Meshal Albaiz
Tristan Cunderla
ECE558 - Final Project
12/2/2019

IoT Thermostat - Theory of Operation

Project Description

The purpose of this project is to create an IoT thermostat system using Android phones and a Raspberry Pi thermostat controller. This system allows the user to set the temperature of the thermostat, and read the current temperature in real-time. This system allows for multiple users to have control over the thermostat and keeps a history of what each user did. There is also a real-time graph tab where the setpoint and current temperature are shown. The user can turn off or turn on the thermostat, and the fan speed can be set between manual (low, medium, and high) and automatic. In automatic mode, the fan speed is based on the difference between the current temperature and the setpoint, the fan speeds up by 10% for every 1°C. If the current temperature is higher than the setpoint temperature then the A/C is cooling and the LED is blue, if the current temperature is lower than the setpoint temperature then the A/C is heating and the LED is red. If the temperatures are within 1°C of each other then the A/C is ventilating and the LED is green. The Raspberry Pi embedded system utilizes a PIC I2C controller, a temperature sensor, RGB LED, and a motor simulating the fan A/C fan.

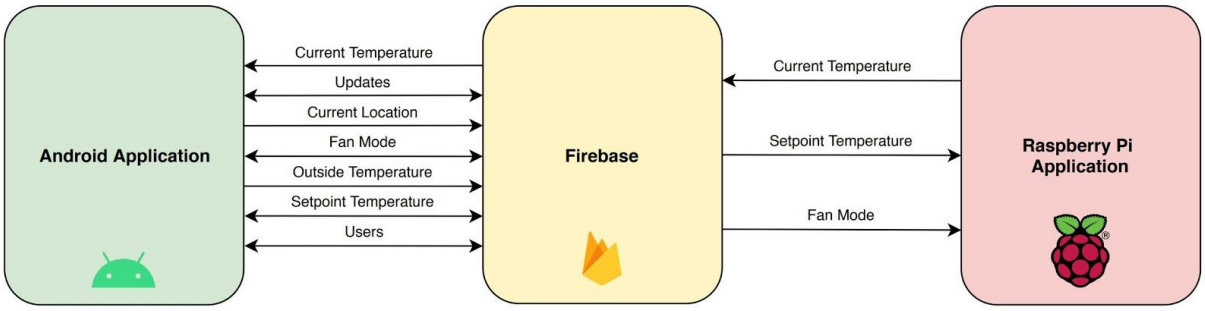


Figure 1: Overall Thermostat System Communication

Deliverables

Minimum Deliverables:

- Android Application
  - Ability to adjust temperature
  - Ability to set fan mode (manual or automatic)
  - Display current temperature from Raspberry Pi temperature sensor
  - Multiple phone integration (when one person changes the settings other phones using the app will be notified)
  - Implementation of sound
  - Ability to read and write data to Firebase
- Raspberry Pi
  - Utilize Android Things
  - Uses LED to indicate whether the system is heating, cooling or stable
  - Motor direction indicates whether system is heating or cooling
  - Ability to read and write data to Firebase

Stretch Goals:

- Add speech recognition to the android application so the user can set the temperature with their voice
- Add a real time graph to android application
- What-if scenarios to android application (example: if I have my system on for this long then it will cost X amount of money)
- Add text to speech to Pi app to verbally indicate that the temperature has been changed
- The ability to schedule the thermostat to operate during a certain time block
- Android app can access current weather conditions
- Add a screen to Raspberry Pi
- Notifications to all phones when thermostat settings change

