JPEG format. Converting the images into JPEG format would be very helpful because that is one of the hardest parts of integrating a camera into our system. However, the website where we found this camera looked a little sloppy, like maybe this was not a legit dealer. There was not a datasheet or much

technical information available for this camera. We emailed the company using the e-mail address on their website and we never got a reply.

The CMUCam was a well documented camera. Information was available in the form of datasheets which would make working with the camera much easier. The camera was fairly cheap and it has the ability to take fairly low resolution images. The only problem would be working with the data from the camera and converting the data into a usable format such as JPEG.

We decided to go with the CMUCam because it was the best documented and it had a digital interface. It would have been nice to use the VGA00AIT1, but we never could get any information about the camera so we decided that the company that makes the camera must not be legit. So we decided to go with the CMUCam because we knew it was readily available and it was very well documented. So we went ahead and ordered the CMUCam.

Ordering the CMUCam turned out to be a mistake. Working with the CMUCam turned out to be much harder than we had originally anticipated. The CMUCam has an internal clock signal that must be synchronized with in order to read the data from the camera. While this task is not too terribly hard, it was a little more than we wanted to work on, considering the time frame of the project.

After giving up on the CMUCam, we purchased a used GameBoy camera at the mall for roughly $10. This was a very cheap camera, so we could afford to play with it. After taking it apart and researching the pins on the camera, it turned out that there were only 9 pins that had to be interfaced with. It turned out that this was a fairly easy task when compared to working with the CMUCam. The GameBoy camera was fairly easy to work with and integrate into the system and it gave pretty nice images as well.

Sensor/Camera Microcontroller: We had originally thought we were going to use an FPGA to interface with the camera. The FPGA would then send the image data through the serial interface on the sensor microcontroller to the SNAP. It was hoped that the FPGA would be able to read the data and convert the raw image data into a compressed format. However, after working with the cameras some, it was determine that it would be just as easy to interface the camera using only the sensor microcontroller. The sensor microcontroller could read the data from the camera and then send this data to the SNAP which would then do the conversion to a compressed image format.
Another decision that had to be made was whether to use the PIC or the cygnal for the

sensor microcontroller. The PIC had the advantage of being a very cheap microcontroller. The disadvantage of the PIC was that it was hard to work with because it was slow to program and it did not have much of a debugging interface. The cygnal on the other hand programmed new coding changes very quickly and it was easy to debug the code. However the development cygnal boards that we had in the lab were fairly expensive units. In the end we decided to prototype our camera on the cygnal microcontroller and then implement this code on the PIC. This helped to speed up the development time of our project and it also reduced the cost by porting it to the PIC.

Spigot Sensor

One last main decision that had to be made for our project was how to detect when the coffee spigot was pulled. We had originally planned to use a flex resistor to determine when the spigot was pulled and to estimate how far it was pulled in order to get an idea of how much coffee was being dispensed. After more review it was determined that no matter how far the spigot was pulled, the coffee seemed to come out at the same rate. Also it was determined that mounting the flex resistor to the spigot was going to be very difficult to do in an aesthetically pleasing manner.

We also researched methods of mounting buttons to the spigot to determine when it was pulled. We were unable to come up with any really good methods for mounting the buttons in an aesthetically pleasing and permanent manner. In the end we found a reed sensor, like is typically used in alarm systems to determine when a door is opened. Reed sensors work off of a magnet field; when the magnet is close to the sensor it is a closed circuit and when the magnet is removed from the proximity of the sensor it is an open circuit. Mounting the reed sensor was very easily accomplished using two screws to mount it to the bracket directly behind the spigot lever. A magnet was then imbedded into the spigot handle. Once some epoxy was added to the spigot handle, it was barely noticeable that the handle had even been modified. The reed sensor made a very clean installation for detecting when the spigot was pulled.

Design

Server Module

Components

The main software components to the server module are shown below in **Figure 1**. The majority of these components will be explained in later sections, but one that bears special mention is the *Tynamo Web Server*. Tynamo is a free-for-non-commercial-use, pure Java, web server with Java Servlets support.

