CIT 595 SPRING 2015

PROJECT

Pebble Watch Temperature Monitoring App

Group 8

Yuetong Chi

Zeyu Li

Jingyuan Wu

yuetongc@seas.upenn.edu

lzeyu@seas.upenn.edu

wujingyu@seas.upenn.edu

April 26, 2015

Chapter 1
Introduction

We implemented an app on Pebble Watch, which users can view current temperature and remotely control the temperature sensor. The project consists of three part:
Temperature sensor (Arduino): Arduino reports temperature every second. The current temperature will display on a seven-segment display board. Also, when temperature is higher than 30, the RGB light will turn red to alert users.
Middleware: The middleware connects Arduino and Pebble watch. On one hand, it will accept temperature reported by Arduino every second and store temperatures in local disk. On the other hand, the middleware also listens on the requests from Pebble watch. Based on the requests sent from Pebble watch, the server will give the current temperature back or control Arduino.
User Interface - Pebble watch: Pebble watch will communicate with server over the Internet via Android phone. Pebble watch send requests to server to get specific information about current temperature. Pebble watch can also send instructions to server in order to control Arduino.

Chapter 2
Basic Functionality

- Pebble Watch sends START request to server and get temperature readings, such as current temperature, highest, lowest and average temperature in the past hour. The code from server part is listed as below:

```
else if(strcmp(token, "/start") == 0){
  if(recent_temp == 0){
     char reply[200];
    strcpy(reply, "{\"fail\":\"fail\"}");
    send(fd, reply, strlen(reply), 0);
    printf("arduino does not connect \n");

  }else{
  pthread_mutex_lock(&recent_temp_lock);
  char reply[200];
  strcpy(reply, "{\n\"recentTemp\": \"");
  char high[20];
  if(cfstate == 'f') sprintf(high, "%.2f", ctof(highest_temp));
  else sprintf(high, "%.2f", highest_temp);
  char low[20];
  if(cfstate == 'f') sprintf(low, "%.2f", ctof(lowest_temp));
  else sprintf(low, "%.2f", lowest_temp);
  char avg[20];
  if(cfstate == 'f') sprintf(avg, "%.2f", ctof(average));
  else sprintf(avg, "%.2f", average);
  char current[20];
  if(cfstate == 'f') sprintf(current, "%.2f", ctof(recent_temp));
  else sprintf(current, "%.2f", recent_temp);
  strcat(reply, current);
  strcat(reply, "\",\n");
  strcat(reply, "\"high\": \"");
  strcat(reply, high);
  strcat(reply, "\",\n");
  strcat(reply, "\"low\": \"");
  strcat(reply, low);
  strcat(reply, "\",\n");
  strcat(reply, "\"avg\": \"");
  strcat(reply, avg);
  strcat(reply, "\",\n");
  strcat(reply, "\"warning\": \"");
  if(recent_temp>highest){
    strcat(reply, "true\"");
  }
  else strcat(reply, "false\"");
  strcat(reply, "\n}");
  send(fd, reply, strlen(reply), 0);
  pthread_mutex_unlock(&recent_temp_lock);
}
```

The code from pebble watch part is listed as below:

```
void select_start_handler(){
  DictionaryIterator *iter;
  app_message_outbox_begin(&iter);
  int key = 0;
  Tuplet value = TupletCString(key, "start");
  dict_write_tuplet(iter, &value);
  app_message_outbox_send();
  if(!timerStarted){
    int sleeptime = 2000; // number of milliseconds to sleep
    timer = app_timer_register(sleeptime, timer_callback, NULL);
    timerStarted = true;
  }
}
```

```
 int key = 0;
 Tuple *data_tuple = dict_find(received, key);
 if (data_tuple) {
   if (data_tuple->value) {
     // put it in this global variable
     strcpy(msg, data_tuple->value->cstring);
     text_layer_set_text(msg_layer, msg);
   }
   else {
     strcpy(msg, "no value!");
     text_layer_set_text(msg_layer, "no value!");
   }
 }
 else {
  text_layer_set_text(msg_layer, "no temp message!");
 }
```

```
if (response.recentTemp) {
  msg = "Curent: " + response.recentTemp;
}
if (response.high) {
  msg = msg + "\n" + "High: " + response.high;
}
if (response.low) {
  msg = msg + "\n" + "Low: " + response.low;
}
if (response.avg) {
  msg = msg + "\n" + "Avg: " + response.avg;
  Pebble.sendAppMessage({"0": msg});
}
```

