

Personal Health Care System using Pulse Rate Sensor

CS 2310 Final Project Report

Ankita Mohapatra

Introduction:

Personal Healthcare has been and will continue being a popular topic in both academia and industrial community. With the rapid development and flexible deployment of wireless multisensory networks, it brings not only the opportunity, but also the challenge of how to benefit human the best, especially for personal medical and health environment.

In this project we get the pulse rate readings of an individual in real time, using a sensor device called Pulse Oximeter. The Pulse Rate readings are used to monitor the health of a person's heart. In case of any irregularities in the pulse rate of the individual, an alarm is triggered and the patient is alerted to visit the emergency as soon as possible. The Emergency manager then contacts the concerned Health Care staff to attend to the patient at the earliest. In the milestone 1, the scenarios are explained at depth using PrjtRemote to pass messages between several components of the system.

Saving the pulse rate of an individual is useful for diagnosing a patient who may have breathing issues while sleeping as well. Sleep apnea is a common ailment that can prevent patients from getting adequate rest during the course of an evening. With this system we could provide the patient with a simple monitor to track breath rate. A doctor could evaluate the output remotely and at a more convenient time. This would allow the patient to avoid an expensive overnight stay in a sleep laboratory, or to see if a visit to one is warranted.

System Design

In this project, I augmented the existed personal healthcare system, the SIS system, by adding six components, including both basic and super components. Additionally, a pulse sensor has also been implemented, which can monitor the pulses identifying normal and high rate of pulse in order to report an emergency alert for the patient to the emergency manager.

The overall structure is presented in Figure 1. As illustrated, every component connects via the SIS server. The component Emergency Manager is the super component that manages

all messages sent from basic components and communicates with users. Detailed description for each component is discussed as below.

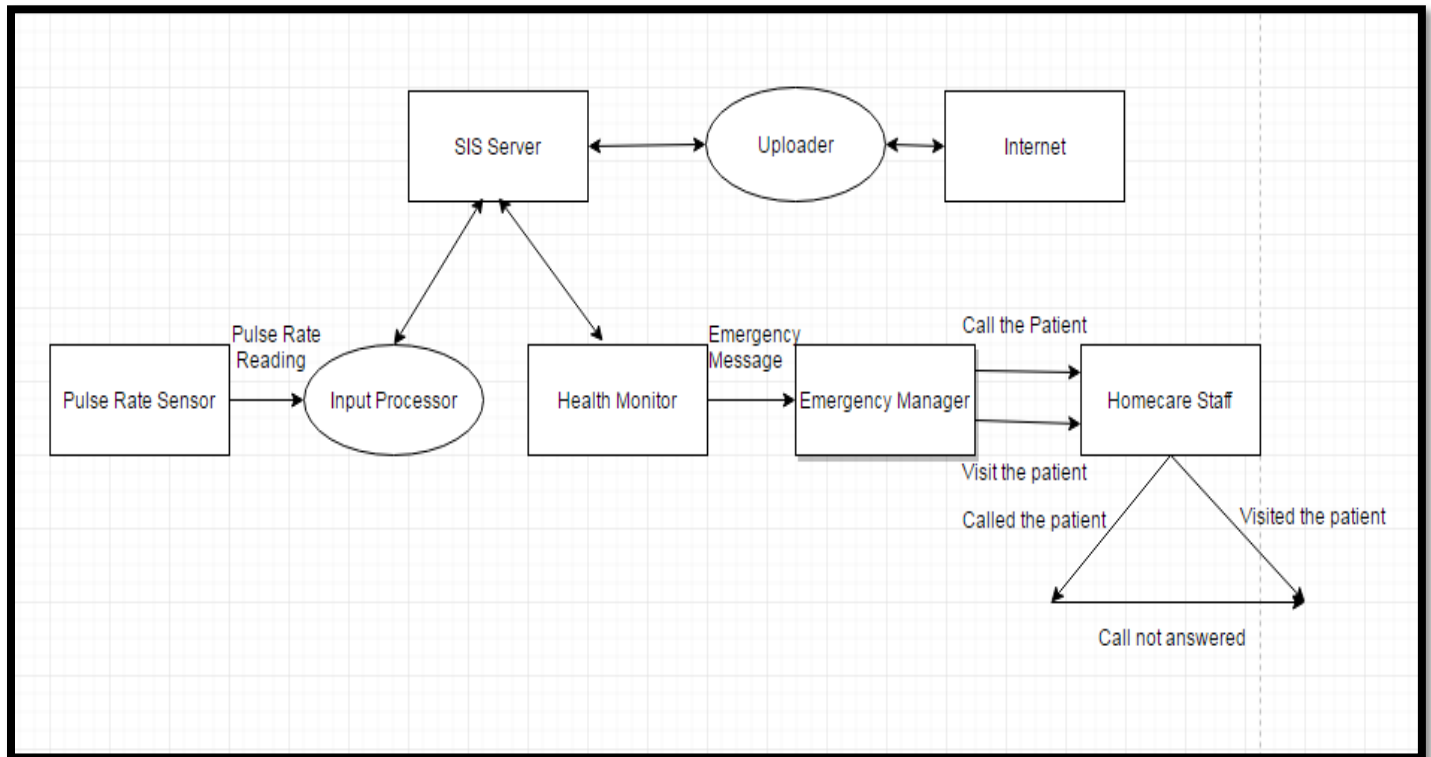


Figure 1. The overall structure of the personal healthcare system.

Components

1. Pulse Sensor:

This component is used to read the pulse rate from the pulse rate sensor which is connected to the patient at all instances. It displays the readings as received from the sensor. The sensor used in the project is **CMS 50D Plus pulse oximeter** which is a full featured fingertip oximeter that is perfect for patients who want to monitor their pulse and oxygen saturation periodically during the night. This type of oximeter can help you as a CPAP user determine the effectiveness of one's current CPAP pressure. With 24 hours of data storage, software, USB cable, carrying case, and 24 month warranty, the 50D Plus is a great way to take control of one's CPAP therapy and your health.