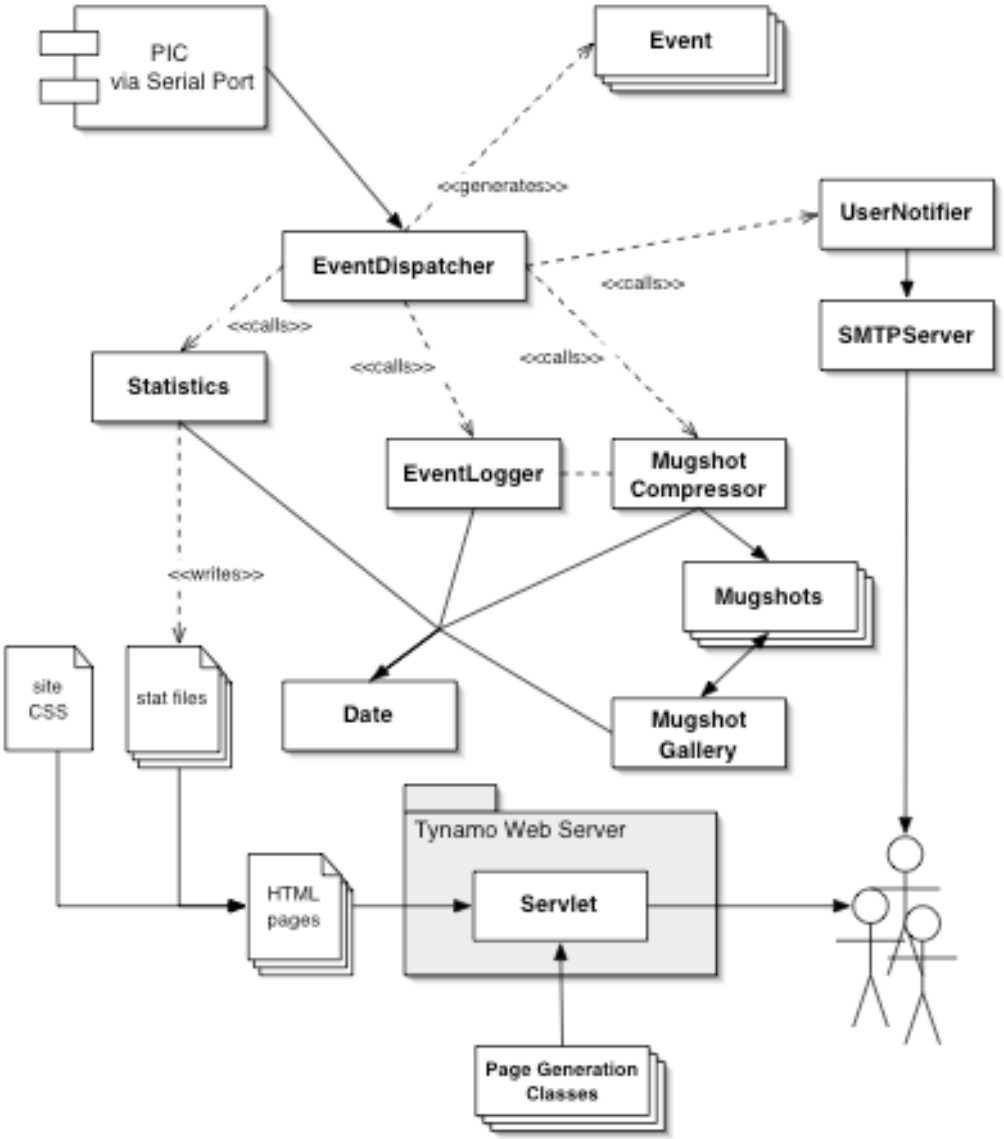

Figure 1 Server Module Component Diagram

Core API Overview

The server module is composed of several Java classes, some built by us and others provided with the SNAP, and a set of HTML pages, CSS pages, and templates. Templates contains placeholders for calculated values and are converted, for performance reasons, into actual inline Java code by a series of utility scripts that are executed as part of the build process. The Statistics module, discussed below, replaces these placeholders with dynamically calculated variables and writes them to a location where they can be included by the web pages.

The following list shows the major functions provided by the core components. Please note that this list is intended as an overview: the actual method names are sometimes given as pseudo-code and may not reflect the actual names used in the Java source. The interested reader is encouraged to review the Javadocs—which are kept updated with the source—for a full API reference.

Software Processes/Components

- **EventDispatcher** Handles incoming events. Notifies EventLogger, Statistics and UserNotifier of events as they are received.
  - ⇒ run() Begin main loop; process events as they are received.
- **EventLogger** Logs events to a text file; also capable of reading an existing text file and simulating logged sensor events.
  - ⇒ readLogFile(logfileToRead) Reads an existing log file and simulates the logged events.
  - ⇒ logEvent(event) Writes event to log file if logging is enabled; otherwise, it is ignored.
  - ⇒ setLogFile(logfilePath) Sets the path to the log file.
  - ⇒ enableLogging() Enables event logging.
  - ⇒ disableLogging() Disables event logging.
- **Event** Abstract class representing a sensor event. A subclass exists for each possible event: MachineRefilledEvent, LevelAlmostEmptyEvent, LevelEmptyEvent, CupPouredEvent.
  - ⇒ Event.eventFromData(data) When passed the data passed from the sensor module, returns the appropriate Event subclass for that data.
  - ⇒ name() Returns the name of the event.
  - ⇒ sensorID() Returns the sensor ID which generated the event
  - ⇒ sensorData() Returns the sensor data associated with the event.
- **Date** Provides calendar-based interface to times formatted as seconds since Unix epoch.
  - ⇒ Date() returns a Date object set to the current system time
  - ⇒ Date(timeMillis) returns a Date object with time set to timeMillis, where timeMillis is in seconds since Unix epoch format
  - ⇒ Date(str) returns a Date object with time set to str, where str is in “DOW, DD Mon YY HH:MM:SS TMZ” format.

