CS2310 - Multi-media Software Engineering

Keren Ye (key36@pitt.edu)

December 2016

Contents

1 Introduction

I extend the project from CS2310 exercise 4. Different from exercise 4, I use real sensors in the project and design the circuits to connect sensors with Arduino board. Besides that, I use a more complicated system design and propose a more detailed scenario for the personal healthcare System. The system can be seen as a prototype of a real personal healthcare system.

2 System Design

Figure 1 shows the system design of the personal healthcare system built on SIS. As shown in the graph, there are five components in the system including GUI, Gesture Recognizer, Emergency Manager, Homecare Staff, and Uploader. The basic functionalities of each component are listed in Table 1.

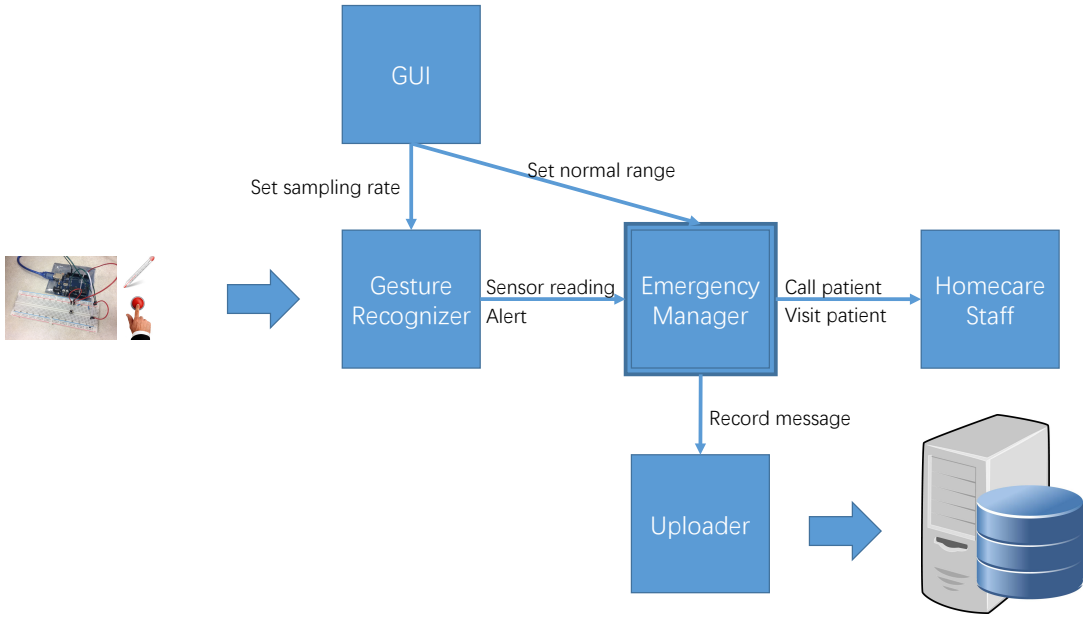

Figure 1: The personal healthcare system built on SIS.

Component	Role	Functionalities
Gesture Recognizer	Basic	Gathers sensor data, sends alert and reading
Emergency Manager	Controller	Filters sensor reading by threshold, sends emergency
Homecare Staff	Advertiser	Displays the notification to the staff
GUI	Monitor	Configures the sampling rate and normal range of the sensor
Uploader	Advertiser	Records the messages in DB, sends alert email

Table 1: The functionalities of components.

2.1 Gesture Recognizer Component

The Gesture Recognizer Component is a basic component (Shown in Figure 2). It is responsible for collecting data from patient, deciding whether the patient is fine. In this project, Gesture Recognizer Component is not an actual gesture recognition device. Instead, a temperature sensor (implemented by thermistor) and button are used to implement the concept. The circuits for connecting the sensor with an Arduino board are demonstrated in Figure 3 and Figure 4.

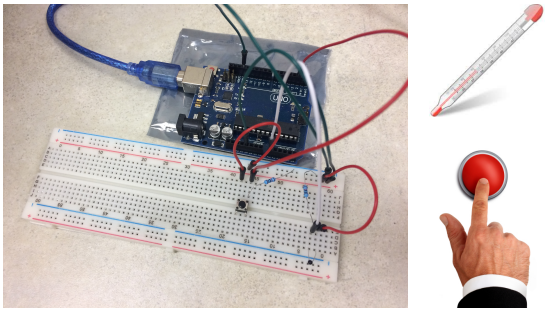

Figure 2: The Gesture Recognizer Component.

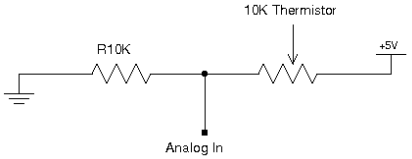

Figure 3: This graph illustrates a simple voltage divider circuit for using the thermistor.

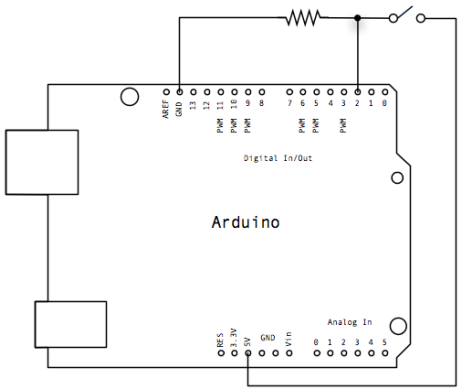

Figure 4: This graph illustrates the circuit for using a button.

Reading Message: Gesture Recognizer Component sends readings of the temperature sensor to Emergency Manager Component regularly (e.g., every 2 seconds), the later is responsible to decide whether the sensor reading indicates a dangerous situation of the patient.

Alert Message: The patient could also send alert message directly through pressing the button. Upon receiving the alert, Gesture Recognizer Component transfers the message to Emergency Manager. Figure 5 shows a screenshot of Gesture Recognizer Component.

2.2 Emergency Manager Component

The Emergency Manager Component is a controller component. It is responsible for processing the alert and reading message sent from the Gesture Recognizer Component. Figure 6 shows a screenshot of the Emergency Manager Component. We see from the figure that the Emergency Manager Component sends Emergency message when the sensor reading exceeds the threshold (normal range is set to $[18.0, 23.0]$ in the example) or the alert message is received.

- Upon receiving "GestureRecognizerAlert" message, Emergency Manager Component notifies Homecare Staff to visit patient

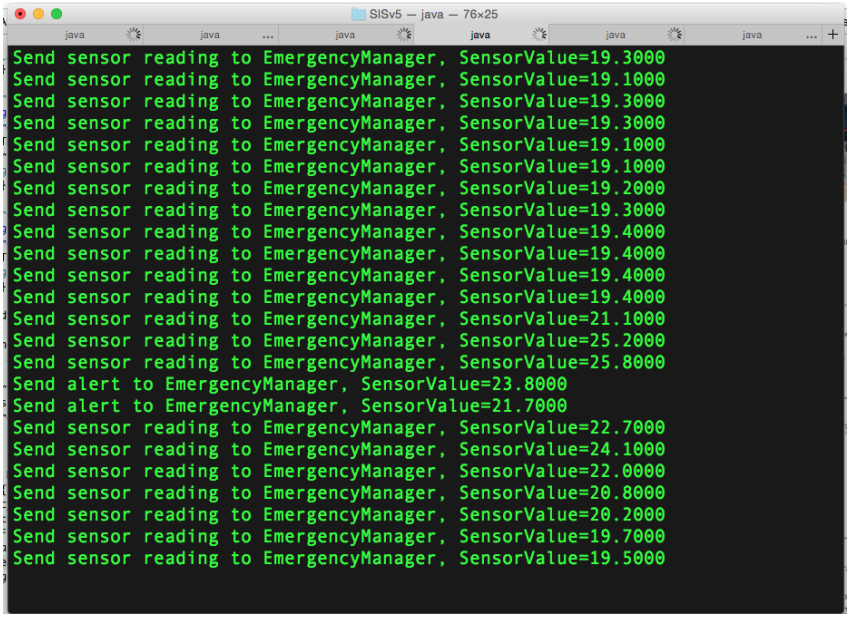

Figure 5: A screenshot of the Gesture Recognizer Component, both alert and reading messages are generated in the example. However, there are generated by different sensors (Button and themistor).

