



Building Automation

11/06/2020

Niharika Gouda

Course: Software Engineering for Autonomous Systems

Università degli Studi dell'Aquila A.A. 2019-2020

Under the guidance of **Prof. Davide Di Ruscio**

Building Automation

Overview

Benefits

- Lower Energy Costs
- Improved comfort
- Increase productivity
- Reduce maintenance costs
- Increased security
- Data collection
- Low environmental impact

Technological Specifications

- Smartphone with Bluetooth Technology
- Temperature Sensors
- CO2 sensors
- Humidity sensors
- Control Unit
- Motors for Opening Windows
- Bluetooth Lamp Holder

Scenarios

1. Switching on/off of fans during summer season and hot climate
2. Switching on/off of fans during winter season
3. Switching on of lights when its dark during night
4. Switching on of lights when CO2 value is above 1000 ppm
5. Opening and closing of windows according to humidity

Diagrams

- Component Diagram
- Interaction Diagram
- Sequence Diagrams:
 - SD1: General MAPE-K flow
 - SD2: Working of CO2 sensors
 - SD3: Working of Humidity sensors
 - SD4: Working of Temperature sensors

From code to application

- Tools used
- Software installation
- Using the simulator
- Configuration of openHAB
- Using openHAB gui

Building Automation

Overview

In this project, we are focusing on implementing new idea of improving the quality of life of household, resulting in economic savings and provides an impact on more sustainable environment. We are trying to detect and solve few problems concerning switching on/off of few electronic objects and closing/opening of windows during time of day and various seasons.

The main attributes of our data are time (to understand day or night and month of the year), temperature, relative humidity, light, CO2 and occupancy.

Benefits of Smart Building Automation

1. Lower energy costs

With the help of a building automation system(BAS), HVAC and electrical systems are run more efficiently, reducing energy consumption and utility bills. A BAS not only helps to automate a building's systems with sensors and timers, but many collect data regarding the building's energy usage, which can be studied for opportunities for potential improvements. This reduction in energy usage not only reduces costs, but also reduces the building's environmental footprint, which is a benefit for everyone.

2. Improved comfort

A BAS automatically adjusts a building's temperature and lighting, maintaining an ideal level of comfort within the facility. By increasing comfort levels, a building automation system also reduces complaints from occupants and can even help with productivity.

3. Increase productivity

Not only are employees more productive when they're comfortable, but because a BAS allows your systems to run more efficiently, they are less likely to break down and require repairs which can be disruptive to a business's day-to-day operations.

4. Reduce maintenance costs

When controlled by a BAS, a building's systems tend to experience less wear and tear, and therefore last longer and require less repairs. However, it's important to still have your systems serviced regularly, regardless of whether or not you use a BAS.

5. Increased security

Building automation systems can be connected to security systems in order to increase safety and peace of mind. These systems can be used to live monitor what is happening within a facility, provide alerts of suspicious activity, or have the doors lock and unlock at specified times.

6. Data collection

A building automation system allows you to monitor and control your building's systems from anywhere, anytime.

7. Low environmental impact

By regulating gas, temperature and opening and closing of objects helps in maintaining eco-friendly environment.

Also, we are trying to solve the following things with the help of our smart building automation systems:

1. Wastage of resources: Light and fan resources run on electricity every time. We tend to forget to switch off them when not in use. With the help of this system, we use some parameters which helps in saving some energy.
2. Comfort: Lights and fans can be switched according to season and time of the day.
3. Check judicious level of CO₂: We can check if the level of CO₂ in the room is how much, based on that scenarios we can understand that either there are lot of people in the room and perform other operations like switching on light and fan. Because high Exposure to CO₂ can produce a variety of health effects. These may include headaches, dizziness, restlessness, a tingling or pins or

needles feeling, difficulty breathing, sweating, tiredness, increased heart rate, elevated blood pressure, coma, asphyxia, and convulsions.

Technological Specifications

Our system will consist of the following devices:

1. Smartphone with bluetooth technology:

Particularly widespread, they are the main actors. They communicate with the sensors installed in the rooms, indicating the presence / absence of the occupant in the room. However, we are not using this device.

