Group 1

PebbleDream Technical Document

Group Member: Yunchen Wei, Lu Lu, Yao Chu

What is the structure of the messages that are sent from the user interface to the middleware? give examples

Message sent from Pebble Watch to cell phone is in the form of a key-value pair. In our case, key is always '0'. And we simply copy the value as part of url and forward it from cell phone to server (middleware).

For example, if I want to query lowest Temperature so far, Pebble watch would send {'0': 'lowTemp'} to cell phone, which would in turns forward it as an XMLHttpRequest to server with URL 'http://[server_ip:port]/lowTemp'. We make the lowTemp as a passed-in function parameter, so we can minimize the work lies on the JavaScript side.

All messages: (see menu.c *menu_select_callback* function)

Query current temperature: /curTemp
Query average temperature: /avgTemp
Query highest temperature: /highTemp
Query lowest temperature: /lowTemp
Query temperature history: /temps
Convert temperature into Celcius: /showC
Convert temperature into Fahrenheit: /showF
Ardiuno stand-by: /stop
Arduino resume: /resume
Query Philly weather: /weather (XMLHttpRequest is a bit different)
Query Yahoo! stock price: /stock (XMLHttpRequest is a bit different)

What is the structure of the messages that are sent from the middleware to the user interface? give examples

To simplify the process, all messages sent by server are in their final format. (what we send over air is what we will display on Pebble Watch screen, so no extra work would be done on Pebble Watch side.)

The format of messages sent is JSON format. e.g. {'msg': 'some text here'}. Again, to make things easier, we always use 'msg' as the key.

Though we have one exception. For temperature history, what we send over air is ten doubles, separated by commas. e.g. {'msg': '12.3,12.5,...,22.2,12.2,'}

Extra work need to be done on Pebble Watch side to figure out how to draw the line chart for temperature history.

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

In the loop function (starting from line 66) of arduino.ino, line 159 to line 170 is used to deal with different messages received from the middleware.

For the fundamental function, the middleware will send a single char to sensor/display.
'c' - will make the temperature on seven-segment display in Celsius;

'f' - will make the temperature on seven-segment display in Fahrenheit;
's' - will stop the sensor from sending current temperature to the server;
'r' - will resume reporting readings.

For the additional function, the middleware will send a string to sensor/display with prefix "tmp" or "stk".

"stk….." - will make it display the price of stock and the corresponding light. The string starts with "stk", then a digit of '+' or '-' indicating whether the price is increasing or decreasing and the rest being the price of the stock.
For example, "stk+123.45" means the price of stock is $123.45 and it's increasing. So the display will show the price 123.45 with GREEN light on. "stk-321.87" means the price of stock is $321.87 and it's decreasing. So the display will show the price 321.87 with RED light on.

"tmp….." - will make it display the temperature forecast. The string starts with "tmp" and followed by the temperature with the last digit being 'c' or 'f' indicating whether it is in Celsius or in Fahrenheit. For example, "tmp23.45c" means temperature forecast is 23.45°C So the display will show 23.45 with a char 'c' at the end. "tmp75.12f" means temperature forecast is 23.45°F. So the display will show 75.12 with a char 'F' at the end.

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

The messages sent from the sensor to the middleware will always represent temperature in Celsius for consistency and convenience. The method SerialMonitorPrint (starting from line 406) in arduino.ino is to send message to the middleware. To make sure the server receives the complete message, the temperature will be bracketed inside a prefix 'S' indicating start and a postfix 'E' indicating end. For example, the temperature 26.625°C will be sent as S27.625E.

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

The server stores the temperature information in a vector of SensorData object, which contains both the temperature and the time when the temperature is reported. When a request asking for the average temperature arrives, the server calls the function getAverage(line 296 server.cpp). This function sums up the temperature in the last hour and divided by number of temperature counted. If the server does to have enough data collected for an hour, the function will calculate the average temperature based on timestamp of each SensorData.