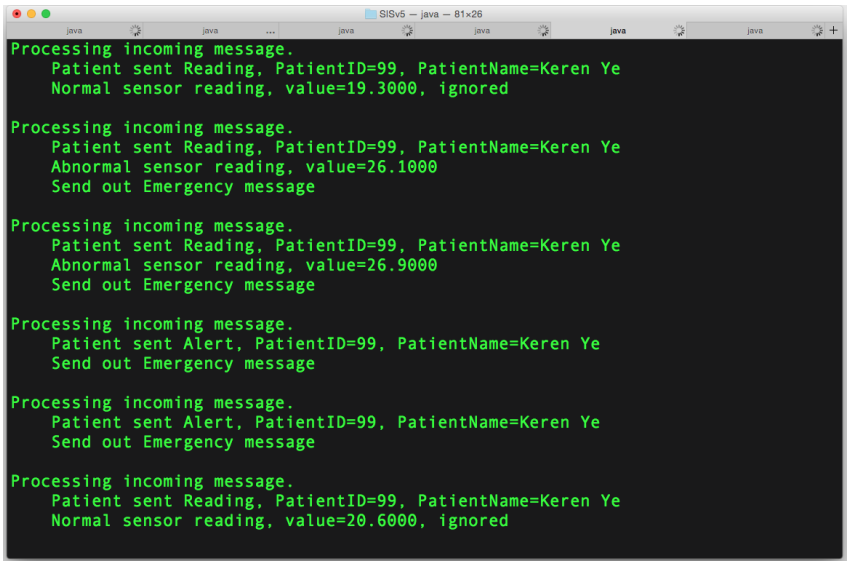

Figure 6: A screenshot of the Emergency Manager Component, the normal range of the temperature is set to $[18.0, 23.0]$.

- Upon receiving "GestureRecognizerReading" message, Emergency Manager compare the sensor reading to a predefined threshold which could be configured using the GUI. It then tells the Homecare Staff to call patient if the sensor reading is abnormal.
- In order to record the sensor reading and alert messages in the database, the Emergency Manager Component also transfers the messages to the Uploader Component.

2.3 Homecare Staff Component

The Homecare Staff Component is an advertiser component which processes and propagates information to the outside world. It has GUI that displays the messages to the employee of homecare staff. Figure 7 shows a screenshot of the Homecare Staff Component.

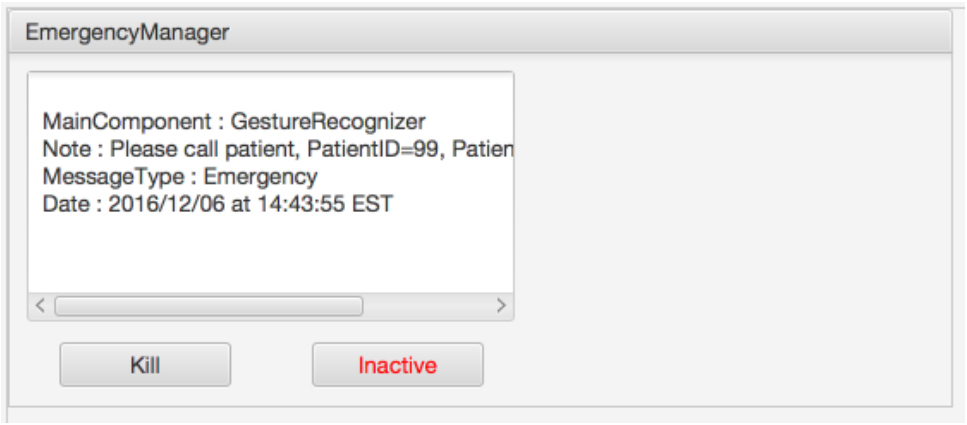

Figure 7: A screenshot of the Homecare Staff Component, it shows that a "Call Patient" message is received.

2.4 Uploader Component

The Uploader Component is also an advertiser component. Upon receiving messages from the Emergency Manager Component, the Uploader records the messages to a database so authorized personnel can access said information. Also, the Uploader Component would send alert messages via email to several predefined email addresses.

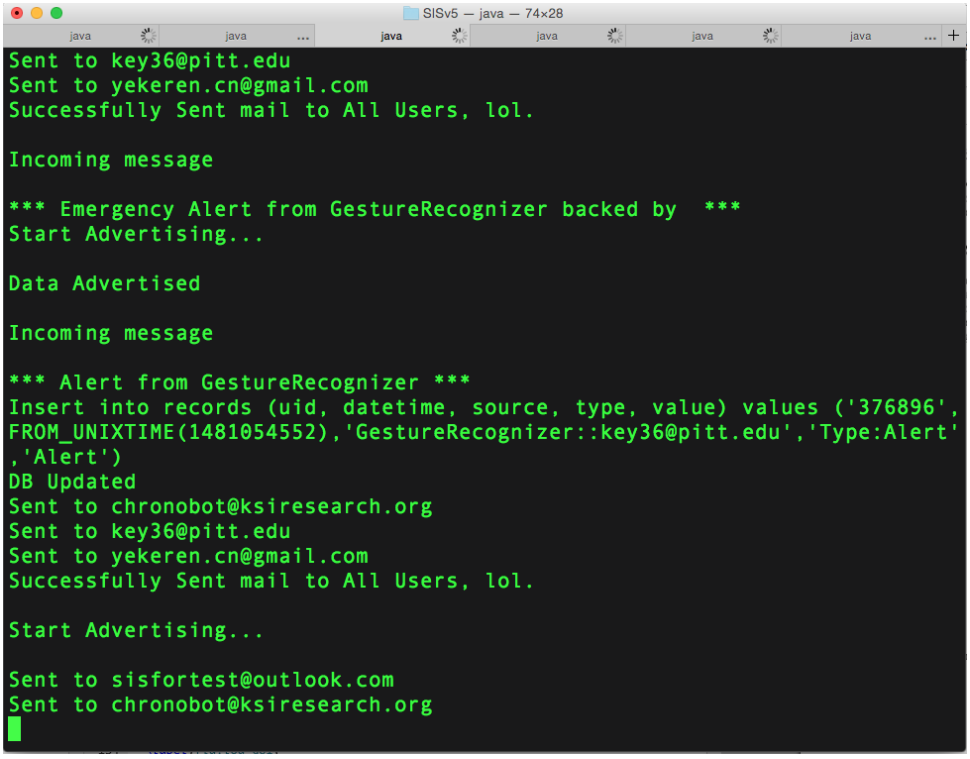

Figure 8: A screenshot of the Uploader Component, it shows that a "Alert" message is recorded.

2.5 GUI Component

The GUI is a predesigned monitor component that monitors all Basic/Controller/Advertiser components within the current scope in the system. All properly registered components have their control panels here. In the project, the GUI component is used for configuring the sampling rate of the temperature sensor in Gesture Recognizer Component and the range of normal temperature in Emergency Manager

Component. Figure 9 shows a screenshot of the GUI Component.

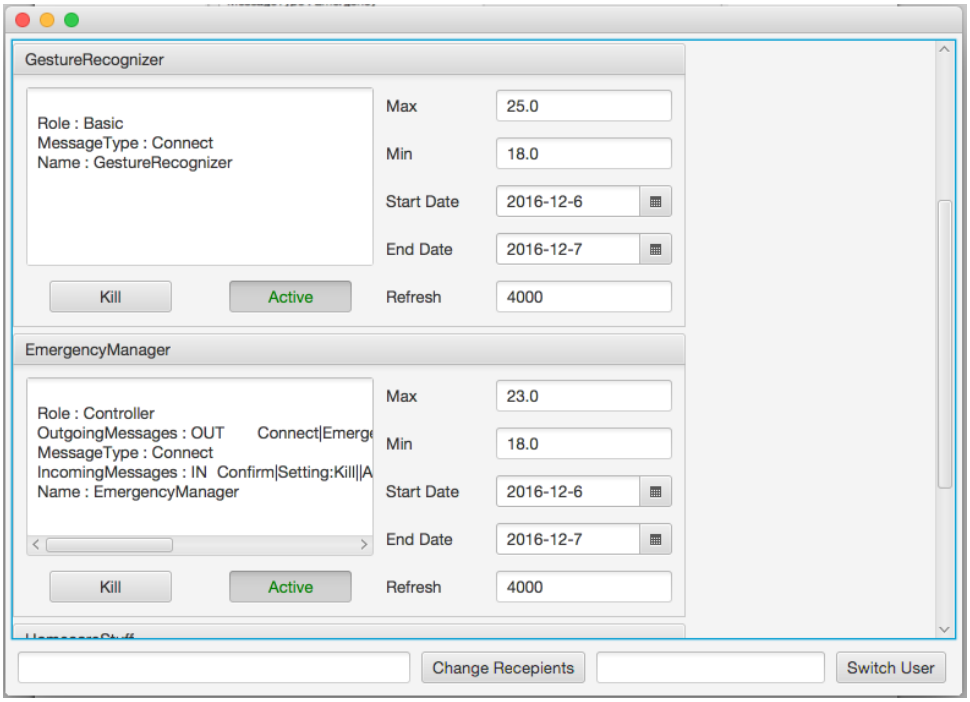

Figure 9: A screenshot of the GUI Component, it shows the control panels of both Gesture Recognizer Component and Emergency Manager Component.