    - ⇒ setCurrentTime() sets the time of this Date object to the current system time
    - ⇒ year(), month(), day(), dayOfWeek(), hour() returns the requested field of this Date as an integer. Except for hour(), all fields start at 1, not zero. hour() is given in 24-hour format.
    - ⇒ Date.isLeapYear(year) returns true if this 4 digit year is a leap year, false otherwise
    - ⇒ Date.daysInMonth(month, year) returns the number of days in given month during the given year. month is an integer in range [1, 12], and year is a 4 digit year.
- **Statistics** Maintains a slew of statistics on coffee maker usage. When statistics-changing events occur the internal stats are updated and these updated values are written to the files included by the web pages. This class also maintains the current state of the coffee pot. Thanks to the log-reading capabilities of the EventLogger, the functionality of this class can be freely changed and all statistics can be brought up-to-date with historical data by re-reading old log files.
    - ⇒ Stats maintained: consumption in cups and gallons (lifetime, this year by month, this month by day, this week by day of week, today by hour), average consumption in cups and gallons (quantity/day over lifetime, quantity/hour over lifetime), average amount of coffee per cup poured (over lifetime), current pot level as percentage of full, last brew time, estimated time to empty based on average quantity/hour, average pot lifespan (by day of week over lifetime), strength of current pot.
    - ⇒ cupPoured(sizeOfCup), potEmpty(), potRefilled() Called when corresponding event occurs; cause statistics to be updated and output files to be written.
- **UserNotifier** Responsible for notifying subscribers when important events occur. Initial implementation will send e-mail via an external SMTP server, and users will subscribe via the web site. Multiple subscription categories are maintained, with each category only receiving particular messages.
    - ⇒ CATEGORY_ALMOST_EMPTY, CATEGORY_NEEDS_REFILL, CATEGORY_FRESH_POT Possible subscriber categories.
    - ⇒ addEmailSubscriber(emailAddress, category) Adds emailAddress to the subscriber list; a subscription message will be sent to the user. If the email address is already subscribed, no action is taken.
    - ⇒ removeEmailSubscriber(emailAddress, category) Removes emailAddress from the subscriber list, if it exists; the address is ignored otherwise.
    - ⇒ notifyPotAlmostEmpty(), notifyPotEmpty(), notifyFreshPot() Sends notifications to subscribers in the corresponding category.
- **Servlet** Together with a number of page-generation classes, this class sits between our core code and the web server and funnels data to the outside world. All web site performance optimizations are handled here: it provides in-memory caching of files and images read from Flash or other parts of the system, manages execution of the event dispatcher thread, and implements a “pseudo-filesystem” which URLs map to.
- **MugShot** Maintains mug shots. Mug shots are stored as GIF compressed image; because of Flash space constraints, these image are stored in volatile RAM. A **MugShotCompressor** thread maintains a queue of incoming raw images and compresses them one at a time. This allows us to fetch an image from the PIC in a high-priority thread for rapid data-transfer, but to perform the actual, CPU intensive compression in a lower priority thread to preserve the responsiveness of the system. Currently, a hard limit to the number of captured mugshots has been

set. When the number of images captured exceeds this value, mugshots begin to expire, oldest first.

- ⇒ MugShots.recentMugShots() Returns an array of the most recent Mugshots
- ⇒ MugShots.lastMugShot() Returns the last Mugshot taken
- ⇒ Mugshot(jpegData) Stores jpegData as a JPEG file timestamped with the current system time
- ⇒ filename() the name of the file this Mugshot is stored in
- ⇒ fullPath() the full path to this Mugshot
- ⇒ timeTaken() the time this Mugshot was taken as a Date object

Internal/External Interfaces

The functional requirements for the server module and the testing procedure for each requirement are as follows:

External Interfaces

- The server module must receive event notifications from the sensor module according to the protocol given in the sensor module specification. This interface will be tested through a command-line program which will accept the name of an existing log file. This log file will be parsed and each logged event will be sent to through the server's event dispatch routine using the same protocol as the sensor module. This effectively simulates sensor events and will allow us to quickly test all possible events by feeding in a test suite log file. Additionally, a command-line interface will be provided to permit interactive simulation of events.

Input/Output

- The sensor will create a number of output files: the event log, the web site access log, and the statistics files. After the event dispatching code mentioned above has been tested, the command-line event simulator will be used to verify that the event log and the statistic files are generated correctly. The web site access log is created automatically by vendor-supplied classes; it will be reviewed for accuracy, but no formal verification process is planned.
- The server must be able to accept raw images from the sensor module, compress them, and serve them when requested from a browser. This functionality will be tested once the entire system is assembled; camera capture events will be triggered and we will verify that images are stored and served correctly from the server.

Internal Interfaces

- The statistics module must be able to handle "rollover," the resetting of statistics when a new day, week, month, or year begins. This is difficult to test since these statistics use the current system time, and it is impractical to set a host PC system clock to the wide range of values needed to accurately test this functionality. This feature can be tested on a host PC or the SNAP itself thanks to a very flexible date-generation architecture: simulated events can either take their timestamp from the system clock or from an arbitrary date string. Thus, we can feed events

to the SNAP with times that will generate rollovers, and then review the resulting output for accuracy.

- The date module must be able to convert times given in seconds since the Unix epoch to traditional date. A test suite asks the date module to convert a range of times from a string to the since-epoch format and back. The resulting value is then compared to the original to verify accuracy.