What are the three additional features that you implemented? indicate which parts of your code implement these features

1. Display line chart over temperature history in the last minute(server.cpp line 324 function sendTemps) on Pebble Watch → Server send history data, Pebble Watch parse it and generate the graph. (main.c layer_graph_update_callback function, line 99 – line 134)

2. Query Philly weather report, display information on Pebble Watch and show it on Arduino → Pebble watch send the query request to cell phone, cell phone query it and have results sent to both Pebble Watch and server (Middleware.js queryWeather function). Server would then forward the result to Arduino(server.cpp line 166).

3. Query Yahoo! stock price, display stock price on Pebble Watch and show price on Arduino (Also, we use red light to indicate price decrease, green light to indicate price increase.) → Pebble watch send the query request to cell phone, cell phone query it and have results sent to both Pebble Watch and server (Middleware.js queryStock function). Server would then forward the result to Arduino(server.cpp line 171). For Arduino, the method DisMsg (starting from line 175) in arduino.ino is used to display the message on seven-segment display in a rolling way.

Pebble Watch

Pebble watch User Interface contains mainly two parts:

1. A feature-selection menu
2. A graphical layer for displaying query result and show temperature line charts

For basic Arduino-based feature, Pebble Watch just send a query string as part of URL to server, and sever would send back messages back to Pebble Watch. So basically Pebble Watch just display result on the screen, plus some error handling (e.g. not sending successfully, message drop, etc.)

For additional features query weather + query stock price, Pebble Watch would send a request to cell phone, and cell phone serves as another middleware, which would in turns do the actual query, get the result and send to both Pebble Watch and server. (Middleware.js queryWeather + queryStock function)

For additional features draw line chart over the last minute temperature, it is implemented as follows: (main.c layer_graph_update_callback + myatof function. Since Cloud Pebble does not support C atof function, so we have to implement our own version.)

1. Pebble Watch asks for 10 number of last minute from server
2. Server send back the message contains the 10 number
3. Pebble Watch convert the string message into ten double number
4. Determine the min and max number in these 10 number, so we can decide how to draw the ten points and edges properly (e.g. draw the min number at the bottom of screen, draw the max number at the top of screen). We also need to know the screen size.
5. Draw ten points with radius on screen.
6. Connect the ten points

2. Server

The server.cpp main function opens two threads. The first thread getTem is responsible for reading temperature information, turning it into a SensorData object and storing it in a SensorData vector. It first creates a file descriptor for Arduino. If the connection is successful, it starts to read bytes from Arduino and append the bytes to a string message until a '\n' is read. Then the temperature information is retrieved and turned into a SensorData object, which records the temperature as well as the time when the temperature is read. The SensorData object is then added into a SensorData vector to keep track of the temperature history. Several

functions(server.cpp line 250, 258, 286, 291, 296, 318, 324) are provided to calculate temperature statistics such as average temperature from the vector.

The second thread startServer opens the server and listens to request from some port specified by the command line. When a request arrives, the GET information is retrieved(server.cpp line 122) and corresponding response is sent to the client. If the request asks for temperature information, the server calls the corresponding functions(server.cpp line 250, 258, 286, 291, 296, 318, 324) to get the temperature statistics and respond with a JSON object. When the request asks to communicate with Arduino, such as displaying number on Arduino, changing the light color, the server sends the control message to Arduino through the write function.

3. Arduino

The sensor will send back to server the temperature in Celsius every second if not in stand-by mode. And listen to the message send back by server. If it is ‘c’, the mode for display will be set to Celsius (also the default mode). If it is ‘f’, it will be set to Fahrenheit. The temperature will be displayed with only one digit after decimal point but the temperature send back to server will be as accurate as it senses. The F to C conversion for display is done in the Arduino side.
For the additional features, the main task for Arduino is to display every digit of the message send from Server. So the display need to be in a rolling mode, which means the current display will be shifted left by one digit after a certain interval