3 System Demo

I have made a demo video for my personal healthcare system and updated it to YouTube. Anyone interested with the project could watch the demo video on YouTube via:

`http://www.youtube.com/watch?v=UeJ6WZZQbJU`

4 Source Code

I have made my source code available online, which can be found here:

`http://people.cs.pitt.edu/~yekeren/cs2310/SISv5_key36.tar.gz`

5 Extra Deeds

- ### I have made both the project demo and the source code available online.
- ### I have applied both thermistor and button as sensors in my project compared to my classmates who use only one.
- ### My design of the system is more complicated than that of exercise 4. It contains five components in total and detailed messages such as setting the sampling rate and setting the normal range using GUI are implemented. Also, it provides graphical interface for homecare staff.
- ### I use the uploader to both records data in database and send email.
- ### Place holder.

6 Appendix

6.1 Source code of Gesture Recognizer Component

6.1.1 CreateGestureRecognizer.java

```
1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileReader;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.net.Socket;
8  import java.net.HttpURLConnection;
9  import java.net.URL;
10 import java.text.ParseException;
11 import java.text.SimpleDateFormat;
12 import java.util.ArrayList;
13 import java.util.Arrays;
14 import java.util.Date;
15 import java.util.List;
16 import java.util.Timer;
17 import java.util.TimerTask;
18
19 public class CreateGestureRecognizer {
20
21     // socket for connection to SISServer
22     static Socket universal;
23     private static int port = 53217;
24     // message writer
25     static MsgEncoder encoder;
26     // message reader
27     static MsgDecoder decoder;
28     // scope of this component
29     private static final String SCOPE = "SIS.Scope1";
30    // name of this component
31     private static final String NAME = "GestureRecognizer";
32     // messages types that can be handled by this component
33     private static final List<String> TYPES = new ArrayList<String>(
34         Arrays.asList(new String[] { "Setting", "Confirm" }));
35
36     private static int refreshRate = 500;
37     private static Timer timer = new Timer();
38
39     // shared by all kinds of records that can be generated by this component
40     private static KeyValueList reading = new KeyValueList();
41     // shared by all kinds of alerts that can be generated by this component
42     private static KeyValueList alert = new KeyValueList();
43
44     private static SimpleDateFormat format = new SimpleDateFormat(
45         "yyyy-MM-dd HH:mm:ss");
46
47     /*
48      * Main program
49      */
50     public static void main(String[] args) {
51         while (true) {
52             try {
53                 // try to establish a connection to SISServer
54                 universal = connect();
55
56                 // bind the message reader to inputstream of the socket
57                 decoder = new MsgDecoder(universal.getInputStream());
58                 // bind the message writer to outputstream of the socket
59                 encoder = new MsgEncoder(universal.getOutputStream());
60
61                 /*
62                  * construct a Connect message to establish the connection
63                  */
64                 KeyValueList conn = new KeyValueList();
65                 conn.putPair("Scope", SCOPE);
```

```
66                conn.putPair("MessageType", "Connect");
67            conn.putPair("Role", "Basic");
68                conn.putPair("Name", NAME);
69                encoder.sendMsg(conn);
70
71                initRecord();
72
73                // KeyValueList for inward messages, see KeyValueList for
74                // details
75                KeyValueList kvList;
76
77                while (true)
78                {
79                    // attempt to read and decode a message, see MsgDecoder for
80                    // details
81                    kvList = decoder.getMsg();
82
83                    // process that message
84                    ProcessMsg(kvList);
85                }
86
87            } catch (Exception e) {
88                e.printStackTrace();
89                try {
90                    Thread.sleep(1000);
91                } catch (InterruptedException e2) {
92                }
93                System.out.println("Try to reconnect");
94                try {
95                    universal = connect();
96                } catch (IOException e1) {
97                }
98            }
99        }
100   }
101
102   /*
103    * used for connect(reconnect) to SISServer
104    */
105   static Socket connect() throws IOException
106   {
107       Socket socket = new Socket("127.0.0.1", port);
108       return socket;
109   }
110
111   private static void initRecord()
112   {
113       reading.putPair("Scope", SCOPE);
114       reading.putPair("MessageType", "Reading");
115       reading.putPair("Sender", NAME);
116       reading.putPair("Receiver", "EmergencyManager");
117       reading.putPair("Purpose", "GestureRecognizerReading");
118       reading.putPair("PatientID", "99");
119       reading.putPair("PatientName", "Keren Ye");
120
121       alert.putPair("Scope", SCOPE);
122       alert.putPair("MessageType", "Alert");
123       alert.putPair("Sender", NAME);
124       alert.putPair("Receiver", "EmergencyManager");
125       alert.putPair("Purpose", "GestureRecognizerAlert");
126       alert.putPair("PatientID", "99");
127       alert.putPair("PatientName", "Keren Ye");
128   }
129
130   public static String getHTML(String urlToRead) throws Exception {
131       StringBuilder result = new StringBuilder();
132       URL url = new URL(urlToRead);
133       HttpURLConnection conn = (HttpURLConnection)url.openConnection();
134       conn.setRequestMethod("GET");
135       BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
136       String line;
137       while ((line = rd.readLine()) != null) {
138           result.append(line);
```

```
        }
        rd.close();
        return result.toString();
    }

    private static SensorReading readSensorValue() throws Exception {
        SensorReading retval = new SensorReading();
        retval.temperature = 0.0;
        retval.emergency = 0;

        try {
            String html = getHTML("http://127.0.0.1:8098");
            String[] array = html.split("\\\t", -1);
            retval.temperature = Double.parseDouble(array[0]);
            retval.emergency= Integer.parseInt(array[1]);
        } catch(Exception e) {
        }
        return retval;
    }

    private static void componentTask()
    {
        try
        {
            SensorReading retval = readSensorValue();

            if (0 == retval.emergency) {
                System.out.printf("Send sensor reading to EmergencyManager, SensorValue=%.4f\n",
                    retval.temperature);
                reading.putPair("SensorValue", "" + retval.temperature);
                encoder.sendMsg(reading);
            } else {
                System.out.printf("Send alert to EmergencyManager, SensorValue=%.4f\n", retval.temperature);
                encoder.sendMsg(alert);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void ProcessMsg(KeyValueList kvList) throws Exception
    {

        String scope = kvList.getValue("Scope");
        if (!SCOPE.startsWith(scope)) {
            return;
        }

        String messageType = kvList.getValue("MessageType");
        if (!TYPES.contains(messageType)) {
            return;
        }

        String sender = kvList.getValue("Sender");
        String receiver = kvList.getValue("Receiver");
        String purpose = kvList.getValue("Purpose");

        switch (messageType)
        {
            case "Confirm":
                System.out.println("Connect to SISServer successful.");
                break;
            case "Setting":
                if (receiver.equals(NAME)) {
                    System.out.println("Message from " + sender);
                    System.out.println("Message type: " + messageType);
                    System.out.println("Message Purpose: " + purpose);

                    switch (purpose) {
                        case "Activate":
                            String refresh_rate = kvList.getValue("RefreshRate");
                            if (refresh_rate != null && !refresh_rate.equals("")) {
                                refreshRate = Integer.parseInt(refresh_rate);
```

```
                        }
                        try {
                            timer.cancel();
                            timer = new Timer();
                        } catch (Exception e) {
                            // TODO: handle exception
                        }
                        timer.schedule(new TimerTask() {
                               public void run() {
                                   componentTask();
                               }
                        }, 0, refreshRate);
                        System.out.println("Algorithm Activated");
                        break;

                    case "Kill":
                        try {
                            timer.cancel();
                        } catch (Exception e) {
                            // TODO: handle exception
                        }
                        System.exit(0);
                        break;

                    case "Deactivate":
                        try {
                            timer.cancel();
                        } catch (Exception e) {
                            // TODO: handle exception
                        }
                        System.out.println("Algorithm Deactivated");
                        break;
                }
            }
            break;
        }
    }
}

class SensorReading {
    double temperature;
    int emergency;
}
```