- Pebble Watch sends CONVERT request to server and makes Arduino to change report mode. Arduino will display temperature in Fahrenheit. (By default, in Celsius)

The code from server part is listed as below:

```
else{
  if(strcmp(token, "/convert") == 0){
    printf("convert: compare token %s\n", token);
    if(cfstate == 'c'){
      cfstate = 'f';
      int bytes_written = write(arduino, "f", 1);
      printf("send convert request to arduino \n");
    }
    else{
      cfstate = 'c';
      int bytes_written = write(arduino, "c", 1);
      printf("send convert request to arduino \n");
    }
  }
```

The code from pebble watch part is listed as below:

```
void select_convert_handler(){
  DictionaryIterator *iter;
  app_message_outbox_begin(&iter);
  int key = 0;
  Tuplet value = TupletCString(key, "convert");
  dict_write_tuplet(iter, &value);
  app_message_outbox_send();
}
```

- Pebble Watch sends PAUSE/RESUME request to server and makes Arduino to stop/resume reporting temperatures.

The code from server part is listed as below:

```
else if(strcmp(token, "/pause") == 0){
  printf("pause: compare token %s\n", token);
  int bytes_written = write(arduino, "p", 1);
  printf("send pause request to arduino ");
}
else if(strcmp(token, "/resume") == 0){
  int bytes_written = write(arduino, "r", 1);
}
```

The code from pebble watch part is listed as below:

```
void select_pause_handler(){
  DictionaryIterator *iter;
  app_message_outbox_begin(&iter);
  int key = 0;
  Tuplet value = TupletCString(key, "pause");
  dict_write_tuplet(iter, &value);
  text_layer_set_text(msg_layer, "Pause");
  app_timer_cancel(timer);
  app_message_outbox_send();

}

void select_resume_handler(){
  DictionaryIterator *iter;
  app_message_outbox_begin(&iter);
  int key = 0;
  Tuplet value = TupletCString(key, "resume");
  dict_write_tuplet(iter, &value);
  int sleeptime = 2000;
  timer = app_timer_register(sleeptime, timer_callback, NULL);
  app_message_outbox_send();
}
```

Chapter 3

Additional Functionality

- Temperature Alert: when the temp is higher than the maximum reasonable temperature, which is 30 Celsius (°C) during the Summer season, and over 10 Celsius (°C) during the Winter season, the sensor will automatically send an alert to the watch, which will make the watch vibrate.
- Temperature trend: show the temp trend within the day in a chart.
- Season Switch: The reasonable temperature ranges of different seasons are be different. User can choose either Summer Season Mode or Winter Season Mode accordingly. For Summer Season, the reasonable temperature range is from 23 Celsius to 30 Celsius. For Winter Season, the reasonable temperature range is from 0 Celsius to 10 Celsius. When the temperature stays within the range, the light on sensor will be green; when the temperature is higher than the reasonable range, the light on sensor will be red; when the temperature is lower than the reasonable range, the light on sensor will be blue.

Implementation

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

  Pebble Watch sends different requests by specifying different url paths to the middleware.

  For example:
  - send START request (get temperature readings from Arduino) : GET /start
  - send CONVERT request (convert temperature report between Fahrenheit/Celsius mode) : GET /convert
  - send PAUSE request (let Arduino stop sending report to pebble watch) : GET /pause
  - send RESUME request (let Arduino resume sending report to pebble watch) : GET /resume

```
Pebble.addEventListener("appmessage",
  function(e) {
    if(e.payload){
      if(e.payload.msg){
        if(e.payload.msg == "convert"){
          sendToServer("convert");
        }
        else if(e.payload.msg == "pause"){
          sendToServer("pause");
        }
        else if(e.payload.msg == "start"){
          sendToServer("start");
        }
        else if(e.payload.msg == "resume"){
          sendToServer("resume");
        }
        else sendToServer(e.payload.msg);
        }
}
});



function sendToServer(command) {

  var req = new XMLHttpRequest();
  var ipAddress = "158.130.107.49"; // Hard coded IP address
  var port = "3001"; // Same port specified as argument to server
  var url = "http://" + ipAddress + ":" + port + "/" + command;
  var method = "GET";
  var async = true;
  var isConnected = false;
```