2. Temperature sensors:

They are installed in every room in the house. They send the presence or absence signal of the occupant to the control unit. A temperature sensor is an electronic device that measures the temperature of its environment and converts the input data into electronic data to record, monitor, or signal temperature changes. There are many different types of temperature sensors. Some temperature sensors require direct contact with the physical object that is being monitored (contact temperature sensors), while others indirectly measure the temperature of an object (non-contact temperature sensors).

3. Humidity sensors:

They are installed in every room in the house. They send the signal to the control box in case detect the presence of an occupant. A humidity sensor is a device that detects and measures water vapor. There are a range of calibrated and amplified sensor products that measure relative humidity (RH).

4. CO₂ sensors:

Installed by operators specialized in safety in environments identified as sensitive. They send the detection signal to the control unit. A carbon dioxide sensor or CO₂ sensor is an instrument for the measurement of carbon dioxide gas. The most common principles for CO₂ sensors are infrared gas sensors (NDIR) and chemical gas sensors. Measuring carbon dioxide is important in monitoring indoor air

quality, the function of the lungs in the form of a capnograph device, and many industrial processes.

5.Control unit:

It deals with processing the signals relating to the temperature, humidity and CO2 sensors. Each signal is suitably processed and forwarded to the predisposed device the implementation of planned procedures, such as motors for opening windows. It is programmed through the graphic interface of Openhab, which allows to set the preferences of the occupants.

6.Motors for opening windows:

They receive the signal from the control unit in case of humidity detection. They open automatically windows in rooms where installed.

7.Bluetooth lamp holder:

They receive the signal from the control unit when a presence is detected occupant in the room where present. They automatically turn on the lights in the room where installed.

Scenarios

1. Switching on of fans during summer season and hot climate

While reading the digital reader value of temperature sensors, if the value is greater than 30 and if the months of the year are from May to July, the control unit will instruct an actuator to switch on the fans. Building automation system takes care of this.

2. Switching off of fans during winter season

While reading the digital reader value of temperature sensors, if the value is the months of the year from January to April and then from August to December and temperature is between 20- 24, the control unit will instruct an actuator to switch on the fans. Building automation system takes care of this.

3. Switching on of lights when its dark during night

While reading the digital reader value of CO₂ sensors, irrespective of CO₂ value if the hour of the day is ≥ 17 and less than ≤ 24 , the lights will be automatically switched on. The control unit will instruct an actuator to switch on the lights. Building automation system takes care of this.

4. Switching on of lights when CO₂ value is above 1000 ppm

While reading the digital reader value of CO₂ sensors, if the value of CO₂ is above 1000 ppm, the lights will be automatically switched on. That indicates that many people are there in the room, hence there is requirement of lights. Therefore, we switch on the lights in accordance with the principle of occupancy detection. The control unit will instruct an actuator to switch on the lights. Building automation system takes care of this. According to this paper: *“Occupancy detection in non-residential buildings – A survey and novel privacy preserved occupancy monitoring solution”*, its given that: “occupancy” as the total number of people present in a defined part of a building (we will use Erickson et al.’s definition in this paper). However, some authors define occupancy as the “total number of present people in a space and their action (taken or not) against the indoor environment”. Energy saving was further increased from 9% up to 30–40% in Erickson et al. work . In real-time occupancy data was also used as an input to the HVAC and surprisingly energy saving was between 29% and 80%.

5. Opening and closing of windows according to humidity

During summers if the humidity value in the room is greater than 70 and less than 100 or during winters if the reading of the digital reader value of humidity sensors is less than 20. The control unit will instruct an actuator to close all the windows in the house. Building automation system takes care of this.

Diagrams

The following diagrams clarify the operation and implementation of the system realized:

1. Component Diagram

The architectural style adopted for the realization of Building Automation is based on the MAPE-K loop. Our architecture is therefore composed of five autonomous components principals that interact with each other. The data is monitored by the 'Monitor' component, analyzed by the component 'Analyzer' and, on the basis of policies based on self-awareness on environmental awareness, some actions are planned by the component 'Planner' and run from the 'Executor' component. Planner has the required knowledge useful for performing the execution activities of sensors and display in openHAB.