Sensor Module

The sensor module is the interface between the coffee maker and the web server. It takes readings from the coffee maker, including coffee level, brewing status, lever pulling to get coffee, and it takes pictures of the user as they get coffee.

The sensor module consists of three IR phototransistor/emitter pairs, a Reed sensor, a relay, the Game Boy Camera, and a PIC 16F877. The IR phototransistor pairs are used to determine the level of the coffee maker. They are mounted on the coffee maker's sight gauge, and when coffee or water is present, their values on the Analog to Digital Converter change.

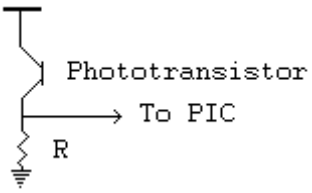

Figure 2 - Phototransistor circuit

The Relay is used to determine if coffee is brewing. We have attached the relay in parallel with the water pump inside the coffee maker. When coffee is brewing the pump is turned on for the entire brew duration. The pump, which runs at 110V AC will trip the relay, and complete a 5V DC circuit going to the PIC.

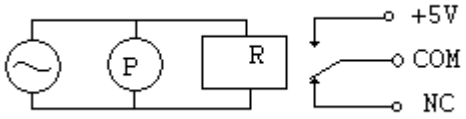

Figure 3 - Relay circuit

The Reed sensor is a device commonly seen in alarm systems. When a magnet is present, the internal circuit is closed. When no magnetic field is present the internal circuit is open. To monitor when a user pulls the lever we have hollowed out the handle on the lever and placed a bar magnet inside of it. The Reed sensor to detect the magnet is mounted on the lever guard, and is right up against the top of the lever handle when the lever is at rest. Afterwards, we filled in the lever handle with epoxy, sanded it down to match the contours of the handle, and painted it black.

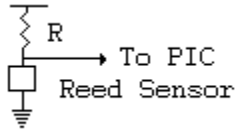

Figure 4 - Reed sensor circuit

Components

Software

The software on the PIC is implemented as a state machine. After initialization of all the hardware, the snap polls the lever for a pull, and periodically checks the coffee level, brewing status, and the SNAP. When a lever pull is detected the PIC initializes the GB Camera and takes a picture. When it detected that the lever has been released, the pull time is sent to the SNAP and the coffee level is checked. Periodically the PIC will query the SNAP for an updated message to display, checks the coffee level, and checks to see if the relay has been triggered.

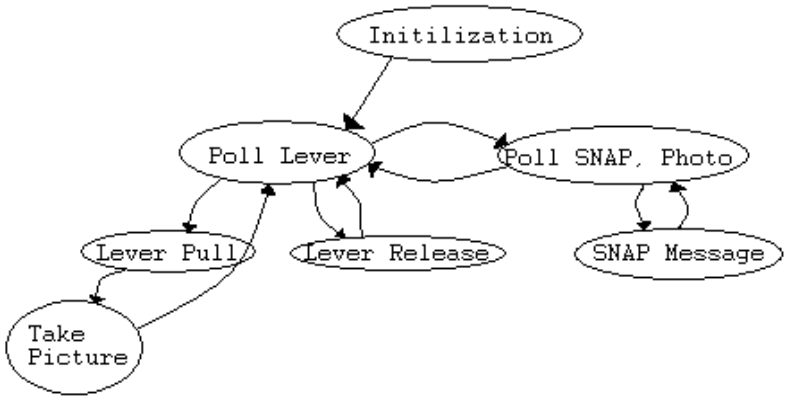

Figure 5 - Sensor module state diagram

GB Camera

The Game Boy Camera is a small 9-pin grayscale camera. When the GB Cam takes an image the image is stored in its onboard DRAM in a raw format. Because of the dynamic ram, the image must be quickly read or it will be lost. After exposure, the PIC immediately begins reading in the data on the analog out pin. The PIC will convert the voltage on the pin to a discrete 8-bit pixel value and then transmit that byte to the SNAP via serial at 115200 Baud.

Communication with the SNAP

All communication with the snap is handled by serial. For each data transfer there is a 5-way handshake between the PIC and the SNAP, followed by a data header, and finally the data. There are four types of data sent: Image data (with additional headers for RGB or B/W, image width, height, etc), Boolean Events (coffee brewing, coffee brewing complete), Analog Data (Lever pull time), and Snap Messages (20 byte character array).

Error Handling

Most actions on the PIC do not cause any errors. For those few actions that do, namely serial communication, errors are handled gracefully by timeouts, and normal operation then continues. Additionally, for uncaught errors, the Watchdog timer has been enabled on the PIC with approximately a 3.5 second timer. If the PIC were to lockup, the watchdog timer would reset the PIC and it will go back to polling the sensors in less than 10 seconds.

Unused Sensors

We originally had plans for additional types of sensors, as well as different ways of measuring stats using sensors other than those in the final design.

Flex Resistor

We originally planned to use a FLEX resistor to measure the lever pull.
This was scrapped near the end of the project for two reasons:
1) We were unable to find a way to mount the flex sensor in an aesthetically pleasing way as well as any way to minimize the wires attached to it.
2) The original purpose of the FLEX resistor was to determine the angle of the pull, and therefore extrapolate the amount of coffee the user just received. In watching people use the coffee maker, the user almost always pulled the lever as far as it would go.
Because of these two reasons we decided to use a digital button or other measuring device instead, just to detect if the lever had been pulled or not. In software on the SNAP calculations are done to determine the amount of coffee based on the current level of the coffee and the pull time.