- what is the structure of the messages that are sent from the middleware to the user interface? Each time user interface request information, middleware sends back json object which includes high, low, avg temp, and warning status(true/ false) back to the pebble watch.

```
  }else{
  //send high, low, avg temp data to watch
  pthread_mutex_lock(&recent_temp_lock);
  char reply[200];
  strcpy(reply, "{\n\"recentTemp\": \"");
  char high[20];
  if(cfstate == 'f') sprintf(high, "%.2f", ctof(highest_temp));
  else sprintf(high, "%.2f", highest_temp);
  char low[20];
  if(cfstate == 'f') sprintf(low, "%.2f", ctof(lowest_temp));
  else sprintf(low, "%.2f", lowest_temp);
  char avg[20];
  if(cfstate == 'f') sprintf(avg, "%.2f", ctof(average));
  else sprintf(avg, "%.2f", average);
  char current[20];
  if(cfstate == 'f') sprintf(current, "%.2f", ctof(recent_temp));
  else sprintf(current, "%.2f", recent_temp);
  strcat(reply, current);
  strcat(reply, "\",\n");
  strcat(reply, "\"high\": \"");
  strcat(reply, high);
  strcat(reply, "\",\n");
  strcat(reply, "\"low\": \"");
  strcat(reply, low);
  strcat(reply, "\",\n");
  strcat(reply, "\"avg\": \"");
  strcat(reply, avg);
  strcat(reply, "\",\n");
  //if current temp is out of comfortable range, send warning to watch
  strcat(reply, "\"warning\": \"");
  if(recent_temp>highest){
    strcat(reply, "true\"");
  }
  else strcat(reply, "false\"");
  strcat(reply, "\n}");
  send(fd, reply, strlen(reply), 0);
  pthread_mutex_unlock(&recent_temp_lock);
}
```

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

  Middleware and sensor connects by serial communication. Middleware writes data to the channel by using a single character to indicate a specific instruction.

  For example:

  - send CONVERT request to Arduino: write(arduino, “c”, 1)

    ```
      int bytes_written = write(arduino, "c", 1);
      printf("send convert request to arduino \n");
    }
    ```

  - send PAUSE request to Arduino: write(arduino, “p”, 1)

```
else if(strcmp(token, "/pause") == 0){
  printf("pause: compare token %s\n", token);
  int bytes_written = write(arduino, "p", 1);
  printf("send pause request to arduino ");
}
```

- send RESUME request to Arduino: write(arduino, “r”, 1)

```
else if(strcmp(token, "/resume") == 0){
  int bytes_written = write(arduino, "r", 1);
}
```

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

  Middleware and sensor connects by serial communication. Sensor writes temperature (float number) to middleware.

  For example:

  - Arduino report the current temperature: Serial.print(temperature)

```
/*********************************************************************************
 Function Name: SerialMonitorPrint

 Purpose:
   Print current read temperature to the serial monitor.
*********************************************************************************/
void SerialMonitorPrint (byte Temperature_H, int Decimal, bool IsPositive)
{
    //Serial.print("The temperature is ");
    if (!IsPositive)
    {
      Serial.print("-");
    }
    Serial.print(Temperature_H, DEC);
    Serial.print(".");
    Serial.print(Decimal, DEC);
    //Serial.print(" degree C");
    Serial.print("\n\n");
}
```

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

  We have constructed a class <record.h> to serve as a basic unit to record all incoming temperature information. The source code is listed as below:

```
#ifndef RECORD_H
#define RECORD_H

#endif // RECORD_H

class Record{
private:float high, low, avg;
public:Record(){}
    Record(float high, float low, float avg){
        this->high = high;
        this->low = low;
        this->avg = avg;
    }
    float getHigh(){
        return high;
    }
    float getLow(){
        return low;
    }
    float getAvg(){
        return avg;
    }
};
```