Another important component is 'OpenHAB'. From this screen it is possible to set the preferences for temperature and acts as an output where the user can control in each moment the situation in each room.

The 'MQTTConnection' component takes care of starting, managing and closing a connection MQTT.

Monitor: The monitor component constantly monitors context data from sensors and other entities. During the monitoring phase, the component performs a pre-analysis of the incoming data based on defined thresholds to evaluate the significance of changes within the data (filtering of jitter). In case there is a significant delta in relevant context data, the new values (Symptoms) are fed forward into the Analyzer component.

Analyze: If the objectives and therefore the overall process goal cannot be satisfied, an inconsistency is assumed and a change request including a description of the mismatch is sent to the Planner component. In case the overall goal can be met (i. e., the patterns defined in the assertions can be detected within the symptoms), the process engine continues with the execution of the consistent process instance based on the process model.

Plan: The mismatch information currently contains simplified descriptions of the deficits (e. g., value is too high or too low) that occurred. The Planner component is able to interpret these deficits semantically and to derive change plans containing parameters and commands/processes to be executed by the replacement component(s). If the luminance values in a specific room have not reached a certain threshold after process step execution (too low), an alternative actuator (e. g., light switch or light dimmer) able to manipulate this physical value in the same room is selected. Based on the too low-mismatch and its capabilities, the actuator has to be switched on (light switch) or powered up (light dimmer) to increase the current luminance values—thus to restore cyber-physical consistency.

Execute: The executor component receives the change plan from the planner to execute the derived compensation/change actions. As we follow a process-based approach, this involves adapting the current process instance and requesting the basic process engine to execute this instance.