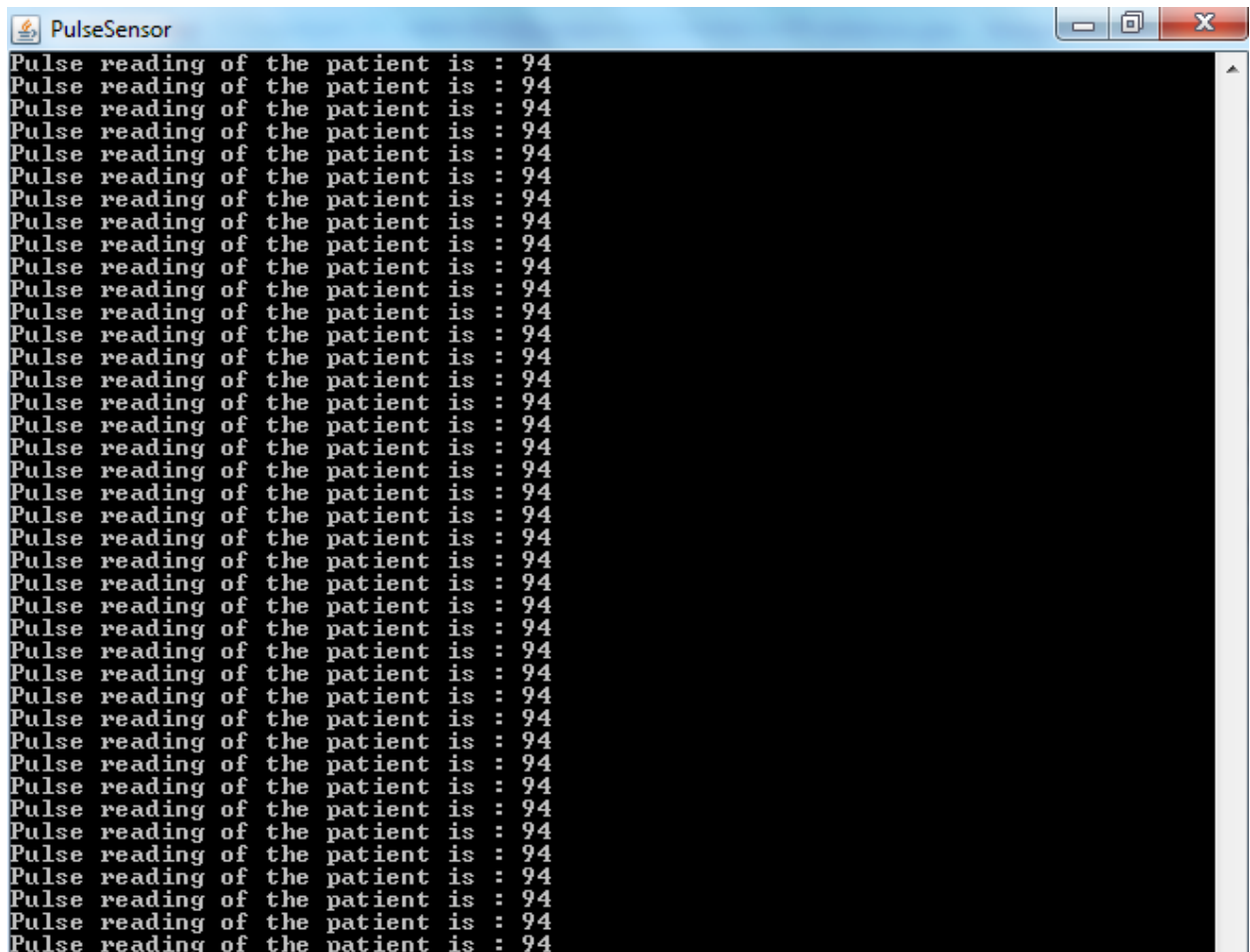


Figure 2. Readings displayed by the PulseSensor Component

2. Input Processor:

The Input Processor is a predesigned Basic component which takes in data stream generated by sensor device and outputs readings for the various vital signals such as Blood Pressure, SPO2, EKG, etc. In this project, this component is used to filter out the irrelevant and garbage values sent by the sensor device.

3. Health Monitor:

The major task of the Health Monitor component is to monitor the patients' pulse rate. As long as it detects an abnormal pulse rate value, it will send out an alert message, which is different from usual alert message, to the Emergency Manager asking for immediate medical attention.

The Health Monitor is designed as given below:

- Pulse Levels between 60 and 100: NORMAL
- Pulse Levels less than 60 and greater than 100: EMERGENCY

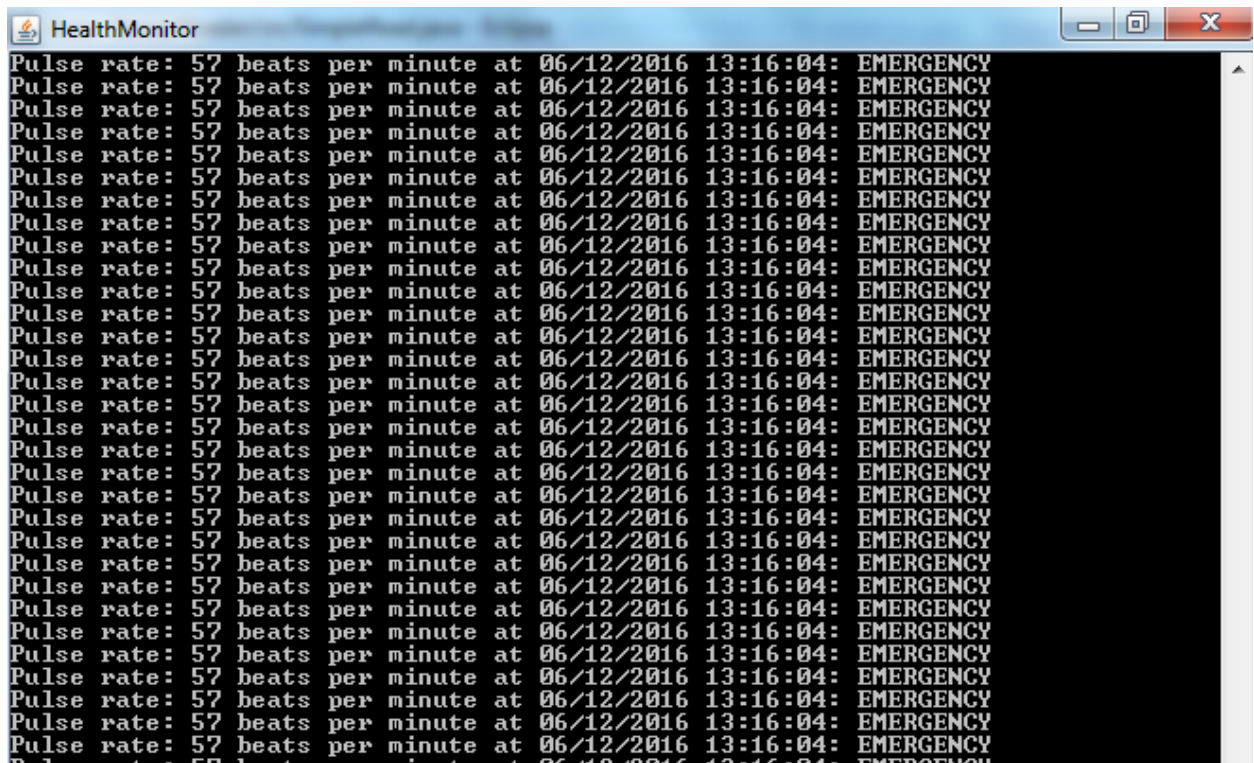


Figure 5. Readings displayed by the Health Monitor Component: EMERGENCY pulse levels

4. Emergency Manager:

This component acts as a super component that can generate different alert messages for the HomeCare Staff after receiving Emergency message from Pulse sensor.

- **Message came as the first time**

The emergency manager will send a message to homecare staff to tell them to call the patient.