Capacitance Probe

One idea for measuring the level of the coffee was a capacitance probe. The capacitance probe would be a copper wire inserted into the coffee tank, and after some sensor processing on the capacitance between the wire and the coffee, the coffee level could be determined. We did not use this idea because of the difficulty in implementing the sensor, as well as safety considerations of having anything come into contact with the coffee

pH sensor

Another planned sensor was a pH sensor. This idea was scrapped because of worry over possible coffee contamination. Additionally, almost all pH sensors are made of glass, and were the sensor to break, we did not want users drinking broken glass.

FPGA for Image Compression

Although not a sensor, there were initially some concerns that compressing the raw image to a gif, jpeg, or png would take too much processing power. Our solution, therefore, was to build a hardware jpg compressor out of an FPGA. It turned out later,

that after a lot of optimization and thread prioritization on the SNAP that the image could be compressed in software in a reasonable amount of time and processing power.

Dedicated Analog/Digital Converter

Originally we had planned for a separate, dedicated AD converter chip, of higher quality than that on the PIC. After doing further reading on the spec. sheet, however, it turned out that using the AD chip would have been overly complicated. At the same time, we discovered that the AD converter on the PIC would be acceptable, and so the AD chip was scrapped to save pin count.

Parts List

Item	Description	Cost
SNAP TiltKit	Java-based microcontroller with on-board 10/100BaseT ethernet	$167.00
GameBoy Camera	Web cam	$15.00
Jameco 176882	PIC 16F877 Microcontroller	$9.49
LCD	20 x 4 character-based LCD for debugging and system status	$33.00
RadioShack 276-142	Infra-red LED/photo-transistors (Quantity: 5)	$14.95
Adaptaplug-K	Power Adapter	$4.99
15 Pin Female DSUB	Sensor Connector	$1.99
9 Pin Female DSUB	Camera Connector	$1.49
15 Pin Male DSUB	Sensor Connector	$1.69
9 Pin Male DSUB	Camera Connector	$1.49
LED's	LED lights for network status	$2.29
Silicone Sealer	Sealer to keep water out of electrical connections	$3.19
22 Gauge Wire	Wire for sensors/camera	$5.19
Rectangular Magnet	Magnet for mounting camera	$2.49
PCB Standoffs	Mounting PIC	$1.49
SPST Slide Switch	Power Switch	$2.29
Adaptaplug-N	Power Adapter	$4.99
Project Box	Project enclosure (8"x6"x3")	$6.99
_ PVC Strap	Mounts for phototransistors	$0.70
Zip Ties	Zip Ties for securing wiring	$1.25
Screws/Nuts	Fasteners for mounting components	$0.83
Tubing	Plastic tubing for mounting phototransistors	$3.50
D-ScrewLocks	Standoffs for SNAP and DSUB connectors	$5.00
Wire Loom	Split convoluted tubing, for hiding wires	$4.30
Cat5e Female End	Connector for sensor module wiring	$5.49
Heat Shrink	Heat shrink tubing	$4.00
Power switch		$1.00
DS232A	Dallas Semiconductor serial/TTL converter	$3.00
Various resistors		
Various potentiometers		
Various capacitors		
Reed Switch and	Switch for spigot monitoring	$4.00

magnet		
40 pin ZIF socket	Used to mount PIC microcontroller	$11.00
Protoboard	Protoboard to solder the PIC sensor component on	$3.00
Radioshack 125V AC relay	Relay to detect brew events	
	Shipping	$60.00
	Total	387.08

Validation/Testing Procedures

Each component of the Coff-e-mail project needs validation and testing procedures to ensure that the component works as designed. Therefore, each component will have its own validation and testing procedure, tailored to ensure its correctness.

Server Module

The functional requirements for the server module and the testing procedure for each requirement are as follows:

External Interfaces

- The server module must receive event notifications from the sensor module according to the protocol given in the sensor module specification. This interface will be tested through a command-line program which will accept the name of an existing log file. This log file will be parsed and each logged event will be sent to through the server's event dispatch routine using the same protocol as the sensor module. This effectively simulates sensor events and will allow us to quickly test all possible events by feeding in a test suite log file. Additionally, a command-line interface will be provided to permit interactive simulation of events.

Input/Output

- The sensor will create a number of output files: the event log, the web site access log, and the statistics files. After the event dispatching code mentioned above has been tested, the command-line event simulator will be used to verify that the event log and the statistic files are generated correctly. Vendor-supplied classes create the web site access log automatically; it will be reviewed for accuracy, but no formal verification process is planned.
- The server must be able to accept JPEG compressed images from the sensor module, store them, and serve them when requested from a browser. This functionality will be tested once the entire system is assembled; camera capture

events will be triggered and we will verify that images are stored and served correctly from the server.