*Goals highlighted in blue have been implemented

System Hardware

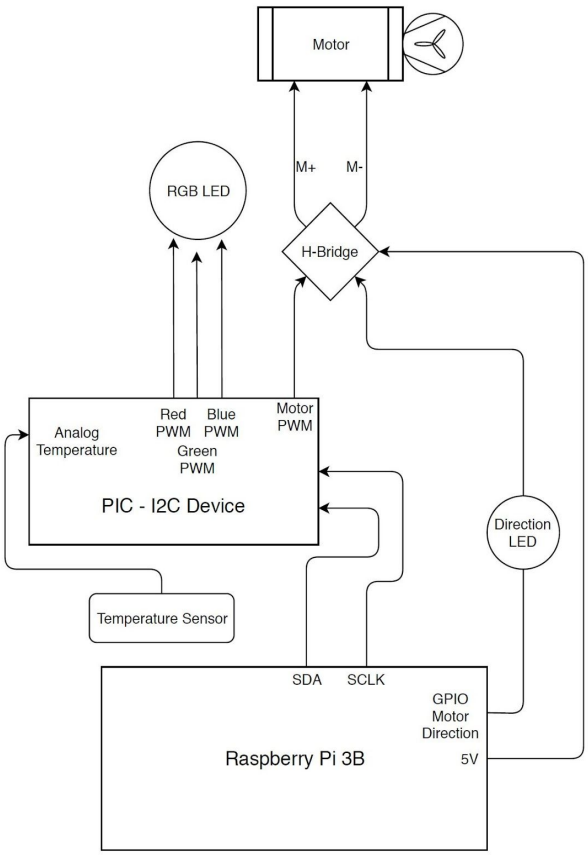
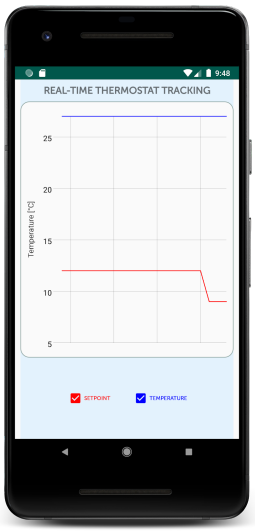


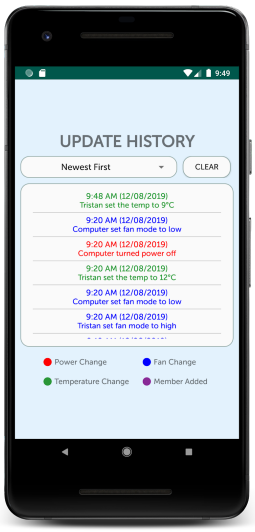
Figure 2: Thermostat Hardware System

Android App Layout

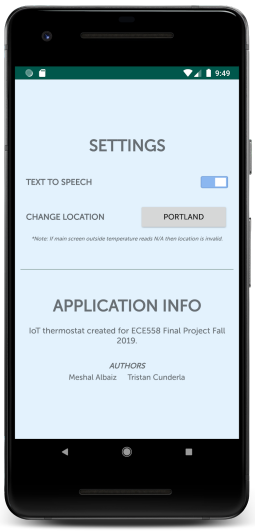
Main Activity



Graph Activity



Updates Activity



Settings Activity

Main

The main activity serves as the main dashboard and control center for the entire system. On this screen the user may use an imported custom circular seekbar to adjust the thermostat setpoint. A button is provided in the center of the circular seekbar that is used to set the setpoint. An additional custom seekbar is used to set the fan mode to one of five different options: off, automatic, low, medium or high. Three circular progress bars are used to display the current temperature reading from the Raspberry Pi, the current outside temperature as well as the setpoint for the system. An ImageView is used to indicate the current fan mode and temperature zone. To indicate the different fan modes an animation is used on the ImageView to rotate the image at different speeds. To indicate the temperature zone (heating, cooling or stable) the color of the ImageView is altered. There are four image buttons. Three of the image buttons are used to open the graph, updates and settings activities and the fourth is connected to the speech to text functionality. Lastly, the title and all labels are displayed using TextView widgets.

Updates

The main widget for the updates activity is a scrollview. This scroll view is populated with numerous linear layouts that depict when an update took place and what the update was as well as who made the update. This scroll view widget is dynamically populated within the java code. A spinner can be used to set the order in which the user would like to view system updates (newest first or oldest first). A button is also provided that allows the user to clear the thermostat update history. TextView widgets display the title as well as a legend for the different types of system updates that can occur and their corresponding color code.

Graph

The graph activity layout consists of a title TextView, a graph GraphView, and two checkboxes that enable or disable the setpoint and current temperature from being shown on the graph.

Settings

The settings activity layout is quite simple. Two buttons are provided, one to turn on and off text-to-speech and the other to change the current location of the thermostat. Serval TextView widgets are used to display the title of the activity as well as labels and application information.