In our <Server.cpp>, we have declared a global variable 7 global variables to track the highest temp, lowest temp, recent temp, how many temp we have received, average temp and a temp vector.

```
// temperature global variable
float highest_temp = numeric_limits<float>::min();
float lowest_temp = numeric_limits<float>::max();
float recent_temp = 0;
int counter = 0;
float sum = 0;
float average = 0;
vector<Record> records;
```

And the algorithm we used is that every time we received a new temperature, we check to see if it is higher than global highest temp, or lower than global lowest temp; if it does, we update the highest temp or lowest temp to the new temp. Also, we will increment the counter by 1 , and add the new temp to the temp sum. We can simply calculate the average by sum temp / counter. Then, when counter reaches 3600 (one hour), we save the highest temp, lowest temp and average temp to the temp vector we have declared before for future uses. When the user asks for highest temp and lowest temp records, we can access the data from temp vector directly. The source code is listed as below:

```
    if(buf[bytes_read-1] == '\n'){
      flag = 1;
      if(message[0] != '\n'){
        // each time get a temperature, save it into the array
        printf("The message is ");
        printf("%s\n", message);
        float temp = atof(message);
        printf("The temperature is %f \n", temp);
        if(temp<50 && temp>highest_temp) highest_temp = temp;
        if(temp>0 && temp<lowest_temp) lowest_temp = temp;
        pthread_mutex_lock(&recent_temp_lock);
        recent_temp = temp;
        pthread_mutex_unlock(&recent_temp_lock);
        counter++;
        sum += temp;
        if(counter==60){
            average = sum/counter;
            Record record(highest_temp, lowest_temp, average);
            pthread_mutex_lock(&records_lock);
            records.push_back(record);
            pthread_mutex_unlock(&records_lock);
            counter = 0;
            sum = 0;
            highest_temp = numeric_limits<float>::min();
            lowest_temp = numeric_limits<float>::max();
        }


      }
    }
  }

}
```

Chapter 4

User documentation

- Enter the app's user interface: The user interface is divided into two parts. The upper part will display current temperature and lowest, highest and average temperature in the past hour. The lower part is a control menu. You can use upper and lower button on the right side of the pebble watch to scroll the control menu. If you press the middle button, it will send corresponding requests to middleware.
- Click the start: get current temperature readings. You can view updated temperature readings from the upper part. When the temperature is higher than 30 (in summer mode, 10 in winter mode), the pebble watch will vibrate. When the temperature is lower than 30(10), the pebble watch will stop vibrating.
- Click the convert: change the report mode of Arduino. You can decide to view temperature report in Fahrenheit or Celsius mode. It will also change the display in Arduino.
- Click the pause: put Arduino in stand-by mode.
- Click the resume: make Arduino resume reporting to pebble watch.
- Click the mode: change the season mode between summer and winter. It will set different cold/hot warning line for Arduino.
- Click the trend: view the trend of temperatures in a day. Click again, then return back to original user interface.

Chapter 5
Additional Feature Description

Our team has implemented three additional features. They are listed as below:

1. Temperature Alert: when the temp is higher than the maximum reasonable temperature, which is 30 Celsius (°C) during the Summer season, and over 10 Celsius (°C) during the Winter season, the sensor will automatically send an alert to the watch, which will make the watch vibrate.

2. Temperature trend: show the temp trend within the day in a chart.

3. Season Switch: The reasonable temperature ranges of different seasons are be different. User can choose either Summer Season Mode or Winter Season Mode accordingly. For Summer Season, the reasonable temperature range is from 23 Celsius to 30 Celsius. For Winter Season, the reasonable temperature range is from 0 Celsius to 10 Celsius.
When the temperature stays within the range, the light on sensor will be green; when the temperature is higher than the reasonable range, the light on sensor will be red; when the temperature is lower than the reasonable range, the light on sensor will be blue.