Internal Interfaces

- The statistics module must be able to handle "rollover," the resetting of statistics when a new day, week, month, or year begins. This is difficult to test since these statistics use the current system time, and it is impractical to set a host PC system clock to the wide range of values needed to accurately test this functionality. This feature will not be formally tested until the code is run on the SNAP, at which point a test driver will be written that will set the SNAP clock to a certain value, simulate an event, set the SNAP clock to a value which will generate rollover, and then simulate another event. The resulting statistic files will then be reviewed for accuracy.
- The date module must be able to convert times given in seconds since the Unix epoch to traditional date. A test suite asks the date module to convert a range of times from a string to the since-epoch format and back. The resulting value is then compared to the original to verify accuracy.

Sensor Module

The functional requirements for the sensor module and the testing procedure for each requirement are as follows:

External Interfaces

- External interfaces for the sensor module consists of the two-way connection with the PIC, the one-way connection with the sever board, and the one-way connection to the LCD display. The LCD will be used as the primary testing tool for the PIC. Later the LCD will be attached to the coffee maker and used to display messages for its users. The PIC will transmit raw pixels serially to the server module. The PIC will not do any additional processing of an image, other than a cursory check to make sure all the data is there. All data passed to the server board will be checked and processed on the server module.

Input/Output

- The connection to the server board will consist of one interrupt line and a serial connection. The PIC will use the interrupt line to notify the server that it has data. The server will then request the data from the PIC over the serial, and then the PIC will send the data according the protocol given in the sensor module specification.
- Connections with the server board will be tested on the server side through a little bit of programming and then outputting data on standard out.

- The connection to the LCD display consists of four data lines and two control lines. ASCII characters will be output on the LCD by using the specification listed for using the HD4470 LCD controller.
- The connection to the LCD will be tested after writing a small driver for it. Testing will consist of sample text strings, but hard coded, and generated on the fly. The LCD driver will be tweaked as necessary. Most changes are expected to be due to timing problems.

Internal Interfaces

- The only internal interface of the sensor module is between the sensors, analogue/digital converter, and the PIC. Connections between the sensors and A/D Converter will be tested with a multi-meter. Data from the A/D Converter will be output on the LCD by the PIC.

Schedule and Deliverables

The final product consists of the following deliverables:

- This document and accompanying source code documentation
- Registered domain name, 'coffee.cs.tamu.edu'
- SNAP board/PIC sensor module board in housing
- Camera in housing with magnetic mounts and flexible cord
- LCD display surface-mounted on board housing
- Phototransistor module mounted to coffee maker
- Reed sensor mounted inside of replacement spigot level
- Sensor bundle cable with easily-detached connector to facilitate coffee maker cleaning
- Brew sensor relay connected to coffee maker water pump

Division of Labor/Responsibilities

Coff-e-mail was divided almost equally between our four members. Each member of the team had critical responsibilities

Don McGee – Sensor mounting, enclosure design, Java software on SNAP, web page design

Eric Peden – Server module architecture, Java software on SNAP, web page design

Payton Quackenbush – Sensor module design, C software for PIC, Java software for SNAP

Zack Roman – Sensor module design, C software for PIC, sensor module hardware, sensor mounting

Economic Analysis

The prototype of Coff-e-mail that was implemented for this project cost approximately $400. This prototype price is much higher than an optimal, mass-produced product's price, as many parts could be saved on or eliminated. For example, the SNAP's motherboard and the proto-board for the PIC could both be combined into one board that could be mass produced. This would eliminate a chunk of the cost for the prototype.

Also, the prototype cost only includes the hardware used in the project. It does not include the man-hours to design and implement the prototype, which would bring the actual prototype cost to a much higher value. However, since this is mostly an up-front (non-recurring) cost, and as we are students doing free research, we can ignore it for the delivery of the product.

We estimate the production cost per unit of Coff-e-mail to be around $300, given volume discounts and improvements such as one board for the PIC and SNAP microcontrollers. As for whether this is a viable price to be sold on the open market, that is not certain. We could conduct market research to verify, but we believe that there is a niche market willing to pay $300 for a "souped up" coffee maker.

Since many of the components for our prototype come from different vendors, the long term sustainability of the project could be affected. If one of the vendors goes out of business or decides to discontinue manufacturing a component used in our design, the Coff-e-mail project would have to be revised. The product will most likely need future updates for its SNAP web server software for new features. This is not a problem, as we could provide software updates over the internet.

Our product meets all local, state, and federal regulations to ensure viable manufacturability. Our testing was done so that each manufactured unit can be tested to verify it is a working unit before shipping out to the customer. Also, our component tolerances should not be a problem, as software could be used to calibrate intolerant components like sensors.

Societal, Safety, and Environmental Analysis

Coff-e-mail has the potential to increase socialization and fraternization of people, including faculty, staff, undergraduates, and graduate students in the Bright Building. By notifying users of the availability of fresh coffee, coffee-drinkers can come together as a group, refill their coffee mugs, and discuss current issues, hobbies, or work. In short, Coff-e-mail has the potential to promote bonding and further the wellness of its users.