Android App Java

Main

The main activity serves as the central hub of the entire Android application. Within this application the user my set the setpoint of the thermostat as well as turn the thermostat on and off and adjust the fan mode. The main activity also allows the user to monitor the thermostat statistics and access other thermostat pages and functionality.

To utilize the BubbleSeekBar package a setOnProgressChangedListener() is created. Within this listener both the getProgressOnActionUp() and getProgressOnFinally() methods are implemented. Both methods implement very similar functionality with very few dissimilarities. Both methods start by checking the progress of the seekbar and implementing a switch statement in order to correlate a numerical seekbar value to a string value indicating the fan setting. The methods then check to see if the user actually changed the fan mode from the previous fan mode. Next, it checks to see if the current user set the fan mode or if another user set it. This check is implemented because the seekbar is updated across all devices when a change is made to the fan mode. This helps determine if an update should be sent to the Firebase or not. If the current user is the one who made the change then the fan mode is written to the Firebase and an update message is created and written to the Firebase and the fan mode seekbar is adjusted accordingly. If the current user did not make the update then nothing is written to the Firebase but the seekbar is still updated. The method then checks the fan mode that the system is set to. If text to speech is on then a verbal indication that the fan mode was changed will be played including the specific mode the system was changed to. The only difference between the two methods is how each function uses the boolean value onActionUsed. This variable indicates to the getProgressOnFinally() method that the getOnProgressChangedListener() method has already executed and there is no need to execute the getProgressOnFinally() method. This functionality had to be implemented because during speech to text commands both methods were executing causing values to be written to the Firebase twice and text to speech audibles playing twice instead of once upon a fan mode change.

Next, the setOnSeekBarChangeListener is implemented for the circular custom seekbar CircleSeekBar. This seekbar is used to adjust the thermostat setpoint. The listener's task is to update the text displayed on a circular button centered within the seekbar and to keep track of the current value of the seekbar by updating an activity instance variable. The color of the seekbar is changed depending on the temperature that is selected. To do this within the seekbar listener the setReachedColor() method, a method included in the custom package that sets the color of seekbar given an integer color value, is used in pair with the set_color() method. The set_color() method returns a color hex value based on a temperature value. If the temperature is above 35°C then the color will be red. If the temperature is between 26°C and 35°C then the color will be orange. If the color is between 20°C and 25°C then the color will be green. Lastly, if the temperature is colder than 20°C then the color will be blue. To set the color

of the seekbar using the setReachedColor() method the set_color() method with the current temperature passed as an input is then passed into the setReachColor() method and thus setting the color of the seekbar. This can be seen in the screenshot below.

```
// set the color of the seekbar depending on the temperature value
mTempSeekBar.setReachedColor(Color.parseColor(set_color(curValue)));
```

Figure 3: Example of How to Set Thermostat Setpoint Seekbar Color

The thermostat setpoint button onClickListener() is then created. When the button is clicked the listener first checks that the new setpoint is not the same as the current setpoint by comparing the setpoint seekbar value to an instance variable that is updated whenever the Firebase setpoint value is changed. If the setpoint is not different, a toast indicating that the setpoint has not changed is made. The listener then checks the fan mode seekbar to confirm that the system is not off. If the system is off then a toast is made indicating that the system must be on to change the setpoint. If the system is on and the setpoint is different from the current system setpoint then the setpoint seekbar progress is written to the Firebase child “set_temp”, an update message is created and then written to the Firebase and if text to speech is on then a verbal indication of the change will be made.

When the current location of thermostat is changed the value listener for “current_location” is triggered. Within this listener the new location is retrieved and an AsyncTask weatherTask is executed, which updates the outside temperature displayed within the main activity dashboard. The AsyncTask weatherTask will be explained in more detail later in this report.

The value listener for “fan_mode” waits for a change to be made to the thermostat fan. When a fan mode change occurs the listener first checks to see if the fan mode is different than the current fan mode. If the fan mode is different, then the listener calls the animate_fan() method which sets the animation for the ImageView widget displaying a fan image. The new fan mode is saved to an instance variable and the method set_fan_color() is called to match the ImageView of fan’s color to the current temperature operating zone of the system. Both the animate_fan() and set_fan_color() methods will be highlighted later in this section of the report.

