Ryan Smith, Theresa Breiner, Brad Thompson
Group 11
CIT 595 - Group Project
4-26-15
Technical Documentation

Our program is called WatchDog. The purpose of the app is to allow you to complete the following tasks: take temperature readings from the Arduino and display them on the watch in real time; toggle the display on the Arduino between fahrenheit and celsius from the watch; put the Arduino in and out of standby from the watch; receive average, low and high temperature readings from the Arduino for the last hour on the watch; monitor a motion sensor with the Arduino and set off an alarm on the watch if triggered; send a message from the watch asking the Arduino to display a scrolling warning message; allow the user to reset the alarm remotely.

Our structure is broken into three parts as directed on the assignment:
1) User Interface - The Pebble watch.
2) Middleware - A program called WatchDog.cpp that includes a built in server listening on a local computer.
3) Sensor - The Arduino with connected thermometer and 7-segment display and a motion sensor.

All communication between the Pebble watch and the Arduino takes place through the middleware which acts as a go-between relaying relevant messages. The middleware also acts as the central repository of information and stores temperature data that will be needed for later requests.

In the interest of being thorough, we will begin by answering the questions from the assignment directly.

1) What is the structure of the messages that are sent from the user interface to the middleware?

All messages that are sent from the user interface to the middleware are sent in the form of an HTTP 1.1 GET request. The WatchDog program on the Pebble activates a number of event listeners corresponding to the watch's buttons. When pressed, the javascript in lab.js forwards the request from the watch using the bluetooth from a connected smartphone.

Each GET request contains a single character indicating which button was pressed. This is sent in the part of the request where the requested file name normally occurs. Here is a sample request:

```
GET /t HTTP/1.1
Host: 10.0.0.4:3002
Accept: */*
Accept-Language: en-us
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: PebbleApp/20150219013617 CFNetwork/711.3.18 Darwin/14.0.0
```

In the above example, the message relayed to the middleware is “t”. There are 7 unique messages that are sent from the user interface to the middleware:

“a” - Asks the middleware to ask the Arduino to toggle the unit of the temperature displayed on the 7-segment display and to return all temperature readings from the middleware in that unit as well. This does not affect the way that numbers are read from the Arduino or stored in the middleware. It just tells the middleware how the user would like to have the data displayed, which it takes into account before returning requests for temperature readings.

“b” - Asks the middleware to return the most recent temperature reading.

“d” - Asks the middleware to return the high, low and average temperatures for the last hour.

“m” - Asks the middleware to request that the Arduino display the warning message.

“r” - Asks the middleware to reset the alarm and request that the Arduino do the same.

“s” - Asks the middleware to request the Arduino enter/exit standby mode.

“t” - Asks the middleware if the motion sensor has been tripped. Trip events are reported by the Arduino but remembered by the middleware so the middleware does not have to ask the Arduino for this information.

When the middleware receives a GET request it checks the message character and discards the rest.

2) What is the structure of the messages that are sent from the middleware to the user interface?

All messages that are sent from the middleware to the user interface are sent as JSON objects containing one single variable called “name” which holds a string message. In most cases the string message contains the information requested and is intended for immediate display on the watch face. There are 2 special messages which are used to relay information to the watch program; “tripped” and “nottripped”. These correspond to the request from the watch that checks the alarm system. “tripped” will send the watch into its tripped state where functionality changes. “nottripped” is ignored.

An example of one of the JSONs sent:

```
{
  “name”:“29.5”
}
```

An error message:

```
{
  “name”:“No data available.”
}
```

3) What is the structure of the messages that are sent from the middleware to the sensor/display?

The messages between the middleware and the sensor/display are very similar to those sent from the user interface to the middleware and consist of a single character that relays information to the display concerning actions the display should take. There are 4 such messages:

“m” - Tells the Arduino to enter message mode which changes the display to a scrolling message.
“r” - Tells the Arduino to exit message mode if it is engaged.
“f” - Tells the Arduino to toggle its display of the temperature between celsius and fahrenheit.
“s” - tells the Arduino to enter/exit standby mode.

The character format is particularly appropriate here given the nature of the serial connection and prevents us from having to perform checks on the Arduino such as verifying complete words were received. The middleware performs these checks when receiving messages going the other direction as they are longer than a single character and it cannot be guaranteed that the entire message is present without verification.

4) What is the structure of the messages that are sent from the sensor/display to the middleware?

All messages sent between the sensor/display and the middleware are read in as strings. Most frequently these are string representations of a double representing the current temperature reading. There are 2 additional special values:
“-274.0” - We have established this temperature to be outside the range of our app’s operation and indicates “no data”. It is sent, processed and stored in the same way as the other temperature readings but is treated differently when working with temperature data. This is the only value that can be transmitted during standby mode.
“tripped” - This tells the middleware that the motion sensor has been triggered.

5) How did you keep track of the average temperature? Describe your algorithm and indicate which part of your code implements this feature.