6.1.2 read_sensor_value.py

```
import sys
from threading import Thread
import SimpleHTTPServer
import SocketServer
import serial

sensor_value = 0

class MyRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        print >> sys.stderr, sensor_value
        self.wfile.write('%s' % (sensor_value))

def threaded_function(arg):
    global sensor_value

    # Start serial listener.
    ser = serial.Serial('/dev/tty.usbmodem14111', 9600)
    while True:
        sensor_value = ser.readline()

thread = Thread(target=threaded_function, args=[0,])
thread.start()
```

```
27
28  # Start http server.
29  PORT = 8098
30  httpd = SocketServer.TCPServer(("", PORT), MyRequestHandler)
31  print "serving at port", PORT
32  httpd.serve_forever()
33
34  thread.join()
```

6.2 Source code of Emergency Manager Component

6.2.1 CreateEmergencyManager.java

```
1   import java.io.BufferedReader;
2   import java.io.File;
3   import java.io.IOException;
4   import java.io.InputStreamReader;
5   import java.io.OutputStream;
6   import java.net.Socket;
7   import java.net.UnknownHostException;
8   import java.util.ArrayList;
9   import java.util.StringTokenizer;
10  import java.util.*;
11  import java.io.*;
12  import java.net.*;
13  import java.lang.*;
14  //import javax.mail.Authenticator;
15  //import javax.mail.Message;
16  //import javax.mail.PasswordAuthentication;
17  //import javax.mail.Session;
18  //import javax.mail.Transport;
19  //import javax.mail.internet.InternetAddress;
20  //import javax.mail.internet.MimeMessage;
21
22  public class CreateEmergencyManager {
23
24      private static double _min = 18, _max = 27;
25     // socket for connection to SISServer
26     private static Socket universal;
27     private static int port = 53217;
28     // message writer
29     private static MsgEncoder encoder;
30     // message reader
31     private static MsgDecoder decoder;
32
33     // scope of this component
34     private static final String SCOPE = "SIS.Scope1";
35     // name of this component
36     private static final String NAME = "EmergencyManager";
37     // messages types that can be handled by this component
38     private static final List<String> TYPES = new ArrayList<String>(
39           Arrays.asList(new String[] { "Setting", "Alert", "Reading", "Confirm" }));
40
41     // summary for all incoming / outgoing messages
42     private static final String incomingMessages =
           "IN\tConfirm|Setting:Kill||Alert:GestureRecognizerAlert|Reading:GestureRecognizerReading";
43     private static final String outgoingMessages = "OUT\t Connect|Emergency";
44
45     // shared by all kinds of emergencies that can be generated by this component
46     private static KeyValueList emergency = new KeyValueList();
47
48     static String alert_msg = "Patient has abnormal bloodPressure because of uncomfortable temperature";
49     static ArrayList<Double> tempRecord = new ArrayList<Double>();
50
51     /*
52      * Main program
53      */
54     public static void main(String[] args) {
55        while (true) {
56           try {
57              // try to establish a connection to SISServer
```

```
58              universal = connect();
59
60              // bind the message reader to inputstream of the socket
61              decoder = new MsgDecoder(universal.getInputStream());
62              // bind the message writer to outputstream of the socket
63              encoder = new MsgEncoder(universal.getOutputStream());
64
65              /*
66               * construct a Connect message to establish the connection
67               */
68              KeyValueList conn = new KeyValueList();
69              conn.putPair("Scope", SCOPE);
70              conn.putPair("MessageType", "Connect");
71              conn.putPair("IncomingMessages", incomingMessages);
72                  conn.putPair("OutgoingMessages", outgoingMessages);
73              conn.putPair("Role", "Controller");
74              conn.putPair("Name", NAME);
75              encoder.sendMsg(conn);
76
77              initRecord();
78
79              // KeyValueList for inward messages, see KeyValueList for
80              // details
81              KeyValueList kvList;
82
83              while (true) {
84                 // attempt to read and decode a message, see MsgDecoder for
85                 // details
86                 kvList = decoder.getMsg();
87
88                 // process that message
89                 ProcessMsg(kvList);
90              }
91
92           } catch (Exception e) {
93              // if anything goes wrong, try to re-establish the connection
94              try {
95                 // wait for 1 second to retry
96                 Thread.sleep(1000);
97              } catch (InterruptedException e2) {
98              }
99              System.out.println("Try to reconnect");
100             try {
101                universal = connect();
102             } catch (IOException e1) {
103             }
104          }
105       }
106    }
107
108    /*
109     * used for connect(reconnect) to SISServer
110     */
111    static Socket connect() throws IOException {
112       Socket socket = new Socket("127.0.0.1", port);
113       return socket;
114    }
115
116    private static void initRecord() {
117
118       emergency.putPair("Scope", SCOPE);
119       emergency.putPair("MessageType", "Emergency");
120       emergency.putPair("Sender", NAME);
121
122       // Receiver may be different for each message, so it doesn't make sense
123       // to set here
124       // alert.putPair("Receiver", "RECEIVER");
125    }
126
127    /*
128     * process a certain message, execute corresponding actions
129     */
130    static void ProcessMsg(KeyValueList kvList) throws IOException {
```

```
131
132         String scope = kvList.getValue("Scope");
133
134         String broadcast = kvList.getValue("Broadcast");
135         String direction = kvList.getValue("Direction");
136
137         if(broadcast!=null&&broadcast.equals("True")){
138
139            if(direction!=null&&direction.equals("Up")){
140               if(!scope.startsWith(SCOPE)){
141                  return;
142               }
143            }else if(direction!=null&&direction.equals("Down")){
144               if(!SCOPE.startsWith(scope)){
145                  return;
146               }
147            }
148         }else{
149            if(!SCOPE.equals(scope)){
150               return;
151            }
152         }
153
154         String messageType = kvList.getValue("MessageType");
155         if(!TYPES.contains(messageType)){
156            return;
157         }
158
159         String sender = kvList.getValue("Sender");
160
161         String receiver = kvList.getValue("Receiver");
162
163         String purpose = kvList.getValue("Purpose");
164
165           System.out.println();
166           System.out.println("Processing incoming message.");
167
168           switch (messageType) {
169               case "Alert":
170                   switch (sender) {
171                       case "GestureRecognizer":
172                           switch (purpose) {
173                               case "GestureRecognizerAlert":
174                                   // Transfer to Homecare Stuff
175                                   String patientID = kvList.getValue("PatientID");
176                                   String patientName = kvList.getValue("PatientName");
177
178                                   System.out.printf("  Patient sent Alert, PatientID=%s, PatientName=%s\n",
                                        patientID, patientName);
179                                   System.out.println("  Send out Emergency message");
180
181                                   emergency.putPair("MainComponent", sender);
182                                   emergency.putPair("Note", String.format("Please visit patient, PatientID=%s,
                                        PatientName=%s", patientID, patientName));
183                                   emergency.putPair("Date", System.currentTimeMillis() + "");
184
185                                   encoder.sendMsg(emergency);
186
187                                   // Transfer to Uploader
188                                   kvList.putPair("Receiver", "Uploader");
189                                   kvList.putPair("Date", System.currentTimeMillis() + "");
190                                   encoder.sendMsg(kvList);
191                                   break;
192                           }
193                   }
194                   break;
195               case "Reading":
196                   switch (sender) {
197                       case "GestureRecognizer":
198                           switch (purpose) {
199                               case "GestureRecognizerReading":
200                                   String patientID = kvList.getValue("PatientID");
201                                   String patientName = kvList.getValue("PatientName");
```

```
202
203                             System.out.printf("  Patient sent Reading, PatientID=%s, PatientName=%s\n",
                                    patientID, patientName);
204
205                             String sensorValue = kvList.getValue("SensorValue");
206                             double sensorReading = 0;
207                             try {
208                                 sensorReading = Double.parseDouble(sensorValue);
209                             } catch (Exception e) {
210                                 sensorReading = 0;
211                             }
212
213                             if (sensorReading < _min || sensorReading > _max) {
214                                 // Transfer to Homecare Stuff
215                                 System.out.printf("  Abnormal sensor reading, value=%.4f\n",
                                        sensorReading);
216                                 System.out.println("  Send out Emergency message");
217
218                                 emergency.putPair("MainComponent", sender);
219                                 emergency.putPair("Note", String.format("Please call patient,
                                        PatientID=%s, PatientName=%s", patientID, patientName));
220                                 emergency.putPair("Date", System.currentTimeMillis() + "");
221
222                                 encoder.sendMsg(emergency);
223
224                                 // Transfer to Uploader
225                                 kvList.putPair("Receiver", "Uploader");
226                                 kvList.putPair("Date", System.currentTimeMillis() + "");
227                                 encoder.sendMsg(kvList);
228                             } else {
229                                 System.out.printf("  Normal sensor reading, value=%.4f, ignored\n",
                                        sensorReading);
230                             }
231                             break;
232                     }
233                 }
234                 break;
235             case "Confirm":
236                 System.out.println("Connect to SISServer successful.");
237                 break;
238             case "Setting":
239                 if (receiver.equals(NAME)) {
240                     switch (purpose) {
241                         case "Kill":
242                             System.exit(0);
243                             break;
244                         case "Activate":
245                             String strMax = kvList.getValue("Max");
246                             String strMin = kvList.getValue("Min");
247                             if (strMax != null && !strMax.equals("")) {
248                                 _max = Double.parseDouble(strMax);
249                             }
250                             if (strMin != null && !strMin.equals("")) {
251                                 _min = Double.parseDouble(strMin);
252                             }
253                             System.out.println(strMax);
254                             System.out.println(strMin);
255                             break;
256                     }
257                 }
258                 break;
259         }
260     }
261
262
263     // The following function is used to calculate the slope of temperature
264     public static double calculateSlope() throws IOException
265     {
266         String fileName = "temperatureRecord.csv";
267         BufferedReader br = new BufferedReader(new FileReader(fileName));
268
269         double tempSlope = 0.0;
270
```

```
271         String temp;
272         while((temp = br.readLine()) != null)
273         {
274             String[] str = temp.split(",");
275             if(tempRecord.size() <= 20)
276             {
277                 double newEntry = Double.parseDouble(str[1]);
278                 tempRecord.add(newEntry);
279             }
280             else
281             {
282                 tempRecord.remove(0);
283                 double newEntry = Double.parseDouble(str[1]);
284                 tempRecord.add(newEntry);
285             }
286         }
287         System.out.println("");
288         System.out.print("The twenty minute temperature record are as follows: ");
289         System.out.println(tempRecord);
290         System.out.println("");
291         Collections.sort(tempRecord);
292         double min = tempRecord.get(0);
293         double max = tempRecord.get(tempRecord.size() - 1);
294         tempSlope = (max - min) / max;
295         System.out.printf("tempSlope is %.2f", tempSlope);
296         System.out.println("n");
297 
298         return tempSlope;
299     }
300 }
```