Both the “current_temp” and “set_temp” listeners operate in very similar fashion. The listeners begin by reading their respective children values from the Firebase. After obtaining a value from the Firebase the listener will update the TextView and ProgressBar widget displaying their respective temperature values. Both the TextView and ProgressBar widget are set to the color symbolizing which temperature zone that they fall in. The color for the TextView widget is set using the setTextColor() method and the color for the ProgressBar is set using getProgressDrawable().setColor(getResources.getColor(*fan color*), PorterDuff.Mode.SRC_IN). Lastly, the current temperature zone is calculated and the ImageView fan color is updated using the set_fan_color() method.

The last Firebase listener is for the “users” child. This listener keeps track of all the unique users that have used the app. When the application starts up or when the “user” child is edited then this listener will check to see if the current mobile device is “registered” within the Firebase. To do this the listener iterates through the “user” children looking for the devices serial number. To access the device’s serial number BUILD.Serial is used. If the device is found in the Firebase then no action is needed. If the device is not in the Firebase then an AlertDialog window is created. The AlertDialog window prompts the user to enter their preferred username. Upon hitting the confirmation button the username that was provided is written to Firebase with the device’s serial number as the key and then an update message is created and written to the database.

The listener for the graph, updates and settings buttons simply launches their respective activities by sending an intent with no expectation of receiving a response back. The listener for voice command button begins executing the speech to text functionality of the application.

The speech to text functionality was implemented with the help of a tutorial found online at Stack Tips. The link to this tutorial will be provided in the resources section of the report. The tutorial provides instructions on how to initialize a text to speech object, which can handle text to speech or speech to text, as well as how to send an intent from the current main activity to an activity that processes the user’s speech and translates it to a string which is returned to the main activity using the onActivityResult() method. When the speech to text activity is complete it passes an intent back to the main activity containing a string containing the words that the user spoke. If the string returned from the speech to text activity is not null then the method begins to look for keywords within the string. If the string contains words such as “power”, “turn” or “fan” then the application infers that the user wants to change the fan mode or power state of the system. If the string contains words such as “temp”, “temperature”, “Celsius”, “set” or “point” then the application infers that the user wishes to change the thermostat set point. Once the method determines if the user wishes to change the fan mode, power or setpoint then the voice string is then examined again for more specific keywords. For example, if the application determines the user wishes to change the fan mode it will look for keywords such as “automatic”, “auto”, “high”, “medium”, “low” and “off”. If one of these keywords in not detected then the application will indicate that an invalid fan mode was provided. If one of these keywords is detected then the fan mode seekbar is set accordingly using the setProgress() method, the “fan_mode” child value is updated in the Firebase and an update message is created summarizing the change. A similar methodology is used to set the setpoint of the system. If the string does not contain any of these keywords then a toast is made indicating to the user than invalid command was provided.

The update_history() method is responsible for creating update messages to be written to the Firebase. This method takes in one string input value which indicates what kind of update has taken place. The first task of this method is to construct an update message to write to the Firebase. To do this the string that was passed as input is used within a switch statement. If the string is “set_temp” that means that the setpoint was changed and a custom string including

who changed the setpoint and to what temperature will be generated. When “set_fan” is passed a custom message with the user and and new fan mode is made indicating a change with the system fan mode. The input option “power_off” generates a string containing who turned off the thermostat. Currently, the number of update messages within the Firebase is limited to 20. This number can be easily increased or decreased by changing the number within one for loop. To keep the most recent 20 updates within the Firebase the updates are read and sorted every time someone generates an update message. To do this, the method utilizes a listener for a single event for the “updates” child of the Firebase. Within this listener the key of each child within “updates” is retrieved using a for loop. Within the for loop the key values, which are time stamps in the form of strings, are translated to calendar objects and then added to a date list array. By translating the strings into calendar objects the time stamps can be sorted from oldest to newest using the sort() on the list array. This translation and sorting process can be seen in Figure 4.

```
// iterate through all the children of "updates"
for (DataSnapshot snapshot : dataSnapshot.getChildren()){
    // create a new calendar instance
    Calendar first_time = Calendar.getInstance();
    // obtain time of message in Firebase
    String update_time = snapshot.getKey();
    // attempt to translate string tme value to calendar object
    try {
        first_time.setTime(sdf.parse(update_time));
    } catch (ParseException e) {
        e.printStackTrace();
    }
    // add the calendar object to the list array
    time_array.add(first_time.getTime());
}
// sort list array by time from oldest to newest
Collections.sort(time_array);
```