However, such benefits do not come with out a cost. A primary concern is of course, safety. In operating any electrical device, there is always the hazard of electrical shock. Some of this hazard is innate in the coffee maker, and beyond our control. However, we must be conscious of the hazards our modifications will make. Therefore, we have endeavored to take utmost care in protecting exposed electronics and to minimize the potential for electrical shock. All exposed electronics will be protected or covered to prevent accidental splashing from water or coffee. Additionally, in our initial designs, sensors we deemed to have too much risk, such as a probe placed into the coffee pot to determine the pH or level of the coffee, were rejected because of the potential contamination risk to the coffee. Our final design ensures that no components come into physical contact with the coffee.

The environmental impact we assume will be negligible. The coffee maker has been installed for a long enough time for everyone to have become accustomed to it. Likewise, as an appliance in an enclosed room, there are no additional effects on the outside environment. Power consumption, by virtue of this being an embedded system, will be minimal. We have spent considerable time and effort to ensure that all components in the system are powered from a single power supply, and that any relatively high-power components, such as the camera, provide a low-power standby mode which will be used whenever possible to minimize wasted energy.

Future Work

The one aspect of the final product we were most displeased with was SNAP performance. Although far speedier than its main competition, the TINI, it's certainly not a speed demon. The following are some possibilities for improving performance; implementations are left as an exercise for the reader.

- **Faster processor.** When all else fails, throw in more horsepower. The Jstik is an even faster native-Java processor; it costs over twice as much as the SNAP, but promises orders of magnitude better performance. However, since the primary bottleneck appears to be in network transmission, this might be overkill.
- **Custom web server.** Since Tynamo provides full servlet compliance, it includes a lot of features we never touch. It's possible that implementing a custom web server could provide better performance thanks to its ability to make application-specific optimizations.
- **Separate web server.** The SNAP performs well until it's asked to provide lots of data over the network. A potential speed-up that makes use of this fact would be a standalone web server. For example, a UNIX daemon could query the SNAP periodically over TCP/IP to request just the coffee statistics and any new mugshots. This daemon could then generate the pages, or communicate with a CGI, to provide the final HTML/CSS to the browser. Here, since only a single machine communicates with the SNAP, the network performance problems are reduced, and the standalone web server can "fill in" when the SNAP is busy with other work. This would offer the best performance, but it requires a PC elsewhere to enable the web site. One of our original requirements was to avoid the need for a dedicated PC; while me met this goal, performance suffered greatly.

A number of other improvements could be made, as well. The serial communication code on the PIC could stand to be re-written, the storage and handling of event logs on the SNAP could be made more compact and more robust, and there are a world of opportunities for the web site. We hope that someone will pick up the torch and continue with further development of the Coff-E-Mail system; we believe it has a lot of promise, and with a bit of work and research could even become a marketable product. All rights reserved, by the way.

Appendices

Pictures

Figure 6 – Stock coffee maker

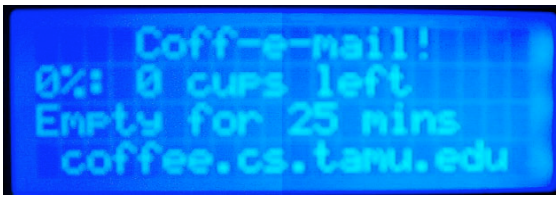

Figure 7 - Coff-e-mail status LCD

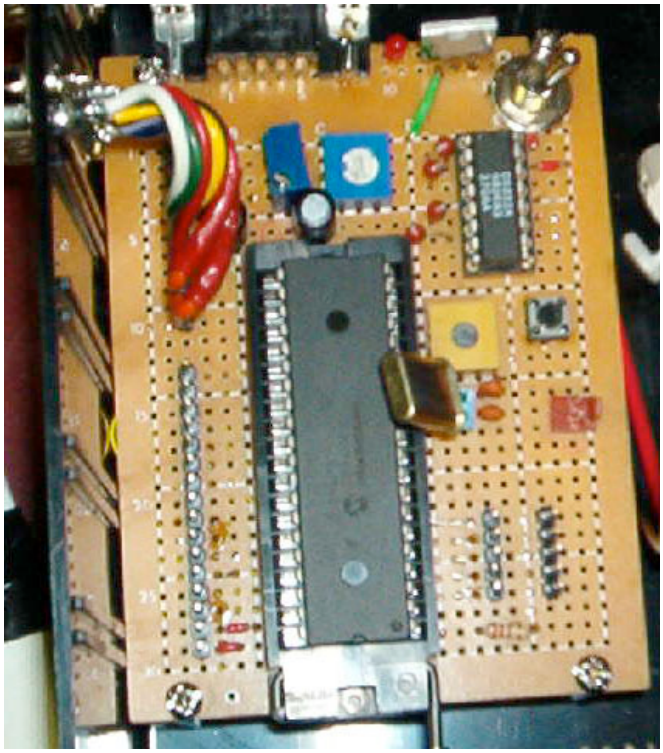

Figure 8 - Sensor module (PIC circuit)