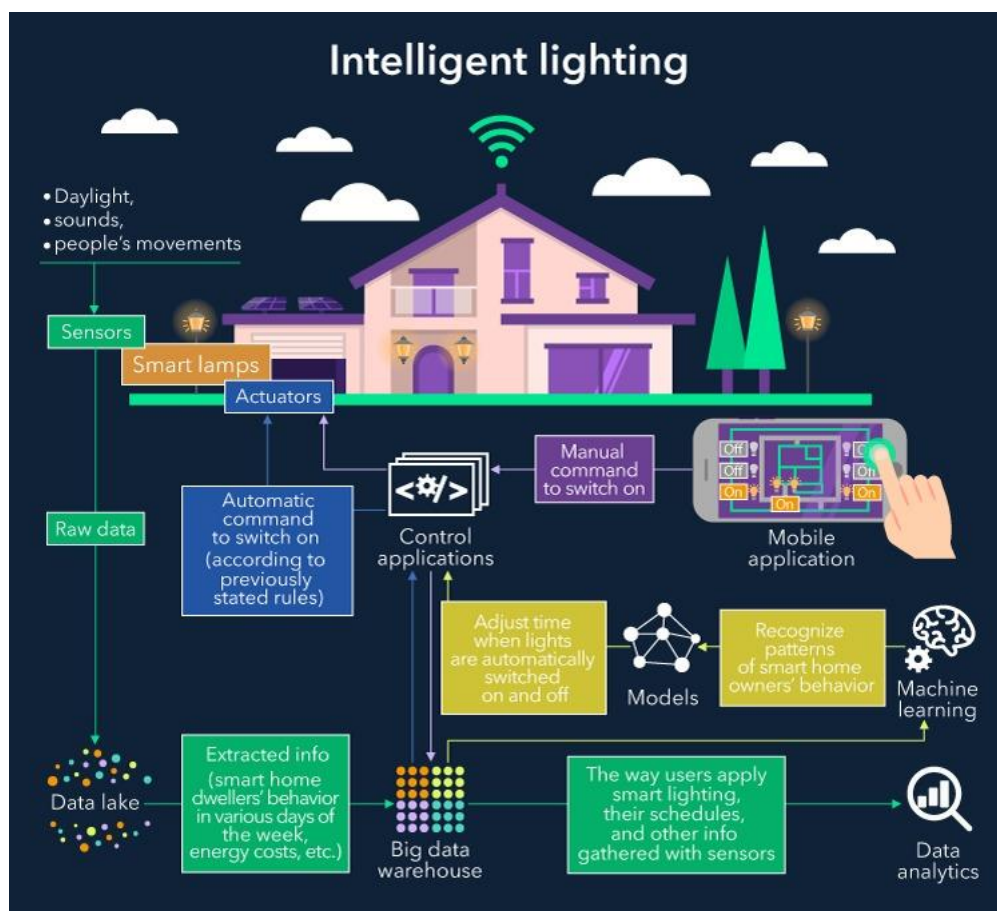
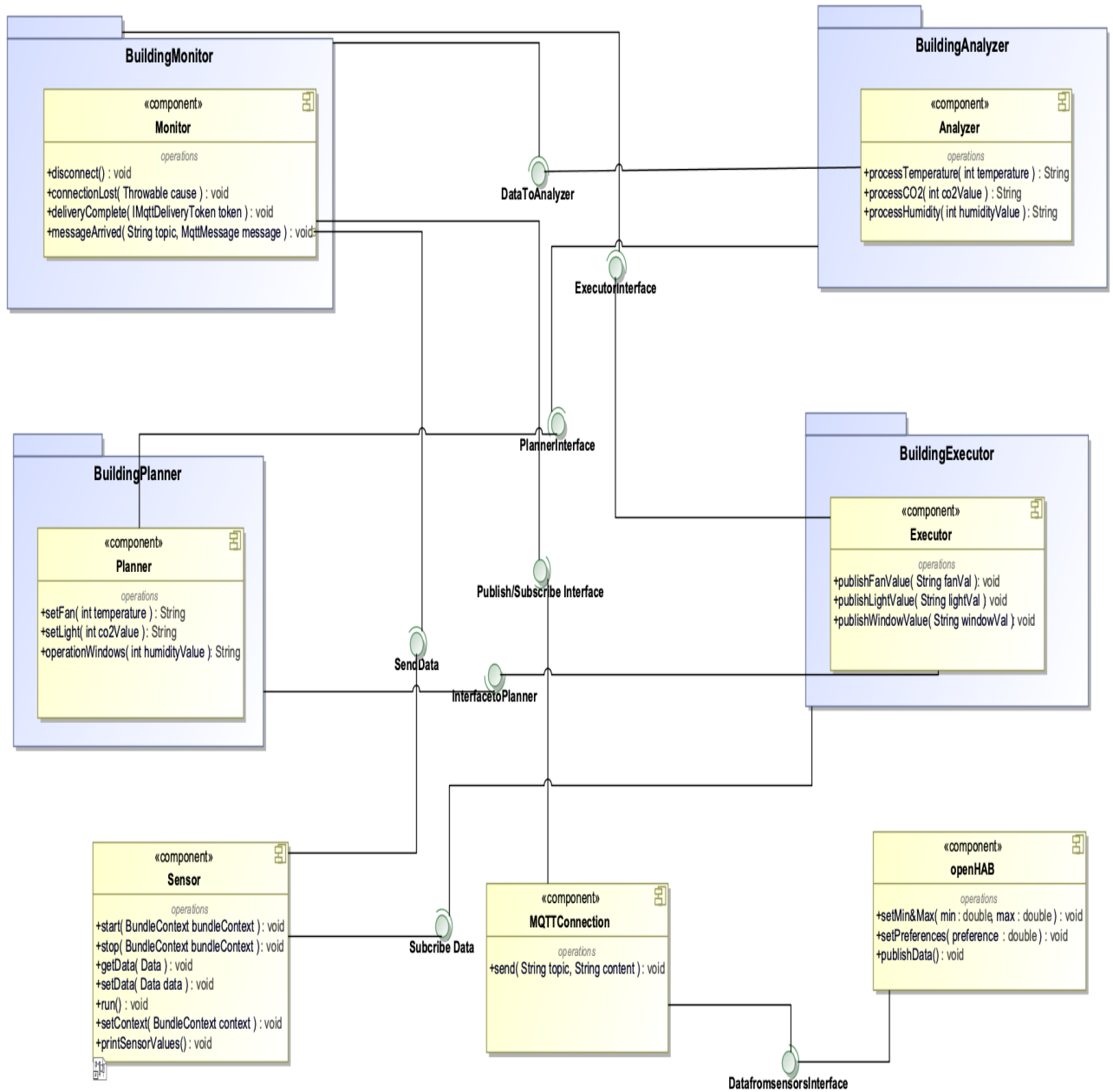