Figure 4: Creating Date List Array and then Sorting List Array

After sorting the times within the data list array entitled time_array the listener checks the length of the array. If the array length is less than 20 then the update text is written to the Firebase with the current time as it’s key. If the array length is 20 then the key time value at index 0 is used to retrieve and remove the oldest update message saved within the Firebase. The new update is then written to the Firebase with the current time as it’s key.

The previously mentioned weatherTask AsyncTask uses the OpenWeather API to retrieve the current weather conditions for the location of the thermostat. The weatherTask task overrides the doInBackground() and onPostExecute() methods. The doInBackground() method obtains the current weather from OpenWeather by using the HttpRequest.executeGet() method. Upon completion the doInBackground() method returns a string containing the weather information in JSON format that is then provided to the onPostExecute() method as an input. When the onPostExecute() method is called it begins by creating a JSON object of the string returned from the doInBackground() method. The method then obtains the current temperature by first obtaining the JSON object “main” and then by using the getInt() method on the “temp” key in the “main” JSON object. The temperature value is then displayed using a TextView widget and circular progress bar just like the setpoint and current temperature values. The color of the

TextView widget and progress bar are also set similarly to other temperature values. If the task is unable to obtain a temperature value for the specified location then the TextView widget will display “N/A” and the progress bar will be set to zero.

The method animate_fan() is called within the value listener for the “fan_mode” child of the Firebase. The main functionality of this method is to set the rotation speed of an ImageView of a fan that indicates the current fan state of the system. Different fan modes spin the ImageView widget at different speeds providing a visual indication of the systems state. Upon execution the method first determines the animation duration by using a case statement based on the input string value. If a different user changed the fan speed then the seekbar indicating the fan mode will also be updated within this switch statement using the setProgress() method. As the animation duration decreases, meaning the object will spin faster, the seekbar value increases. A pre-existing rotation animation file is then loaded by calling the AnimationUtils.loadAnimatoin() method to create a new animation object for the fan image. The animation duration is then set using the setDuration() method and the animation is started indefinitely (or until it is stopped) using startAnimation().

The onStart() method has been overridden to ensure that the text to speech setting set in the settings activity is recovered each time the main activity is accessed. This method also locks the activity in portrait mode.

Lastly, the set_fan_color() method is called within several different Firebase listeners. This method is used to determine the current temperature zone that the thermostat system is currently operating in. The stable or venting temperature zone is defined when the setpoint temperature is equal to the current temperature reading. If the system is operating in this stable zone the color of the ImageView of the fan is set to green. When the setpoint is above the stable temperature zone then system is heating and the color of the fan is set to red. If the setpoint is below the stable temperature zone then system is cooling and the color of the is set to blue. If the system is off then the color of the ImageView of the fan is set to dark gray.

Updates

The majority of the information within the updates activity is held within two string arrays. One string array holds the time stamps for when an update occurred and the other holds the update message including the user and what change was made. At the moment the last 20 updates to the thermostat system are saved to the Firebase but this number can be adjusted by changing just one number. Both these arrays are populated and maintained within a ValueEventListener() for the “updates” child within the Firebase. Each child has a timestamp as the key and the update message as the value. On startup or when a change occurs within the child, both arrays are cleared to ensure that only the most recent and relevant updates are shown. The listener then iterates through each child of “updates” in the Firebase. First, the key, or timestamp, is retrieved for the child and converted to a calendar object and then stored into the time_array array. The timestamp is converted to a calendar object so that the times can be

sorted easily and efficiently within time_array. After retrieving all the timestamp values from the Firebase then the array is sorted using the built-in sort() method, which organizes the dates from oldest to newest. The listener then begins to match the timestamp key value to it's update message value. To do this the listener iterates through the time_array one index at a time. To associate the correct message with the correct timestamp, the timestep is passed as a child to the reference of the Firebase and the getValue() method is used to return the update message string. The string is then added to the update_array string array at the same index as it's timestamp in time_array. Lastly, the set_scroll_view() method is called based on what index of the spinner is selected.