6.3 Source code of Homecare Staff Component

6.3.1 CreateHomecareStuff.java

```
1 
2  import java.io.IOException;
3  import java.net.Socket;
4  import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.List;
7 
8  import javafx.application.Application;
9  import javafx.application.Platform;
10 import javafx.beans.value.ChangeListener;
11 import javafx.beans.value.ObservableValue;
12 import javafx.collections.FXCollections;
13 import javafx.collections.ObservableMap;
14 import javafx.geometry.Bounds;
15 import javafx.scene.Scene;
16 import javafx.scene.control.ScrollPane;
17 import javafx.stage.Stage;
18 
19 public class CreateHomecareStuff extends Application {
20 
21    private ObservableMap<String, KeyValueList> map = FXCollections
22          .observableHashMap();
23 
24    private Proc pro = new Proc(map);
25 
26    public static final String SCOPE = "SIS.Scope1";
27    // name of this component
28     public static final String NAME = "HomecareStuff";
29 
30    public void start(Stage primaryStage) {
31       try {
32          SISFlow root = new SISFlow(map);
33          //ScrollPane root = new ScrollPane();
34          Scene scene = new Scene(root);
35 
36          primaryStage.setScene(scene);
```

```
37          primaryStage.setMinWidth(515);
38          primaryStage.setMinHeight(500);
39          primaryStage.show();
40       } catch (Exception e) {
41          e.printStackTrace();
42       }
43    }
44
45    public void stop() throws Exception {
46       // TODO Auto-generated method stub
47       super.stop();
48       pro.close();
49    }
50
51    public static void main(String[] args) {
52
53       launch(args);
54    }
55 }
56
57 class MonitorTask implements Runnable {
58    private volatile boolean running = true;
59
60    // socket for connection to SISServer
61    private Socket universal;
62    private static int port = 53217;
63    // message writer
64    private MsgEncoder encoder;
65    // message reader
66    private MsgDecoder decoder;
67    // scope of this component
68    //private final String SCOPE = "SIS.Scope1";
69    // messages types that can be handled by this component
70    private final List<String> TYPES = new ArrayList<String>(
71          Arrays.asList(new String[] { "Emergency", "Confirm"}));
72
73    private ObservableMap<String, KeyValueList> map;
74
75    public MonitorTask(ObservableMap<String, KeyValueList> m) {
76       map = m;
77    }
78
79    public void terminate() {
80       running = false;
81       try {
82          universal.close();
83       } catch (IOException e) {
84          // TODO Auto-generated catch block
85          System.out.println("Close the connection");
86          // e.printStackTrace();
87       }
88    }
89
90    public void run() {
91       // TODO Auto-generated method stub
92       runTask();
93    }
94
95    public void runTask() {
96       while (running) {
97          try {
98             // try to establish a connection to SISServer
99             universal = connect();
100
101            // bind the message reader to inputstream of the socket
102            decoder = new MsgDecoder(universal.getInputStream());
103            // bind the message writer to outputstream of the socket
104            encoder = new MsgEncoder(universal.getOutputStream());
105
106            /*
107             * construct a Connect message to establish the connection
108             */
109            KeyValueList conn = new KeyValueList();
```

```
110             conn.putPair("Scope", CreateHomecareStuff.SCOPE);
111             conn.putPair("MessageType", "Connect");
112             conn.putPair("Role", "Advertizer");
113             conn.putPair("Name", CreateHomecareStuff.NAME);
114             encoder.sendMsg(conn);
115
116             // KeyValueList for inward messages, see KeyValueList for
117             // details
118
119             while (running) {
120                ProcessMsg(decoder.getMsg());
121             }
122
123          } catch (Exception e) {
124             // if anything goes wrong, try to re-establish the connection
125             // e.printStackTrace();
126             if (running) {
127                try {
128                   // wait for 1 second to retry
129                   Thread.sleep(1000);
130                } catch (InterruptedException e2) {
131                }
132                System.out.println("Try to reconnect");
133                try {
134                   universal = connect();
135                } catch (IOException e1) {
136                }
137             }
138          }
139       }
140    }
141
142    /*
143     * used for connect(reconnect) to SISServer
144     */
145    private Socket connect() throws IOException {
146       Socket socket = new Socket("127.0.0.1", port);
147       return socket;
148    }
149
150    private void ProcessMsg(KeyValueList kvList) throws Exception {
151
152       String scope = kvList.getValue("Scope");
153       if (!CreateHomecareStuff.SCOPE.startsWith(scope)) {
154          return;
155       }
156
157       String messageType = kvList.getValue("MessageType");
158       if (!TYPES.contains(messageType)) {
159          return;
160       }
161
162       String sender = kvList.getValue("Sender");
163
164       String receiver = kvList.getValue("Receiver");
165
166       String purpose = kvList.getValue("Purpose");
167
168       String date = kvList.getValue("Date");
169
170         System.out.println();
171         System.out.println("Processing incoming message.");
172
173         switch (messageType) {
174             case "Emergency":
175                 String note = kvList.getValue("Note");
176                 System.out.printf("  Notice: %s\n", note);
177
178                 Platform.runLater(new Runnable() {
179                        public void run() {
180                        // if you change the UI, do it here !
181                        kvList.removePair("Scope");
182                        map.put(sender, kvList);
```

```
                    }
                    });
                break;

            case "Confirm":
                System.out.println("Connect to SISServer successful.");
                break;
        }
    }
}

class Proc {

    private Thread thread;

    private MonitorTask monitorTask;

    public Proc(ObservableMap<String, KeyValueList> m) {
        monitorTask = new MonitorTask(m);
        thread = new Thread(monitorTask);
        thread.start();
    }

    public void close() {

        try {
            monitorTask.terminate();
            thread.join();
            // universal.shutdownInput();
            // universal.shutdownOutput();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

6.4 Source code of GUI Component

6.4.1 CreateGUI.java