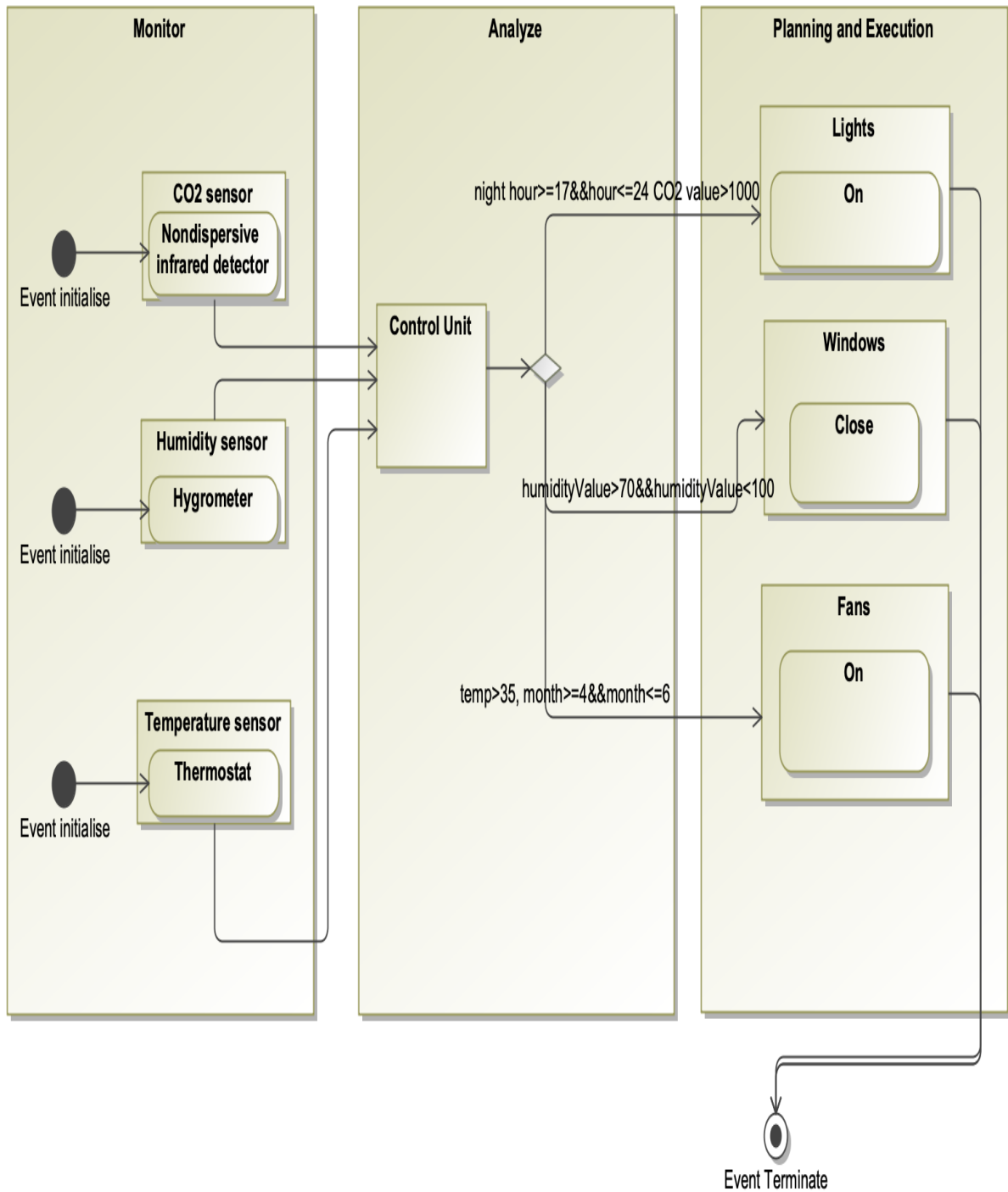