- **Message came not the first time**

The emergency manager will send a message to homecare staff to tell them to visit the patient

After a message is processed, a timer will start a countdown, if there is no message coming after previous message for certain amount of time, which is a threshold, the emergency manager will clean the history so that if a message coming, the message will be treated as the first time attempt.

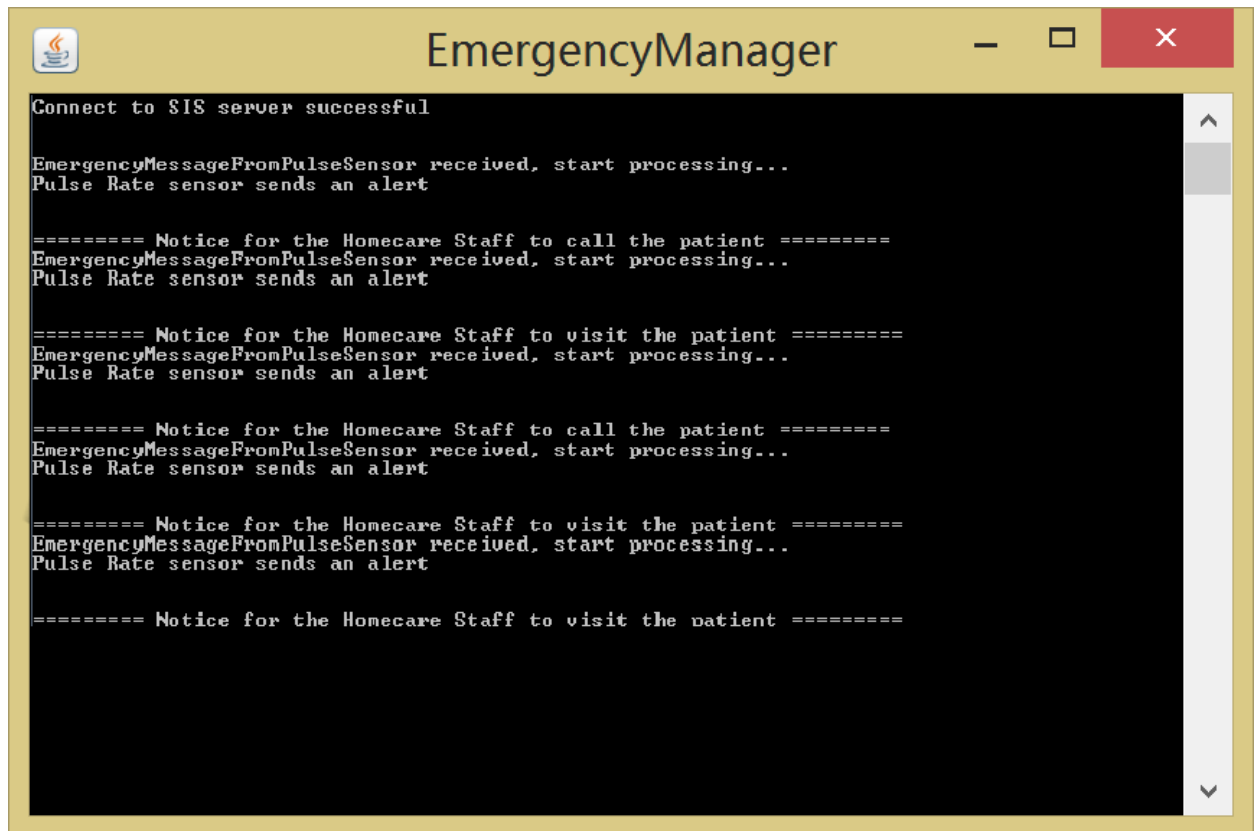


Figure 6. Messages displayed by the Emergency Manager Component

5. Homecare Staff:

This component will work to do the job that emergency manager asked. When it received the message, it will display it in a pop up window.

- **Message: Call The Patient**

It will display call patient information on screen to homecare staff and wait for the response from them to see if the patient picks up the phone. If the feedback is patient didn't pick up the phone, it will display visit patient information on screen to homecare staff.