```

import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableMap;
import javafx.geometry.Bounds;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.stage.Stage;

public class CreateGUI extends Application {

    private ObservableMap<String, KeyValueList> map = FXCollections
          .observableHashMap();

    private Proc pro = new Proc(map);

    public static final String SCOPE = "SIS.Scope1";
    // name of this component
     public static final String NAME = "GUI";
```

```
29
30      public void start(Stage primaryStage) {
31         try {
32            SISFlow root = new SISFlow(map);
33            //ScrollPane root = new ScrollPane();
34            Scene scene = new Scene(root);
35
36            primaryStage.setScene(scene);
37            primaryStage.setMinWidth(515);
38            primaryStage.setMinHeight(500);
39            primaryStage.show();
40         } catch (Exception e) {
41            e.printStackTrace();
42         }
43      }
44
45      public void stop() throws Exception {
46         // TODO Auto-generated method stub
47         super.stop();
48         pro.close();
49      }
50
51      public static void main(String[] args) {
52
53         launch(args);
54      }
55   }
56
57   class MonitorTask implements Runnable {
58      private volatile boolean running = true;
59
60      // socket for connection to SISServer
61      private Socket universal;
62      private static int port = 53217;
63      // message writer
64      private MsgEncoder encoder;
65      // message reader
66      private MsgDecoder decoder;
67      // scope of this component
68      //private final String SCOPE = "SIS.Scope1";
69      // messages types that can be handled by this component
70      private final List<String> TYPES = new ArrayList<String>(
71            Arrays.asList(new String[] { "Reading", "Alert", "Confirm",
72                  "Connect" }));
73
74      private ObservableMap<String, KeyValueList> map;
75
76      public MonitorTask(ObservableMap<String, KeyValueList> m) {
77         map = m;
78      }
79
80      public void terminate() {
81         running = false;
82         try {
83            universal.close();
84         } catch (IOException e) {
85            // TODO Auto-generated catch block
86            System.out.println("Close the connection");
87            // e.printStackTrace();
88         }
89      }
90
91      public void run() {
92         // TODO Auto-generated method stub
93         runTask();
94      }
95
96      public void runTask() {
97         while (running) {
98            try {
99               // try to establish a connection to SISServer
100              universal = connect();
101
```

```
102             // bind the message reader to inputstream of the socket
103             decoder = new MsgDecoder(universal.getInputStream());
104             // bind the message writer to outputstream of the socket
105             encoder = new MsgEncoder(universal.getOutputStream());
106 
107             /*
108              * construct a Connect message to establish the connection
109              */
110             KeyValueList conn = new KeyValueList();
111             conn.putPair("Scope", CreateGUI.SCOPE);
112             conn.putPair("MessageType", "Connect");
113             conn.putPair("Role", "Monitor");
114             conn.putPair("Name", CreateGUI.NAME);
115             encoder.sendMsg(conn);
116 
117             // KeyValueList for inward messages, see KeyValueList for
118             // details
119 
120             while (running) {
121                ProcessMsg(decoder.getMsg());
122             }
123 
124          } catch (Exception e) {
125             // if anything goes wrong, try to re-establish the connection
126             // e.printStackTrace();
127             if (running) {
128                try {
129                   // wait for 1 second to retry
130                   Thread.sleep(1000);
131                } catch (InterruptedException e2) {
132                }
133                System.out.println("Try to reconnect");
134                try {
135                   universal = connect();
136                } catch (IOException e1) {
137                }
138             }
139          }
140       }
141    }
142 
143    /*
144     * used for connect(reconnect) to SISServer
145     */
146    private Socket connect() throws IOException {
147       Socket socket = new Socket("127.0.0.1", port);
148       return socket;
149    }
150 
151    private void ProcessMsg(KeyValueList kvList) throws Exception {
152 
153       String scope = kvList.getValue("Scope");
154       if (!CreateGUI.SCOPE.startsWith(scope)) {
155          return;
156       }
157 
158       String messageType = kvList.getValue("MessageType");
159       if (!TYPES.contains(messageType)) {
160          return;
161       }
162 
163       String sender = kvList.getValue("Sender");
164 
165       String receiver = kvList.getValue("Receiver");
166 
167       String purpose = kvList.getValue("Purpose");
168 
169       switch (messageType) {
170       case "Connect":
171          String name = kvList.getValue("Name");
172          String role = kvList.getValue("Role");
173          if (!name.equals(CreateGUI.NAME)&&!role.equals("Monitor")) {
174             Platform.runLater(new Runnable() {
```

```
175                public void run() {
176                   // if you change the UI, do it here !
177                   kvList.removePair("Scope");
178                   map.put(name, kvList);
179                }
180             });
181          }
182          break;
183       case "Alert":
184       case "Reading":
185
186          Platform.runLater(new Runnable() {
187             public void run() {
188                // if you change the UI, do it here !
189                kvList.removePair("Scope");
190                map.put(sender, kvList);
191             }
192          });
193
194          break;
195       case "Confirm":
196          System.out.println("Connect to SISServer successful.");
197          break;
198       }
199    }
200 }
201
202 class Proc {
203
204    private Thread thread;
205
206    private MonitorTask monitorTask;
207
208    public Proc(ObservableMap<String, KeyValueList> m) {
209       monitorTask = new MonitorTask(m);
210       thread = new Thread(monitorTask);
211       thread.start();
212    }
213
214    public void close() {
215
216       try {
217          monitorTask.terminate();
218          thread.join();
219          // universal.shutdownInput();
220          // universal.shutdownOutput();
221       } catch (InterruptedException e) {
222          // TODO Auto-generated catch block
223          e.printStackTrace();
224       }
225
226    }
227
228 }
```

6.5 Source code of Uploader

6.5.1 CreateUploader.java

```
1  import java.io.IOException;
2  import java.net.Socket;
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.List;
6  import java.util.Properties;
7  import java.util.Calendar;
8
9  import javax.mail.BodyPart;
10 import javax.mail.Message;
11 import javax.mail.MessagingException;
12 import javax.mail.Multipart;
```

```
13  import javax.mail.PasswordAuthentication;
14  import javax.mail.Session;
15  import javax.mail.Transport;
16  import javax.mail.internet.InternetAddress;
17  import javax.mail.internet.MimeBodyPart;
18  import javax.mail.internet.MimeMessage;
19  import javax.mail.internet.MimeMultipart;
20
21  public class CreateUploader
22  {
23      // socket for connection to SISServer
24      static Socket universal;
25      private static int port = 53217;
26      // message writer
27      static MsgEncoder encoder;
28      // message reader
29      static MsgDecoder decoder;
30
31      // scope of this component
32      private static final String SCOPE = "SIS.Scope1";
33     // name of this component
34      private static final String NAME = "Uploader";
35      // messages types that can be handled by this component
36      private static final List<String> TYPES = new ArrayList<String>(
37          Arrays.asList(new String[] { "Alert", "Reading", "Emergency", "Confirm", "Setting" }));
38
39      private static UploaderReading reading = new UploaderReading();
40
41      // variables for sending emails
42      static final String SMTP_HOST_NAME = "smtp.ksiresearch.org.ipage.com";
43      static final String SMTP_PORT = "587";
44      static final String emailMsgTxt = "Test Message Contents";
45      static final String emailSubjectTxt = "Personal Healthcare Data From SIS System."; // title
46      static final String emailFromAddress = "chronobot at ksiresearch.org";
47      static final String SSL_FACTORY = "javax.net.ssl.SSLSocketFactory";
48
49      /*
50       * Main program
51       */
52      public static void main(String[] args)
53      {
54          while (true)
55          {
56              try
57              {
58                  // try to establish a connection to SISServer
59                  universal = connect();
60
61                  // bind the message reader to inputstream of the socket
62                  decoder = new MsgDecoder(universal.getInputStream());
63                  // bind the message writer to outputstream of the socket
64                  encoder = new MsgEncoder(universal.getOutputStream());
65
66                  /*
67                   * construct a Connect message to establish the connection
68                   */
69                  KeyValueList conn = new KeyValueList();
70                  conn.putPair("Scope", SCOPE);
71                  conn.putPair("MessageType", "Connect");
72                  conn.putPair("Role", "Advertiser");
73                  conn.putPair("Name", NAME);
74                  encoder.sendMsg(conn);
75
76                  // KeyValueList for inward messages, see KeyValueList for
77                  // details
78                  KeyValueList kvList;
79
80                  update();
81
82                  while (true)
83                  {
84                      // attempt to read and decode a message, see MsgDecoder for
85                      // details
```