Fig: Example of MAPE-K loop



2. Interaction Diagram

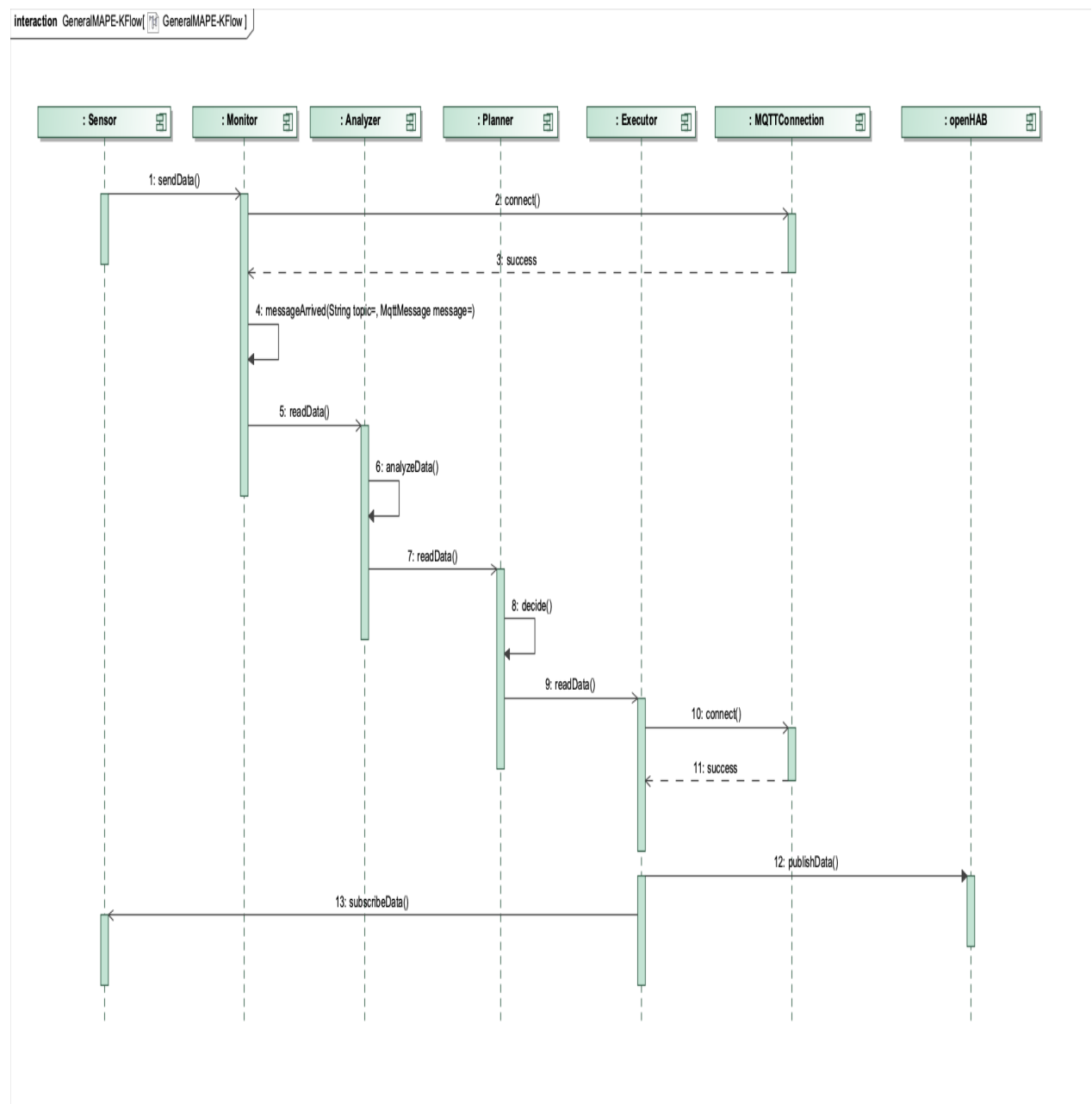
[activity BuildingAutomation[BuildingAutomation]]