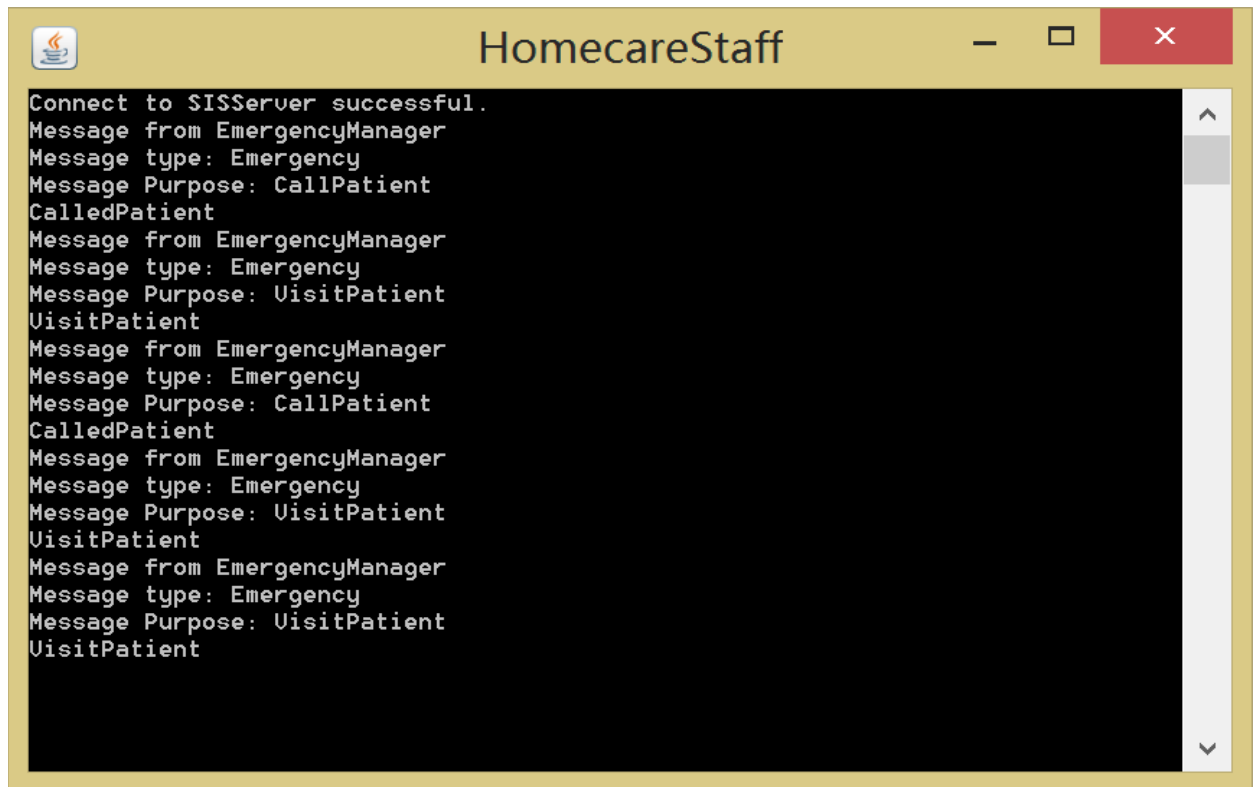


Figure 7. Messages displayed by the HomeCare Staff Component

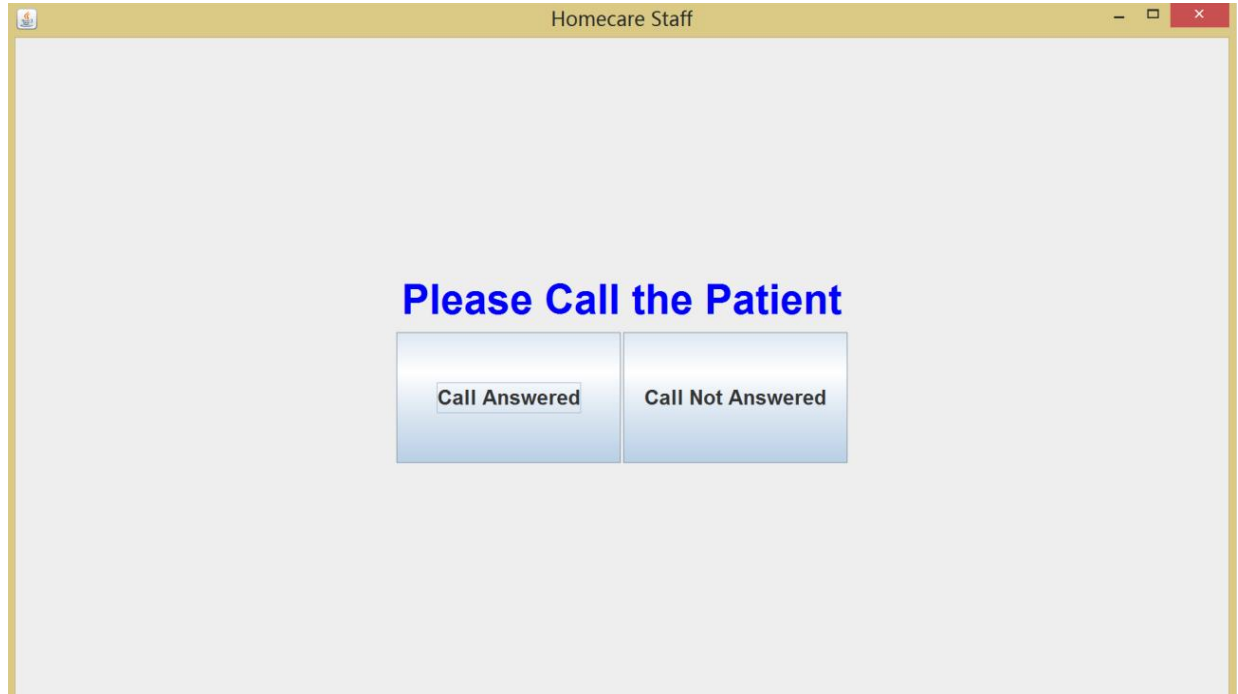
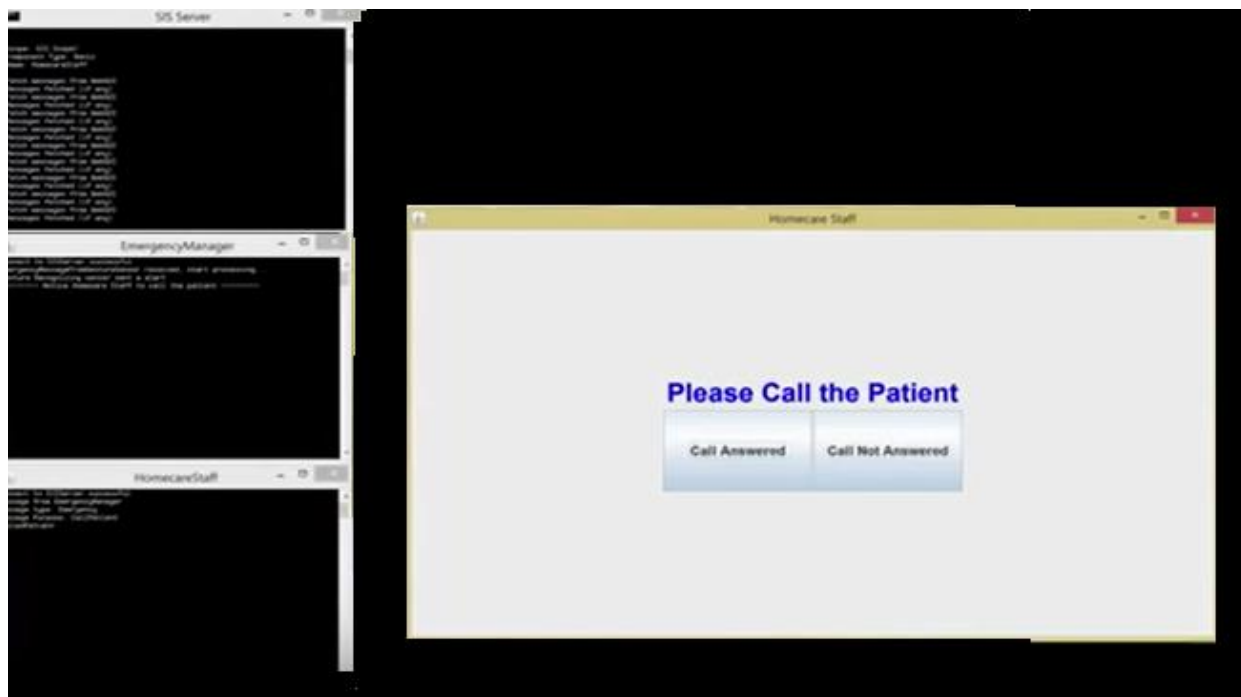
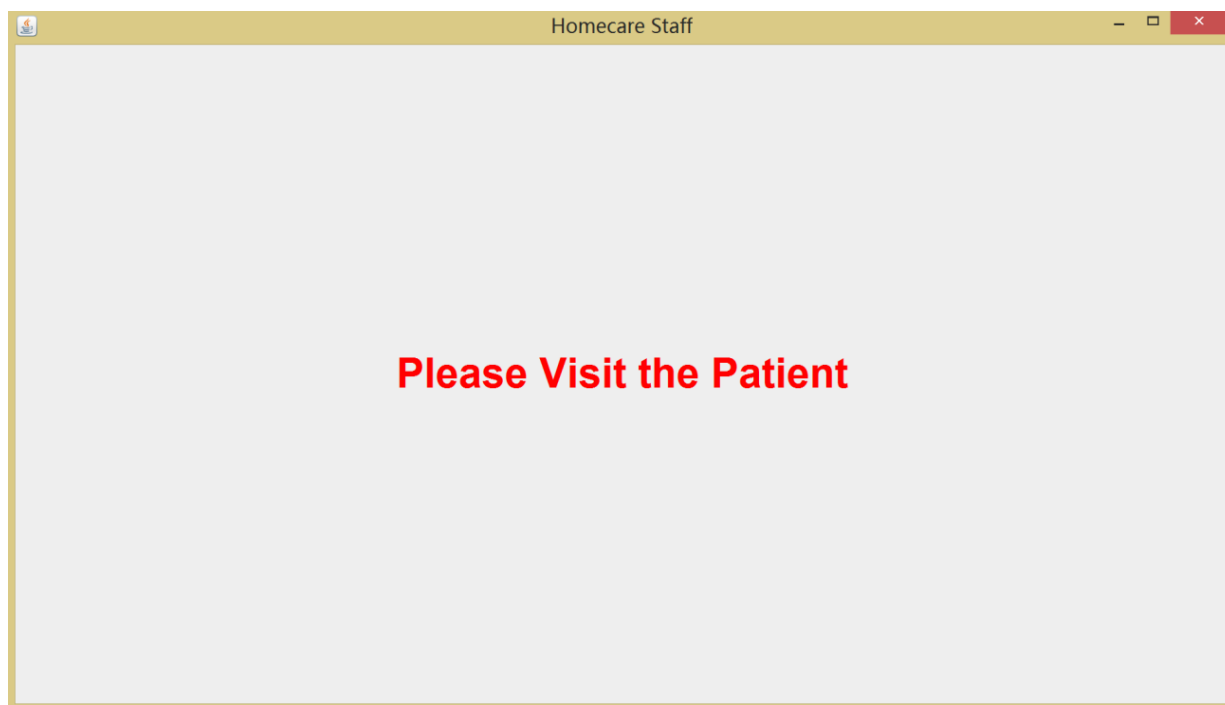