When the middleware program is executed it creates and joins a separate thread for receiving data from the Arduino. This thread runs the “storeData” function that begins on line 421 in WatchDog.cpp. The Arduino sends a temperature reading every second. When there is an error reading data or the Arduino is in standby mode it sends “-274.0”. These values are stored in an array called “temps” that holds 3600 values, representing the readings from every second in the last hour. “temps” is initially pre-filled with special value -274.0. A variable called nextTempPointer keeps track of the number of readings in the array and tells the program at what index to store the next value received. After an hour, on the 3601st reading, the next TempPointer will loop back to the beginning of the array to index 0 and will begin incrementing again from there. In this way, the “temps” array holds the data from the previous hour or all data received if the program has been running for less than 1 hour.

When a request is received for the most recent temperature a function called packageTempJSON (line 128 in WatchDog.cpp) is called. This checks for a number of possible error states including an empty array, values of -274.0, and lost connection with the Arduino, then inserts the reading from temps[lastTempPointer - 1] into a JSON object to be sent to the watch. If an error state had been encountered an appropriate message is relayed instead.

When a request is received for the max, min and average temperatures, a function called packageAvgJSON (line 87 in WatchDog.cpp) is called. The function checks again for a variety of possible errors and then calls three helper functions, findMin, findMax and findAverage. These functions will read all values from the temps array in order to make their calculations but

will control for extreme outliers and the special value -274.0 which is ignored. When these functions are returned their results are packaged into a JSON and sent.

All interaction with the “temps” array and nextTempPointer is done within locks to synchronize it between threads. While a single fluctuating temperature would probably not destroy our results, this ensures we will return the most accurate information possible.

6) What are the three additional features that you implemented? Indicate which parts of your code implement these features.

We implemented 3 additional features:

a) Automatic Temperature Updating.

When the temperature is requested from WatchDog, the app will make continuous requests to the server on the user’s behalf to keep the watch updated with the most recent temperature reading. This is handled by a tick_timer in the watch program which binds one of our functions to the time-keeping component of the watch (tick_handler method, line 167 in main.c). The function is called every second and a request is sent for the temperature every other second if temperature is desired. This is determined by a variable called “wantAverage”. When the user presses a button corresponding to this polling feature “wantAverage” is set to true. When the user requests a feature that is not needed in real-time such as max/min/avg “wantAverage” is set to false so as not to overwrite the desired information.

b) Motion Sensor Alarm.

When the motion sensor is triggered on the Arduino it sends a special message to the middleware to inform it (line 155 in WatchDog.ino). The middleware records this event by setting a boolean value called “tripped” to true. In the aforementioned tick_timer method in main.c, a request to check the status of “tripped” is sent every other second and a message indicating “tripped” or “nottripped” is returned from the middleware (function checkTripped, line 262 in WatchDog.cpp). If “tripped” is returned the watch goes into tripped/alarm mode. “INTRUDER ALERT!!!” is displayed on the screen and the watch pulsates repeatedly.

The alarm mode is indicated by a variable in the watch program also named “tripped”. When this variable is true, the buttons on the watch behave differently and the alarm can be reset by double clicking the up button (up_double_clicked_handler, line 150 in main.c). When this takes place, tripped is set to false on the watch and a reset request is sent to the middleware. The middleware interprets this message and sets its tripped variable back to false and returns a message confirming it has done so. The middleware also sends a message to the Arduino asking it to exit message mode if it has entered it (next feature). At this point the alarm will cease and the middleware is ready to report the next motion detector trigger.

c) Send Scrolling Message to Intruder

When the watch program is in the alarm state the select button can be employed by the user to send a message to the intruder (select_click_handler, line 90 in main.c). After detecting motion, the Arduino behaves normally in case the user would prefer not to make the intruder aware of his/her detection. When the button is pressed, a message is sent to the middleware asking it to contact the Arduino and request that it enter message mode. This is controlled by a variable on the Arduino called “msgMode”. When set to true, the RGB light will flash between red and blue and the 7-segment display will show a scrolling message

saying “POLICE CALLED” (line 205 in WatchDog.ino). The Arduino will maintain normal functioning and temperature reporting during this time and this will only affect the display. When the user resets the alarm, the Arduino will exit message mode after finishing the message currently scrolling on the display.

Citations

Parts of each program we are submitting have come from outside sources. These are sources provided to us during the course through assignments such as the httpserver homework and this project. We have not used any other outside sources. I’ll list citations by file:

WatchDog.cpp: Fundamental server operations were borrowed from the following 2 sources:
1 - http://www.prasannatech.net/2008/07/socket-programming-tutorial.html
2 - http://www.binarii.com/files/papers/c_sockets.txt
This is mostly found in function server_thread between lines 320-355 and in function main between lines 498-503.

main.c: The majority of this code come from I2C_7SEG_Temperature.pde, Copyright 2008, Gravitech. Original file modified to contain all functionality mentioned in description except reading and transmitting the temperature and displaying celsius temp reading on 7-Seg display. The majority of these alterations occur within the infinite loop in the loop() function.

lab.js: Initial skeleton and onload method provided by Dr. Chris Murphy in this assignment.

WatchDog.ino: Initial skeleton for app_event_loop and basic send/receive functions provided by Dr. Chris Murphy in this assignment.