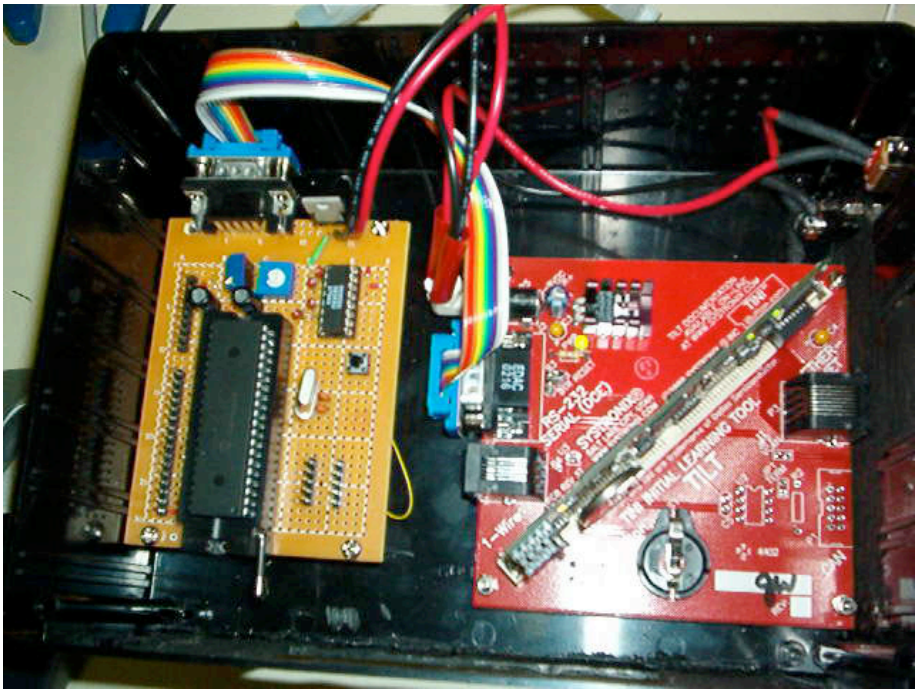

Figure 9 - Inside Coff-e-mail system (top view)

Figure 10 - Coff-e-mail in action

Figure 11 - Coff-e-mail close-up (camera on the right)

Bibliography

"A client-server architecture for distributed measurement systems." Bertocco, M.; Ferraris, F.; Offelli, C.; Parvis, M. *Instrumentation and Measurement, IEEE Transactions on*, Vol.47, Iss.5, Oct 1998, p. 1143-1148.
URL: http://ieeexplore.ieee.org/iel4/19/16119/00746572.pdf?isNumber=16119∏=STD&arnumber=746572&arNumber=746572&arSt=1143&ared=1148&arAuthor=Bertocco%2C+M.%3B+Ferraris%2C+F.%3B+Offelli%2C+C.%3B+Parvis%2C+M.

"Building smart services for smart home." Chemishkian, S. *Networked Appliances, 2002 IEEE 4th International Workshop on*, 2002, p. 215-224.
URL: http://ieeexplore.ieee.org/iel5/7710/21127/00980856.pdf?isNumber=21127∏=STD&arnumber=980856&arNumber=980856&arSt=215&ared=224&arAuthor=Chemishkian%2C+S.

"How to build smart appliances?" Schmidt, A.; van Laerhoven, K. *IEEE Personal Communications*, Vol.8, Iss.4, Aug 2001, p. 66-71.
URL: http://ieeexplore.ieee.org/iel5/98/20430/00944006.pdf?isNumber=20430∏=STD&arnumber=944006&arNumber=944006&arSt=66&ared=71&arAuthor=Schmidt%2C+A.%3B+van+Laerhoven%2C+K.

"Providing network connectivity for small appliances: a functionally minimized embedded Web server." Riihijarvi, J.; Mahonen, P.; Saaranen, M.J.; Roivainen, J.; Soininen, J.-P. *IEEE Communications Magazine*, Vol.39, Iss.10, Oct 2001. p. 74-79.
URL: http://ieeexplore.ieee.org/iel5/35/20680/00956116.pdf?isNumber=20680∏=STD&arnumber=956116&arNumber=956116&arSt=74&ared=79&arAuthor=Riihijarvi%2C+J.%3B+Mahonen%2C+P.%3B+Saaranen%2C+M.J.%3B+Roivainen%2C+J.%3B+Soininen%2C+J.-P.

"A flexible data logging device for wind potential measurements and statistical magnitudes." Sparis, B.D.A. *Electronics, Circuits and Systems, The 6th IEEE International Conference on*, Vol.3, 1999, p. 1483-1486.
URL: http://ieeexplore.ieee.org/iel5/6565/17615/00814450.pdf?isNumber=17615∏=STD&arnumber=814450&arNumber=814450&arSt=1483&ared=1486+vol.3&arAuthor=Sparis%2C+B.D.A.

"An Easy-To-Do Embedded Web Server [IC Online]." Witchey, N. *IEEE Internet Computing*, Vol.2, Iss.3, May/Jun 1998. p. 100.
URL: http://ieeexplore.ieee.org/iel4/4236/15040/00683808.pdf?isNumber=15040∏=STD&arnumber=683808&arNumber=683808&arSt=100&ared=100&arAuthor=Witchey%2C+N.

"A low-cost embedded system for Internet based power measurement." Yeary, M.; Sweeney, J.; Swan, B.; Culp, C. *International Journal of Information Technology & Decision Making*, Dec 2003.