```
86                  kvList = decoder.getMsg();
87
88                  // process that message
89                  ProcessMsg(kvList);
90              }
91
92          }
93          catch (Exception e)
94          {
95              // if anything goes wrong, try to re-establish the connection
96              e.printStackTrace();
97              try
98              {
99                  // wait for 1 second to retry
100                 Thread.sleep(1000);
101             }
102             catch (InterruptedException e2)
103             {
104             }
105             System.out.println("Try to reconnect");
106             try
107             {
108                 universal = connect();
109             }
110             catch (IOException e1)
111             {
112             }
113         }
114     }
115 }
116
117 /*
118  * used for connect(reconnect) to SISServer
119  */
120 static Socket connect() throws IOException
121 {
122     Socket socket = new Socket("127.0.0.1", port);
123     return socket;
124 }
125
126 /*
127  * Method for sending email which contains the information which GUI shows
128  * on screen.
129  */
130 static void sendSSLMessage(String from, String recipients[],
131                            String subject, String message) throws MessagingException
132 {
133     boolean debug = false;
134     Properties props = new Properties();
135     props.put("mail.smtp.host", SMTP_HOST_NAME);
136     props.put("mail.smtp.auth", "true");
137     props.put("mail.debug", "true");
138     props.put("mail.smtp.port", SMTP_PORT);
139     props.put("mail.smtp.socketFactory.port", SMTP_PORT);
140     props.put("mail.smtp.socketFactory.class", SSL_FACTORY);
141     props.put("mail.smtp.socketFactory.fallback", "true");
142
143     Session session = Session.getDefaultInstance(props,
144                     new javax.mail.Authenticator()
145     {
146         protected PasswordAuthentication getPasswordAuthentication()
147         {
148             return new PasswordAuthentication(
149                       "chronobot at ksiresearch.org", "Health14");
150         }
151     });
152     session.setDebug(debug);
153
154     Message msg = new MimeMessage(session);
155     InternetAddress addressFrom = new InternetAddress(from);
156     msg.setFrom(addressFrom);
157
158     InternetAddress[] addressTo = new InternetAddress[recipients.length];
```

```
159         for (int i = 0; i < recipients.length; i++)
160         {
161             addressTo[i] = new InternetAddress(recipients[i]);
162             System.out.println("Sent to " + recipients[i]);
163         }
164         msg.setRecipients(Message.RecipientType.TO, addressTo);
165         msg.setSubject(subject);
166         {
167             Multipart multipart = new MimeMultipart("related");
168             {
169                 Multipart newMultipart = new MimeMultipart("alternative");
170                 BodyPart nestedPart = new MimeBodyPart();
171                 nestedPart.setContent(newMultipart);
172                 multipart.addBodyPart(nestedPart);
173                 {
174                     BodyPart part = new MimeBodyPart();
175                     part.setText("SIS DATA:");
176                     newMultipart.addBodyPart(part);
177 
178                     part = new MimeBodyPart();
179                     // the first string is email context
180 
181                     part.setContent(message, "text/html");
182                     //    "Here is the current status of the patient "
183                     //    + "(This is an automatic massage send from SIS system): "
184                     //    + "<br>LastName: Ye"
185                     //    + "<br>FirstName: Keren"
186                     //    + "<br>SPO2: "
187                     //    + reading.spo2
188                     //    + "<br>Systolic: "
189                     //    + reading.systolic
190                     //    + "<br>Diastolic: "
191                     //    + reading.diastolic
192                     //    + "<br>Pulse: "
193                     //    + reading.pulse
194                     //    + "<br>Date of Blood Pressure: "
195                     //    + reading.dateBP
196                   // + "<br>SPO2: "
197                     //    + reading.spo2
198                     //    + "<br>Date of SPO2: "
199                     //    + reading.dateSPO2
200                     //    + "<br>EKG: "
201                     //    + reading.ekg
202                     //    + "<br>Date of EKG: "
203                     //    + reading.dateEKG
204                     //    /*+ "<br>Temperature: "
205                     //    + reading.temp
206                     //    + "<br>Date of Temperature: "
207                     //    + reading.dateTemp
208                     //    + "<br>Kinect Status: "*/
209                     //    + reading.kinectStatus
210                     //    + "<br>Date of Kinect: "
211                     //    + reading.dateKinect, "text/html");
212                     newMultipart.addBodyPart(part);
213                 }
214             }
215             /*
216              * BodyPart part = new MimeBodyPart();
217              * part.setText(
218              * "Here is the SPO2 and Blood Pressure data(This is an automatic massage send from SIS
                     system): LastName: "
219              * + lname
220              * + " FirstName: "
221              * + fname
222              * + "\nSPO2: "
223              * + spo2
224              * + " "
225              * + "\nSystolic: "
226              * + systolic + "\nDiastolic: " + diastolic + "\nPulse: " + pulse);
227              * multipart.addBodyPart(part);
228              */
229             msg.setContent(multipart);
230         }
```

```
231         Transport.send(msg);
232         System.out.println("Successfully Sent mail to All Users, lol.\n");
233     }
234 
235     /*
236      * Method for sending email for Alert Message
237      */
238     static void sendAlertSSLMessage(String from, String recipients[],
239                                     String subject, String message) throws MessagingException
240     {
241     }
242 
243     // ============= end of sending email ====================
244 
245     private static void ProcessMsg(KeyValueList kvList) throws Exception
246     {
247 
248         System.out.println("Incoming message");
249         String scope = kvList.getValue("Scope");
250         if (!SCOPE.startsWith(scope))
251         {
252             return;
253         }
254 
255         String messageType = kvList.getValue("MessageType");
256         if (!TYPES.contains(messageType))
257         {
258             return;
259         }
260 
261         String sender = kvList.getValue("Sender");
262 
263         String receiver = kvList.getValue("Receiver");
264 
265         String purpose = kvList.getValue("Purpose");
266 
267         String strSubject = "Data From EmergencyManager - Keren Ye(key36 at pitt.edu).";
268         String[] recipients = new String[3];
269         recipients[0] = "chronobot at ksiresearch.org";
270         recipients[1] = "key36 at pitt.edu";
271         recipients[2] = "yekeren.cn at gmail.com";
272         switch (messageType)
273         {
274         case "Reading":
275             System.out.println("\n*** Reading from "+sender+" ***");
276             switch (sender) {
277 
278             case "GestureRecognizer":
279                 String strPatientID = kvList.getValue("PatientID");
280                 String strPatientName = kvList.getValue("PatientName");
281                 String strDate = kvList.getValue("Date");
282                 String strSensorValue = kvList.getValue("SensorValue");
283 
284                 long date = 0;
285                 if (strDate != null && !strDate.equals("")) {
286                     date = Long.parseLong(strDate);
287                 }
288                 double sensorValue = 0;
289                 if (strSensorValue != null && !strSensorValue.equals("")) {
290                     sensorValue = Double.parseDouble(strSensorValue);
291                 }
292                 //String strType = "Type:Reading,PatientID:"+strPatientID+",PatientName:"+strPatientName;
293                 String strType = "Type:Reading";
294                 execute(formQuery(date,sender+"::key36 at pitt.edu",strType,Double.toString(sensorValue)));
295 
296                 System.out.println("DB Updated");
297 
298                 sendSSLMessage("key36 at pitt.edu", recipients, strSubject, "We receive an abnormal
                        reading, please call patient to verify.</br>" + "SensorReading: " +
                        Double.toString(sensorValue));
299 
300                 break;
301 
```