Figure 8. GUI for the HomeCare Staff to call the patient



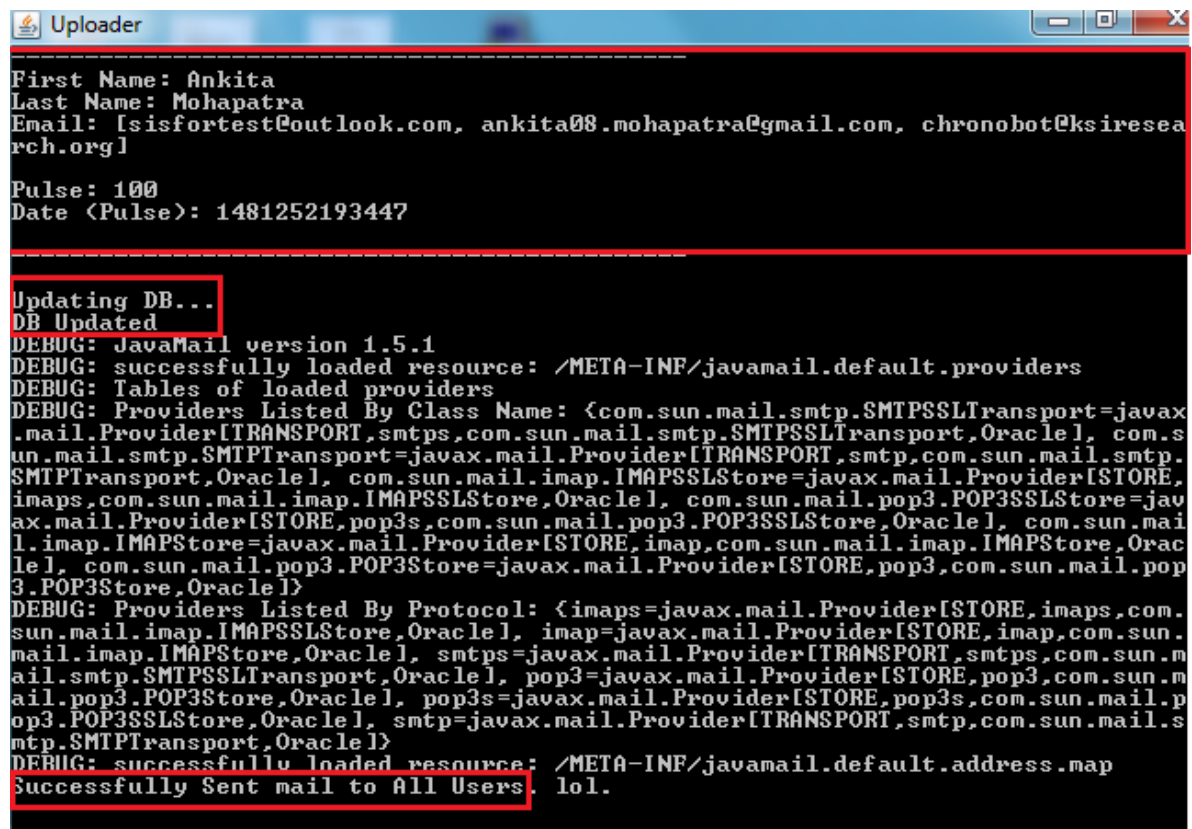
- **Message: Visit The Patient**
It will display visit patient information on screen to homecare staff.



The above is displayed until the patient is alerting the Emergency Manager and the HomeCare staff still hasn't visited the patient. This screen works as a constant reminder for the HomeCare staff to do the task as soon as possible.

6. Uploader:

Uploader is a predesigned Advertiser component which processes and propagates information to the outside world so authorized personnel can access said information. In this project, the Uploader component sends a mail with the patients' information to the SIS server mailbox and also uploads the patients' information in the main database.



```

Uploader
-----
First Name: Ankita
Last Name: Mohapatra
Email: [sisfortest@outlook.com, ankita08.mohapatra@gmail.com, chronobot@ksiresearch.org]

Pulse: 100
Date <Pulse>: 1481252193447
-----

Updating DB...
DB Updated
DEBUG: JavaMail version 1.5.1
DEBUG: successfully loaded resource: /META-INF/javamail.default.providers
DEBUG: Tables of loaded providers
DEBUG: Providers Listed By Class Name: {com.sun.mail.smtp.SMTPSSLTransport=javax
.mail.Provider[TRANSPORT,smtps,com.sun.mail.smtp.SMTPSSLTransport,Oracle], com.s
un.mail.smtp.SMTPTransport=javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.
SMTPTransport,Oracle], com.sun.mail.imap.IMAPSSLStore=javax.mail.Provider[STORE,
imaps,com.sun.mail.imap.IMAPSSLStore,Oracle], com.sun.mail.pop3.POP3SSLStore=jav
ax.mail.Provider[STORE,pop3s,com.sun.mail.pop3.POP3SSLStore,Oracle], com.sun.mai
l.imap.IMAPStore=javax.mail.Provider[STORE,imap,com.sun.mail.imap.IMAPStore,Orac
le], com.sun.mail.pop3.POP3Store=javax.mail.Provider[STORE,pop3,com.sun.mail.pop
3.POP3Store,Oracle]}
DEBUG: Providers Listed By Protocol: {imaps=javax.mail.Provider[STORE,imaps,com.
sun.mail.imap.IMAPSSLStore,Oracle], imap=javax.mail.Provider[STORE,imap,com.sun.
mail.imap.IMAPStore,Oracle], smtps=javax.mail.Provider[TRANSPORT,smtps,com.sun.m
ail.smtp.SMTPSSLTransport,Oracle], pop3=javax.mail.Provider[STORE,pop3,com.sun.m
ail.pop3.POP3Store,Oracle], pop3s=javax.mail.Provider[STORE,pop3s,com.sun.mail.p
op3.POP3SSLStore,Oracle], smtp=javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.s
mtp.SMTPTransport,Oracle]}
DEBUG: successfully loaded resource: /META-INF/javamail.default.address.map
Successfully Sent mail to All Users. lol.
  