Additional Feature Implementations

1. Temperature Alert:
Server: every second the server will send information back to pebble watch including the warning reply which is either true or false. If the current temperature is within alert range, it will set the warning reply to true, and if it is not, the server will set the warning reply to false. Then, the pebble watch will catch the warning reply and respond accordingly. The source code of Server is listed as below:

```
    }
    else if(strcmp(token, "/start") == 0){
      pthread_mutex_lock(&recent_temp_lock);
      char reply[200];
      strcpy(reply, "{\n\"recentTemp\": \"");
      char high[20];
      if(cfstate == 'f') sprintf(high, "%.2f", ctof(highest_temp));
      else sprintf(high, "%.2f", highest_temp);
      char low[20];
      if(cfstate == 'f') sprintf(low, "%.2f", ctof(lowest_temp));
      else sprintf(low, "%.2f", lowest_temp);
      char avg[20];
      if(cfstate == 'f') sprintf(avg, "%.2f", ctof(average));
      else sprintf(avg, "%.2f", average);
      char current[20];
      if(cfstate == 'f') sprintf(current, "%.2f", ctof(recent_temp));
      else sprintf(current, "%.2f", recent_temp);
      strcat(reply, current);
      strcat(reply, "\",\n");
      strcat(reply, "\"high\": \"");
      strcat(reply, high);
      strcat(reply, "\",\n");
      strcat(reply, "\"low\": \"");
      strcat(reply, low);
      strcat(reply, "\",\n");
      strcat(reply, "\"avg\": \"");
      strcat(reply, avg);
      strcat(reply, "\",\n");
      strcat(reply, "\"warning\": \"");
      if(recent_temp>highest){
        strcat(reply, "true\"");
      }
      else strcat(reply, "false\"");
      strcat(reply, "\n}");
      send(fd, reply, strlen(reply), 0);
      pthread_mutex_unlock(&recent_temp_lock);
    }
  }
```

Pebble watch:

```
static void timer_callback(void *data) {
    DictionaryIterator *iter;
    app_message_outbox_begin(&iter);
    int key = 0;
    Tuplet value = TupletCString(key, "start");
    dict_write_tuplet(iter, &value);
    app_message_outbox_send();
    int sleeptime = 2000;
    timer = app_timer_register(sleeptime, timer_callback, NULL);
}
```

```
key = 1;
data_tuple = dict_find(received, key);
if(data_tuple) {
  text_layer_set_text(msg_layer, data_tuple->value->cstring);
  vibes_double_pulse();
}
```

```
if(response.warning == "true"){
  var curTemp = response.recentTemp;
  var warn = "Hign Temp Warning" + "\n" + curTemp;
  Pebble.sendAppMessage({"1": warn });
}
```

2. Temperature trend:
Server: when pebble watch sends back the trend request, server will send back latest 10 average temperature records in temp vector. The source code is listed as below:

Please note: our trend will draw the trend based on latest 10 hours average temperature, which we saved in vector. If user tries to test this function, please go to the server.cpp line 290 to change the counter value from 3600 (one hour) to 15 (second). In this way, user will be able to see the trend line after 150 seconds.

```
else if(strcmp(token, "/trend") == 0){
  int start = 0;
  if(records.size()>=10){
    start = records.size()-10;
  }
  char reply[100];
  strcpy(reply, "{\"data\": \"");
  for(int i=start;i<records.size();i++){
    float record_avg = records.at(i).getAvg();
    int avg_convert = (int)(5*record_avg-125);
    char data[20];
    sprintf(data, "%d", avg_convert);
    strcat(reply, data);
    strcat(reply, "#");
  }
  strcat(reply, "\"}");
  send(fd, reply, strlen(reply), 0);
}
```