The listener for the spinner is quite simple and essentially sets a boolean that indicates the order in which the user would like to view system updates. This boolean value is passed as an input to the set_scroll_view() method.

The set_scroll_view() method populates the scroll view widget. This method takes in one input boolean that determines the order in which the update messages will be displayed. If true is passed as the input then messages will be sorted and displayed from newest to oldest and if the boolean is false then messages will be sorted from oldest to newest. To populate the scroll view both time_array and update_array are needed. For each timestamp and message pair a linear layout is created that houses a TextView widget to display the timestamp and a TextView widget to display the update message text. The text color of these widgets are based on the type of update. In order to change the text color key words are looked for in each update message. A divider line is inserted between each update to help with readability.

The last listener within this activity is the onClickListener associated with the clear button. By clicking this button the user can clear the updates stored on the Firebase. When the user clicks on the button an AlertDialog is created asking the user if they really want to delete the update history. The dialog window provides two options, yes or no. If the user clicks yes then the updates child is deleted from the Firebase using the removeValue() method. If the user clicks no then nothing happens and the dialog window is closed and the user is returned to the updates activity.

Graph

The graph activity utilizes an imported widget called GraphView. It has two series (collections of data points), one for the setpoint and one for the temperature. The graph is updated every time the database values are updated, typically every second from the Raspberry Pi temperature sensor updating. The setpoint and the temperature data points can be shown or hidden using a checkbox.

Settings

The settings activity allows the user to change certain thermostat. The first setting allows the user to turn on and off the text to speech functionality for their device only. This setting is saved in the devices shared preferences and is retrieved from shared preferences in the onCreate() and onStart() methods whenever an activity has text to speech integration. The text to speech setting is controlled using a button. The text to speech button background changes based on whether text to speech is on or off. Text to speech is set to on by default.

In order to keep the location setting consistent throughout all devices it is stored in the Firebase. To change this setting the user can press a button with text indicating the current location of the thermostat. When the user clicks the button an AlertDialog window box is created prompting the user to enter a new location for the thermostat. The user then can choose to change the location by entering a new location and clicking “OK”. When the “OK” button is pressed the text that the user has input is written to the Firebase child “current_location” overwriting the previous location. There is no validation mechanism so the user may enter an incorrect location if they choose. That being said, if an incorrect location is used then the outside temperature that is displayed in the main activity will read “N/A”.

Raspberry Pi Java App

The Android Things application is very simple. It runs a thread with a 1 second timer to read the temperature sensor from the I2C device and update its value in the database. It concurrently reads the database on the event change listener to get the temperature, setpoint, power state, and fan mode. It then sets the LED color and motor PWM and writes their values through the I2C device.

Challenges

This project presented several challenges to the group. One of the major challenges was getting the PMOD HB3 module to work with the system from project 3. The problem was that the direction bit for the HB3 did not correctly set the direction of the motor this was becasue that specific module was damaged and once the group procured a different PMOD module the direction bit functioned as expected. Another challenge was using various custom Java packages for displaying a real time graph as well as custom seekbars to adjust the thermostat setpoint and system fan mode. Group members spent many hours combing through package documentation in order to implement these custom packages not only effectively but also correctly. Lastly, the largest challenge of the project was data synchronization not only between the Android application and the Raspberry Pi application but also between all the mobile devices running the Android application. This challenge was solved by meticulously mapping data flow as well as creating instance variable names that helped depict the difference between local values and system wide values. The use of boolean variables was also used which is highlighted within the project report.

Resources/References

Java Code

- Device Rotation Lock: https://riptutorial.com/android/example/21077/lock-screen-s-rotation-programmatically
- Text-to-Speech: https://stacktips.com/tutorials/android/speech-to-text-in-android

Custom Java Packages

- GraphView: https://github.com/jjoe64/GraphView/wiki
- CircleSeekBar: https://github.com/feeeei/CircleSeekbar
- BubbleSeekBar: https://github.com/woxingxiao/BubbleSeekBar