```

Figure 11. Uploader successfully sends a mail and uploads the records to the database

Description of Scenarios using PetriNets:

I use Petri Net to describe different scenarios in this project. At the beginning, tokens are placed in each component's idle state, shown in Figure 11. As soon as the sensor/monitor detects the monitoring values, a token is placed and will trigger each component's task. Due to the space limit, in addition to the initial state, I will illustrate only one scenario using Petri Net. In this case, first, the Health Monitor detects the abnormal pulse rate value, and invoked the Emergency Manager, as shown in Figure 12. Two tokens are ready, which will trigger the Emergency Manager component to

send out a special alert message to Homecare Staff. After the event is triggered, the tokens' locations are shown in Figure 13. Figure 14 presented the Homecare Staff received the special alert message from Emergency Manager and triggered the event, sending out an emergency message "patient needs help". After this event, the system come back to the initial state.

Notice that, after triggering every event, the token is placed to the idle state of each component.

Thus, the component is always ready for processing new detections or messages.

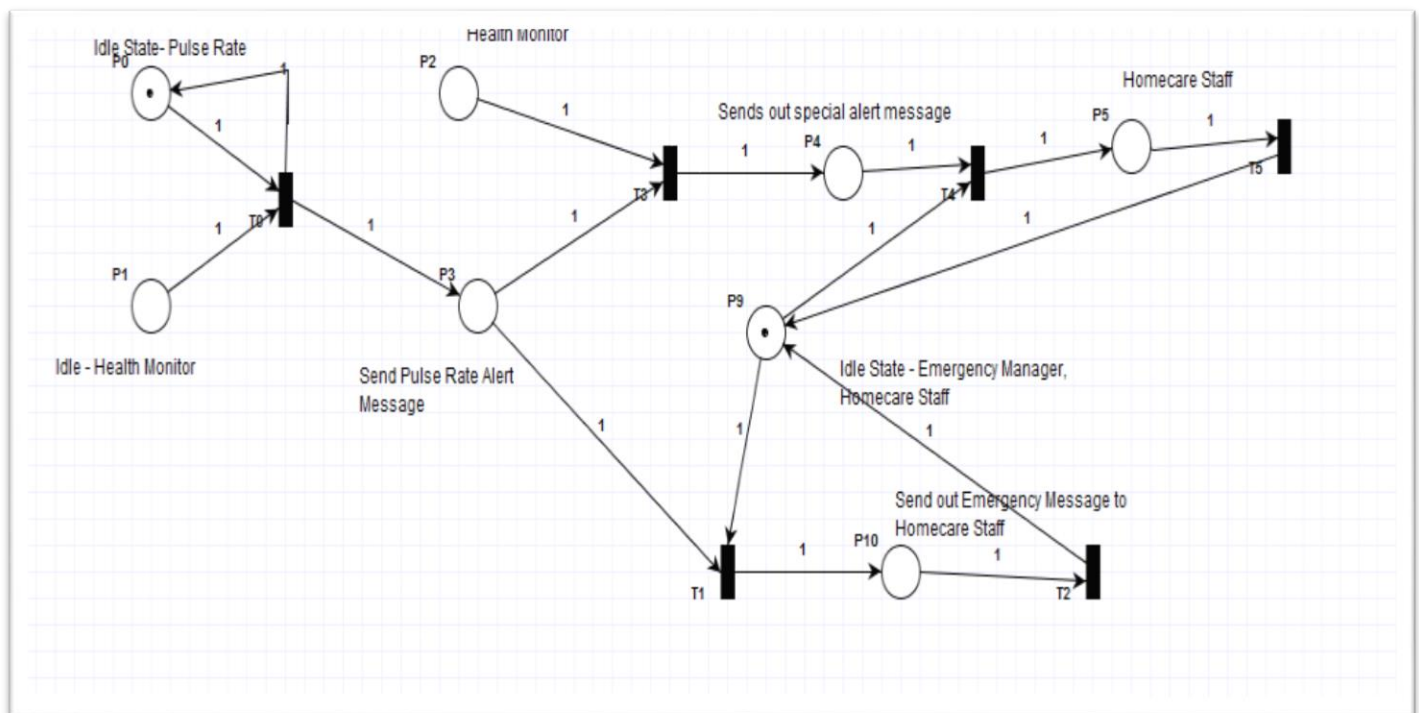


Figure 11. Initial state of the personal healthcare system

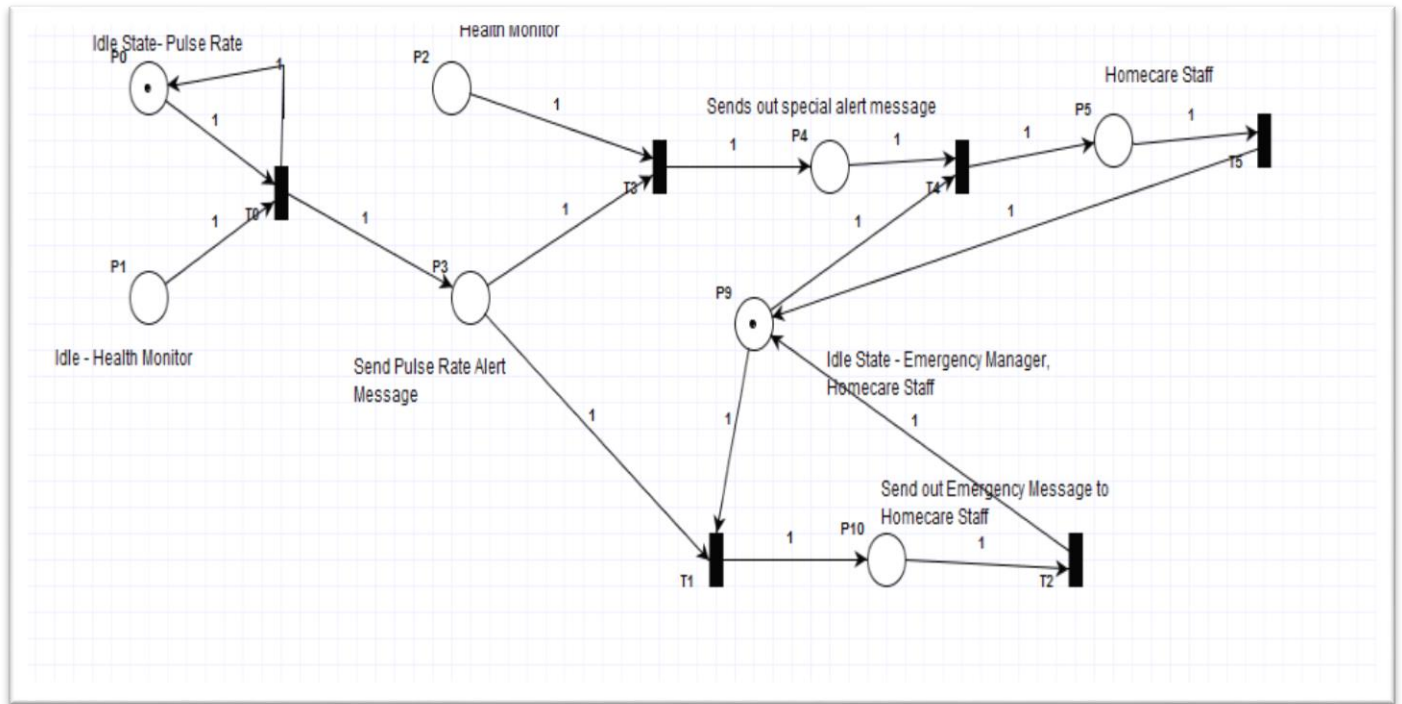


Figure 12. When Pulse Rate is abnormal, Health monitor alerts the Emergency Manager

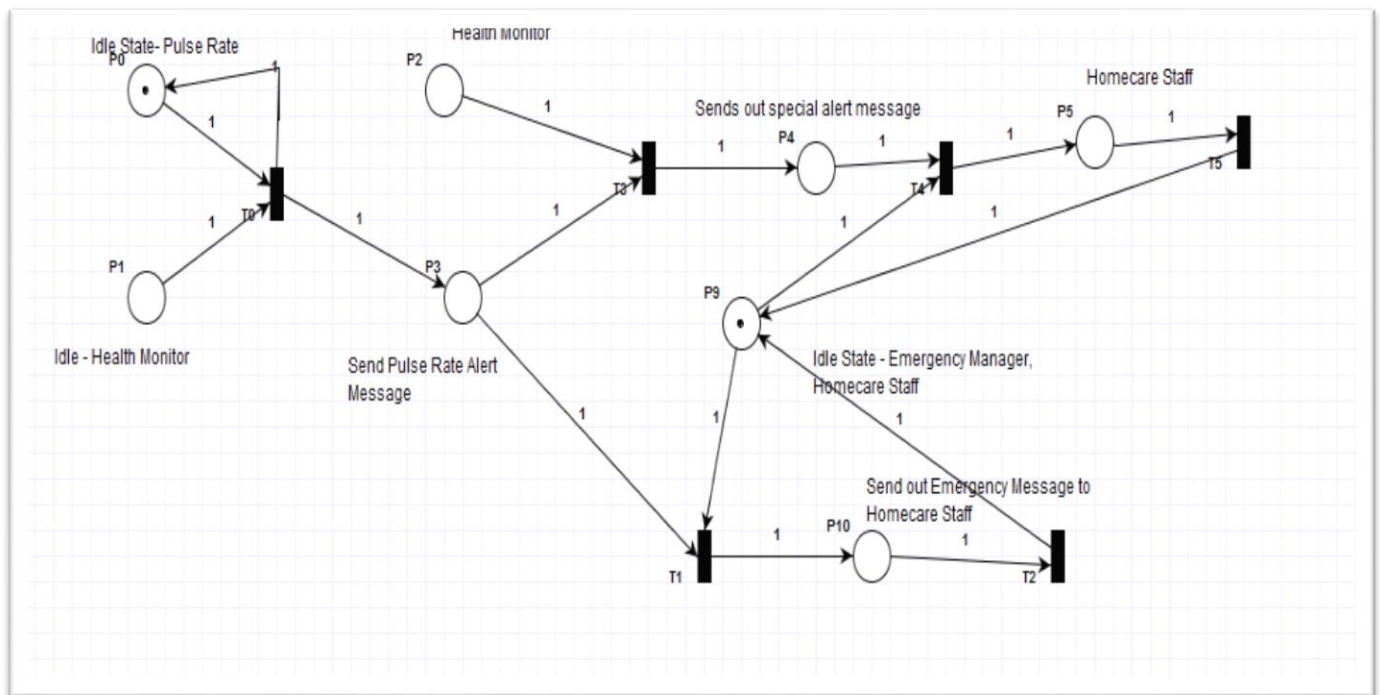


Figure 13. Emergency Manager is sending out a special alert message to Homecare Staff.

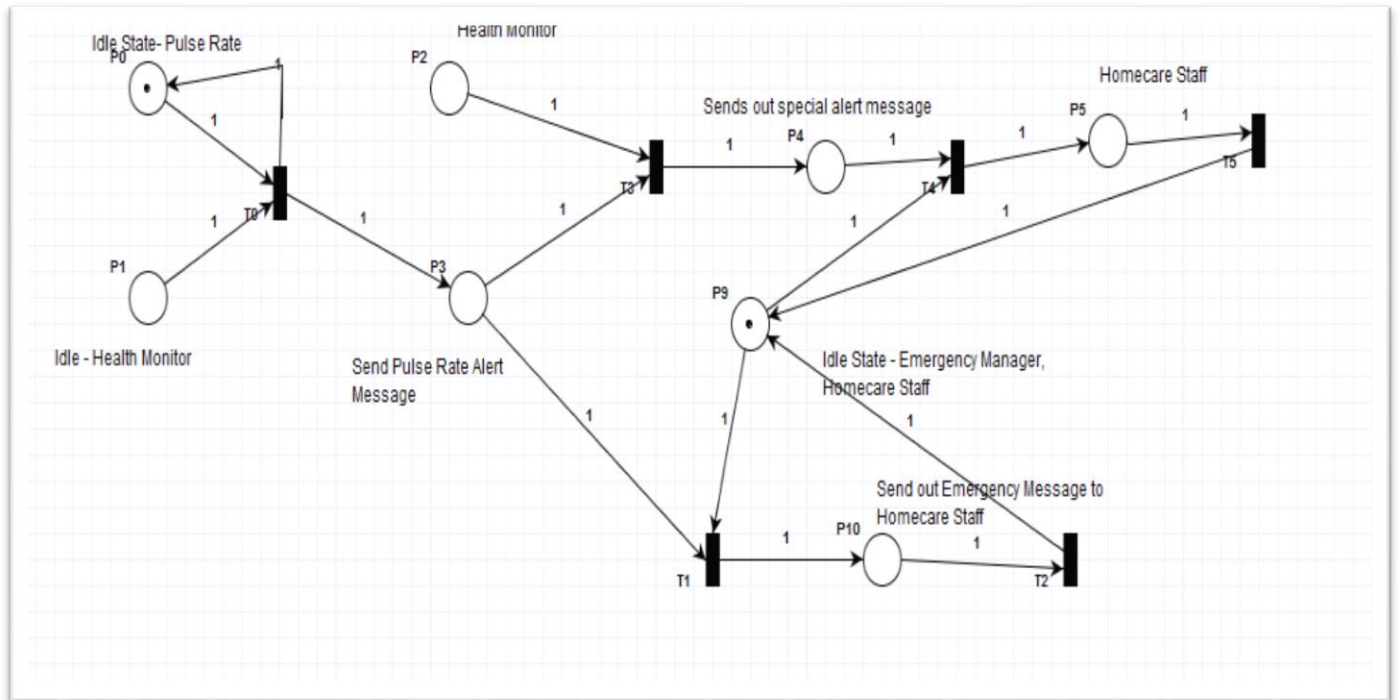


Figure 14. Homecare Staff is attending to the emergency alert message

Possible Future Works:

Currently, the Personal care System is doing a simple job of checking the recorded values and making a decision of whether sending out an emergency message. Ideally, it should perform as an intelligent system that can do the initial diagnose by integrating machine learning classification models. I think this could be an interesting topic to combine different research field and benefits the whole community.

Extra Deeds:

The following bullets are gems I have attempted for:

- Using Petri Net to describe scenarios.
- Graphical User Interface for the Home Care System.

Appendix:

1. Pulse Sensor & Input Processor:

```
import java.io.*;
```

```

import java.util.*;

import javax.comm.*;

import java.net.*;

public class SensorRead implements Runnable, SerialPortEventListener {
    static CommPortIdentifier portId;
    static Enumeration portList;
    int handle = 0;
    int skipper = 0;
    InputStream inputStream;
    SerialPort serialPort;
    Thread readThread;
    //StringBuffer sb = new StringBuffer();
    String str = "";
    Socket s;

    public static void main(String[] args) {
        portList = CommPortIdentifier.getPortIdentifiers();
        System.out.println("PortList : " + portList);
        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals("COM4")) {
                    SensorRead reader = new SensorRead();

                }
            }
        }
    }

    public SensorRead() {
        try {
            serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
        } catch (PortInUseException e) {
            System.out.println(e);
        }
        try {
            inputStream = serialPort.getInputStream();
        } catch (IOException e) {
            System.out.println(e);
        }
        try {
            serialPort.addEventListener(this);
        } catch (TooManyListenersException e) {
            System.out.println(e);
        }
        serialPort.notifyOnDataAvailable(true);
        try {
            serialPort.setSerialPortParams(19200, SerialPort.DATABITS_8,
SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        } catch (UnsupportedCommOperationException e) {
            System.out.println(e);
        }
        readThread = new Thread(this);
        readThread.start();
    }

    public void run() {
        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

```

```

public void serialEvent(SerialPortEvent event) {

    switch (event.getEventType()) {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:

            while(true)
            {
                handle++;

                byte[] readBuffer = new byte[20];
                int ascii;
                try {
                    while (inputStream.available() > 0) {

                        int numBytes = inputStream.read(readBuffer);
                    }
                    int index = 0;
                    byte regular = readBuffer[index];

                    int pulse = (int)regular;

                    // if (pulse > 10)
                        System.out.println("Pulse Rate :" + pulse);

                    try {
                        while (pulse > 0){

                            s = new Socket("127.0.0.1", 54000);
                            DataOutputStream output = new
DataOutputStream(s.getOutputStream());
                            output.writeUTF(Integer.toString(pulse));
                            //System.out.println("Pulse Rate new :" + Integer.toString(pulse));
                        }

                        catch (Exception ex1) {
                            ex1.printStackTrace();
                        }

                    catch (IOException e) {
                        System.out.println(e);
                    }

                    break;}

            }
    }
}

```

2. HomeCare Staff GUI:

Changes in the HomeCare staff component code:

...

```

switch (purpose)
{
    case "CallPatient":
        System.out.println("CalledPatient");
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                showCall();
            }
        });
        break;
    case "VisitPatient":
        System.out.println("VisitPatient");
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                showVisit();
            }
        });
        break;
}

break;

}

}

private static void showVisit() {
    //Create and set up the window.
    JFrame frame = new JFrame("Homecare Staff");
    //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // make the frame half the height and width
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int height = screenSize.height;
    int width = screenSize.width;
    frame.setSize(width/2, height/2);

    // here's the part where i center the jframe on screen
    frame.setLocationRelativeTo(null);
    JLabel label = new JLabel("Please Visit the Patient", SwingConstants.CENTER);
    label.setFont(label.getFont().deriveFont(64.0f));
    label.setForeground(Color.GREEN);
    // label.setPreferredSize(new Dimension(350, 200));
    frame.getContentPane().add(label, BorderLayout.CENTER);
    //frame.setSize(1000, 600);
    //Display the window.
    //frame.pack();
    frame.setVisible(true);
}

public static void showCall() {
    //Create and set up the window.
    JFrame frame = new JFrame("Homecare Staff");
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int height = screenSize.height;
    int width = screenSize.width;
    frame.setSize(width/2, height/2);

```

```

// here's the part where i center the JFrame on screen
frame.setLocationRelativeTo(null);
//frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new BorderLayout());
frame.add(new CallPane(frame), BorderLayout.CENTER);
//frame.setSize(1000, 600);
//Display the window.
//frame.pack();
frame.setVisible(true);
}

public static class CallPane extends JPanel {

    public CallPane(JFrame myprogram) {
        setLayout(new GridBagLayout());
        GridBagConstraints constraint = new GridBagConstraints();
        constraint.gridx = 0;
        constraint.gridy = 0;
        constraint.insets = new Insets(2, 2, 2, 2);
        JLabel label = new JLabel("Please Call the Patient");
        label.setFont(label.getFont().deriveFont(64.0f));
        label.setForeground(Color.RED);
        constraint.gridx = 0;
        constraint.gridy++;
        constraint.fill = GridBagConstraints.NONE;
        // constraint.gridwidth = 2;
        JButton answeredButton = new JButton("Call Answered");
        answeredButton.setFont(answeredButton.getFont().deriveFont(32.0f));
        answeredButton.setPreferredSize(new Dimension(350, 200));
        answeredButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e)
            {
                //Execute when notAnsweredButton is pressed
                myprogram.dispose();
            }
        });
        JButton notAnsweredButton = new JButton("Call Not Answered");
        notAnsweredButton.setFont(notAnsweredButton.getFont().deriveFont(32.0f));
        notAnsweredButton.setPreferredSize(new Dimension(350, 200));
        //Add action listener to notAnsweredButton
        notAnsweredButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e)
            {
                //Execute when notAnsweredButton is pressed
                showVisit();
                myprogram.dispose();
            }
        });
        add(answeredButton, constraint);
        constraint.gridx++;
        add(notAnsweredButton, constraint);
        constraint.gridx = 0;
        constraint.gridy = 0;
    }
}

```

```
constraint.gridwidth = 2;  
add(label, constraint);
```

```
}
```

```
}
```