```
302             }
303             break;
304 
305         case "Alert":
306             System.out.println("\n*** Alert from "+sender+" ***");
307             switch (sender)
308             {
309             case "GestureRecognizer":
310                 String strPatientID = kvList.getValue("PatientID");
311                 String strPatientName = kvList.getValue("PatientName");
312                 String strDate = kvList.getValue("Date");
313 
314                 long date = 0;
315                 if (strDate != null && !strDate.equals("")) {
316                     date = Long.parseLong(strDate);
317                 }
318                 //String strType = "Type:Alert,PatientID:"+strPatientID+",PatientName:"+strPatientName;
319                 String strType = "Type:Alert";
320                 execute(formQuery(date,sender+"::key36 at pitt.edu",strType,"Alert"));
321 
322                 System.out.println("DB Updated");
323                 //static void sendSSLMessage(String from, String recipients[],
324                 //                           String subject, String message) throws MessagingException
325 
326                 sendSSLMessage("key36 at pitt.edu", recipients, strSubject, "We receive an alert message,
                        please visit patient immediately.");
327 
328                 break;
329             case "BloodPressure":
330 
331                 String sys = kvList.getValue("Systolic");
332                 String dia = kvList.getValue("Diastolic");
333                 String pul = kvList.getValue("Pulse");
334                 String datBP = kvList.getValue("Date");
335 
336                 if (sys != null && !sys.equals(""))
337                 {
338                     reading.systolic = Integer.parseInt(sys);
339                 }
340                 if (dia != null && !dia.equals(""))
341                 {
342                     reading.diastolic = Integer.parseInt(dia);
343                 }
344                 if (pul != null && !pul.equals(""))
345                 {
346                     reading.pulse = Integer.parseInt(pul);
347                 }
348                 if (datBP != null && !datBP.equals(""))
349                 {
350                     reading.dateBP = Long.parseLong(datBP);
351                 }
352                 break;
353             case "EKG":
354 
355                 break;
356             case "SPO2":
357                 String spo = kvList.getValue("SPO2");
358                 String datSP = kvList.getValue("Date");
359 
360                 if (spo != null && !spo.equals(""))
361                 {
362                     reading.spo2 = Integer.parseInt(spo);
363                 }
364                 if (datSP != null && !datSP.equals(""))
365                 {
366                     reading.dateSPO2 = Long.parseLong(datSP);
367                 }
368                 break;
369             /*case "Temp":
370                 String tem = kvList.getValue("Temp");
371                 String datTe = kvList.getValue("Date");
372 
373                 if (tem != null && !tem.equals(""))
```

```
                {
                    reading.temp = Double.parseDouble(tem);
                }
                if (datTe != null && !datTe.equals(""))
                {
                    reading.dateTemp = Long.parseLong(datTe);
                }
                break;*/
            case "KinectMonitor":
                String sta = kvList.getValue("Status");
                String datKi = kvList.getValue("Date");

                if (sta != null && !sta.equals(""))
                {
                    reading.kinectStatus = sta;
                }
                if (datKi != null && !datKi.equals(""))
                {
                    reading.dateKinect = Long.parseLong(datKi);
                }
                break;
            }
            System.out.println("Start Advertising...\n");
            update();
            sendSSLMessage(emailFromAddress, reading.recipients, emailSubjectTxt,
                      emailMsgTxt);
            System.out.println("Data Advertised\n");
            break;

        case "Emergency":
            String sup = kvList.getValue("MainComponent");
            String auxs = kvList.getValue("HelperComponents");
            String note = kvList.getValue("Note");

            System.out.println("\n*** Emergency Alert from " + sup + " backed by " + auxs+" ***");

            System.out.println("Start Advertising...\n");
            update();
            sendAlertSSLMessage(emailFromAddress, reading.recipients,
                              "Emergency Alert from " + sup + " backed by " + auxs, note);
            System.out.println("Data Advertised\n");
            break;

        case "Confirm":
            System.out.println("Connect to SISServer successful.");
            break;
        case "Setting":
            if (receiver.equals(NAME))
            {
                System.out.println("Message from " + sender);
                System.out.println("Message type: " + messageType);
                System.out.println("Message Purpose: " + purpose);
                switch (purpose)
                {
                case "SwitchUser":
                    String user = kvList.getValue("UserID");
                    reading.uid = user;
                    System.out.println("User Switched: "+user);
                    switcha();
                    break;
                case "UpdateRecipients":
                    String recs = kvList.getValue("Recipients");
                    reading.recipients = recs.replaceAll("\\s+", "").split(",",0);
                  System.out.println("Recipients Updated: "+Arrays.toString(reading.recipients)+"
                    "+reading.recipients.length);
                    break;
                case "ForceUpload":
                    update();
                    break;

                case "Kill":
                    System.exit(0);
                    break;
```

```
                }
            }
            break;
        }
    }

    private static void execute(String query) throws Exception {
        String url = "http://ksiresearch.org/chronobot/PHP_Post.php";
        PostQuery.PostToPHP(url, query);
    }

    private static String formQuery(long datetime, String source, String type, Object value){
        String sql = "Insert into records (uid, datetime, source, type, value) values ('"
                        + reading.uid
                        + "',"
                        + "FROM_UNIXTIME("+datetime/1000+")"
                        + ",'"
                        + source
                        + "','"
                        + type
                        + "','"
                        + value.toString()
                        + "')";
        System.out.println(sql);
        return sql;
    }

    private static void switcha() throws Exception{
        // uid ip password email firstname lastname
        System.out.println("User Log in...");
        String query =
            "UPDATE users " +
            "SET ip='" + NetTool.getPublicAddress() + "' " +
            "WHERE uid='" + reading.uid + "'";
        execute(query);
        System.out.println("User Logged In");
    }

    private static void update() throws Exception
    {
        return;
        //System.out.println(reading);

        //System.out.println("Updating DB...");

        //execute(formQuery(reading.dateSPO2,"SPO2","spo2",reading.spo2));
        //execute(formQuery(reading.dateBP,"BloodPressure","systolic",reading.systolic));
        //execute(formQuery(reading.dateBP,"BloodPressure","diastolic",reading.diastolic));
        //execute(formQuery(reading.dateBP,"BloodPressure","pulse",reading.pulse));
        //execute(formQuery(reading.dateEKG,"EKG","ekg",reading.ekg));
        ////execute(formQuery(reading.dateTemp,"Temperature","temp",reading.temp));
        //
        //System.out.println("DB Updated");
        //// sendSSLMessage(sendTo, emailSubjectTxt, emailMsgTxt,
        //// emailFromAddress, lname, fname, spo2);
    }
}

class UploaderReading
{
    String uid = "376896";
    String firstName = "Keren";
    String lastName = "Ye";
    String[] recipients = { "sisfortest at outlook.com",
                            "chronobot at ksiresearch.org"
                          };
    int spo2;
    int systolic;
    int diastolic;
    int pulse;
    int ekg;
    long dateBP;
    long dateEKG;
```

```
519     long dateSPO2;
520
521     //double temp;
522     //long dateTemp;
523     String kinectStatus;
524     long dateKinect;
525
526     public String toString()
527     {
528         // TODO Auto-generated method stub
529         StringBuilder builder = new StringBuilder();
530         builder.append("--------------------------------------------\n");
531         builder.append("First Name: " + firstName + "\n");
532         builder.append("Last Name: " + lastName + "\n");
533         builder.append("Email: " + Arrays.toString(recipients) + "\n\n");
534         builder.append("Systolic: " + systolic + "\n");
535         builder.append("Diastolic: " + diastolic + "\n");
536         builder.append("Pulse: " + pulse + "\n");
537         builder.append("Date (Blood Pressure): " + dateBP + "\n\n");
538         builder.append("EKG: " + ekg + "\n");
539         builder.append("Date (EKG): " + dateEKG + "\n\n");
540         builder.append("SPO2: " + spo2 + "\n");
541         builder.append("Date (SPO2): " + dateSPO2 + "\n\n");
542         //builder.append("Temperature: " + temp + "\n");
543         //builder.append("Date (Temperature): " + dateTemp + "\n\n");
544         builder.append("Kinect Status: " + kinectStatus + "\n");
545         builder.append("Date (Kinect): " + dateKinect + "\n");
546         builder.append("--------------------------------------------\n");
547         return builder.toString();
548     }
549 }
```