code in pebble watch are as follows:

```
void select_trend_handler(){
  if(!layer_get_hidden((Layer*)msg_layer)){
      DictionaryIterator *iter;
      app_message_outbox_begin(&iter);
      int key = 0;
      Tuplet value = TupletCString(key, "trend");
      dict_write_tuplet(iter, &value);
      app_message_outbox_send();
      layer_set_hidden((Layer*)msg_layer, true);
      layer_set_hidden((Layer*)mode_layer, true);
      layer_set_hidden(chart_layer, false);
  }
  else{
    layer_set_hidden((Layer*)msg_layer, false);
    layer_set_hidden((Layer*)mode_layer, false);
    layer_set_hidden(chart_layer, true);
  }
}
```

```
static void chart_update_proc(Layer *chart_layer, GContext *ctx){
  int i = 0;
  graphics_context_set_stroke_color(ctx, GColorBlack);
  for(;i<10;i++){
    int y = points[i];
    int x = 5+i*10;
    GPoint p0 = GPoint(x, 62);
    GPoint p1 = GPoint(x, 55-y);
    graphics_draw_line(ctx, p0, p1);
  }
}
```

```
  key = 3;
  Tuple *trend_tuple = dict_find(received, key);
  if(trend_tuple) {
    strcpy(data, trend_tuple->value->cstring);
    parseInt(data);
    layer_set_update_proc(chart_layer, chart_update_proc);
  }

}
```

```
if(response.data){
  var data = response.data;
  Pebble.sendAppMessage({"3": data});
}
```

3. Season Switch:
Server: when pebble watch sends back the change season request, server will receive the request and then update the season mode and the temp range accordingly and also send the request back to arduino. The the light will change color based on whether the current temp is within the reasonable temperature range.
Server code:

```
    else if(strcmp(token, "/summer") == 0){
      seasonmode = 's';
      highest = 30;
      lowest = 23;
      int bytes_written = write(arduino, "s", 1);
      char reply[100];
      strcpy(reply, "{\"mode\": \"summer\"}");
      send(fd, reply, strlen(reply), 0);
      printf("send summer mode request to arduino");
    }
    else if(strcmp(token, "/winter") == 0){
      seasonmode = 'w';
      highest = 10;
      lowest = 0;
      char reply[100];
      int bytes_written = write(arduino, "w", 1);
      strcpy(reply, "{\"mode\": \"winter\"}");
      send(fd, reply, strlen(reply), 0);
      printf("send winter mode request to arduino");
    }
```

Arduino code:

```
    if(input == 's'){
      cold = 23;
      hot = 30;
    }
    if(input == 'w'){
      cold = 10;
      hot = 0;
    }
  }
}
```

Code for pebble watch:

```
void UpdateRGB (byte Temperature_H)
{
  digitalWrite(RED, LOW);
  digitalWrite(GREEN, LOW);
  digitalWrite(BLUE, LOW);        /* Turn off all LEDs. */

  if (Temperature_H <= cold)
  {
    digitalWrite(BLUE, HIGH);
  }
  else if (Temperature_H >= hot)
  {
    digitalWrite(RED, HIGH);
  }
  else
  {
    digitalWrite(GREEN, HIGH);
  }
}

/*****************************************************************************
```

Pebble Watch Code:

```
void select_mode_handler(){
  int key = 0;
  if(mode == 2){
    DictionaryIterator *iter;
    app_message_outbox_begin(&iter);
    Tuplet value = TupletCString(key, "summer");
    dict_write_tuplet(iter, &value);
    app_message_outbox_send();

  }
  else if(mode == 1){
    DictionaryIterator *iter;
    app_message_outbox_begin(&iter);
    Tuplet value = TupletCString(key, "winter");
    dict_write_tuplet(iter, &value);
    app_message_outbox_send();
  }
}
```

```
key = 2;
Tuple *mode_tuple = dict_find(received, key);
if(mode_tuple){
  strcpy(mode_msg, mode_tuple->value->cstring);
  text_layer_set_text(mode_layer, mode_msg);
  if(mode == 1) mode = 2;
  else mode = 1;
}
```

```
if(response.mode){
  var mode = "";
  if(response.mode == "summer") mode = "SMR";
  else if(response.mode == "winter") mode = "WIN";
  Pebble.sendAppMessage({"2": mode});
}
```

Chapter 6
Readme

- Connect computer with Arduino, please check the usb port and change the code with the correct usb port number.
- Run the server by specifying the port number, please run it on 3000.
- We aim at displaying the temperature trend last 10 hours. However, since it makes testing inconvenient, we collect data from each 15 seconds so that after 150 seconds we can view the trend.