3. Sequence Diagrams

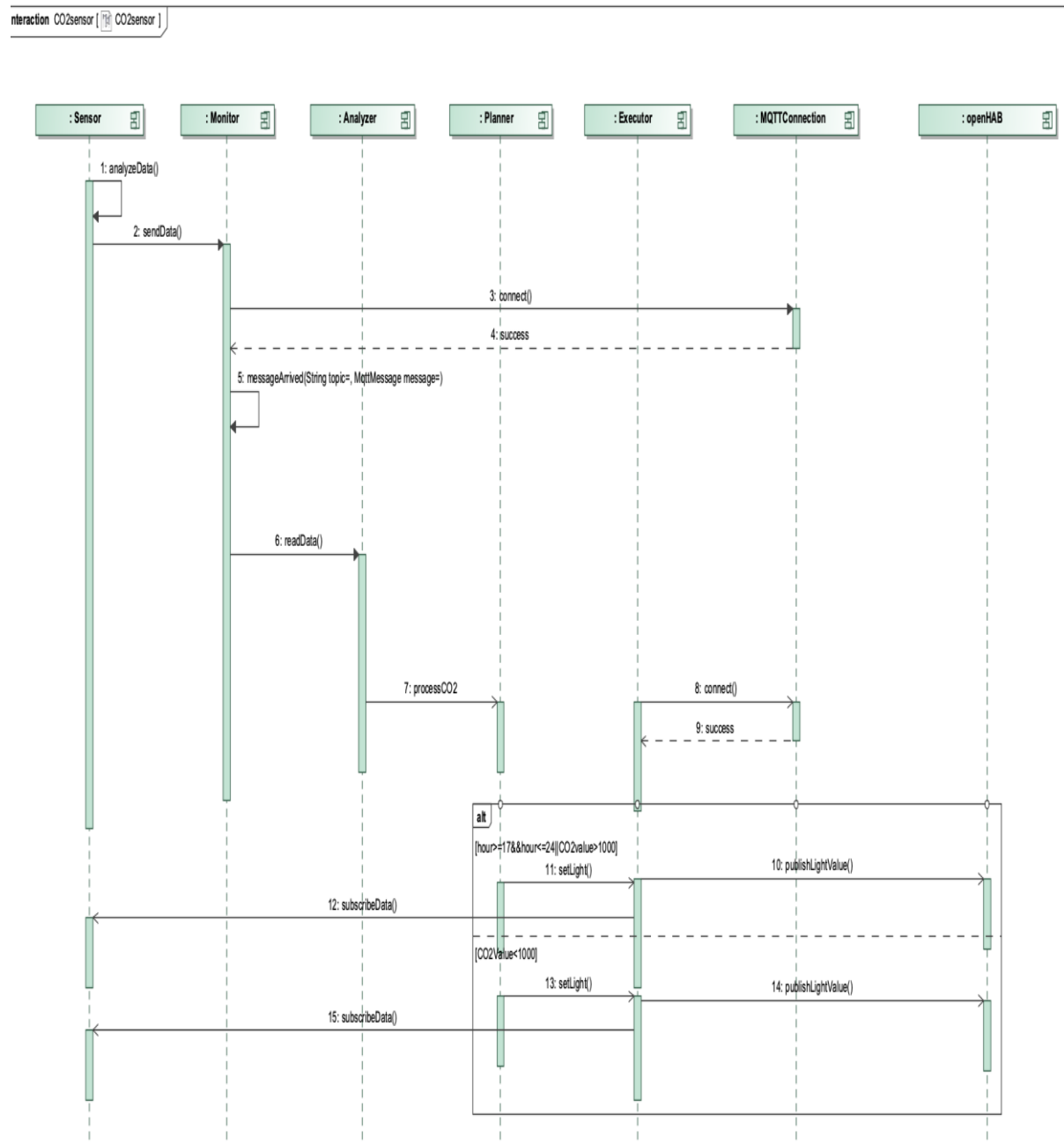
SD1: General MAPE-K flow

The following sequence generally shows how our system works based on MAPE-K. The data, coming from the sensors of the house, are monitored and filtered. The data is then analysed and, in according to some policies, some actions are planned and executed.



SD2: Working of CO2 sensor

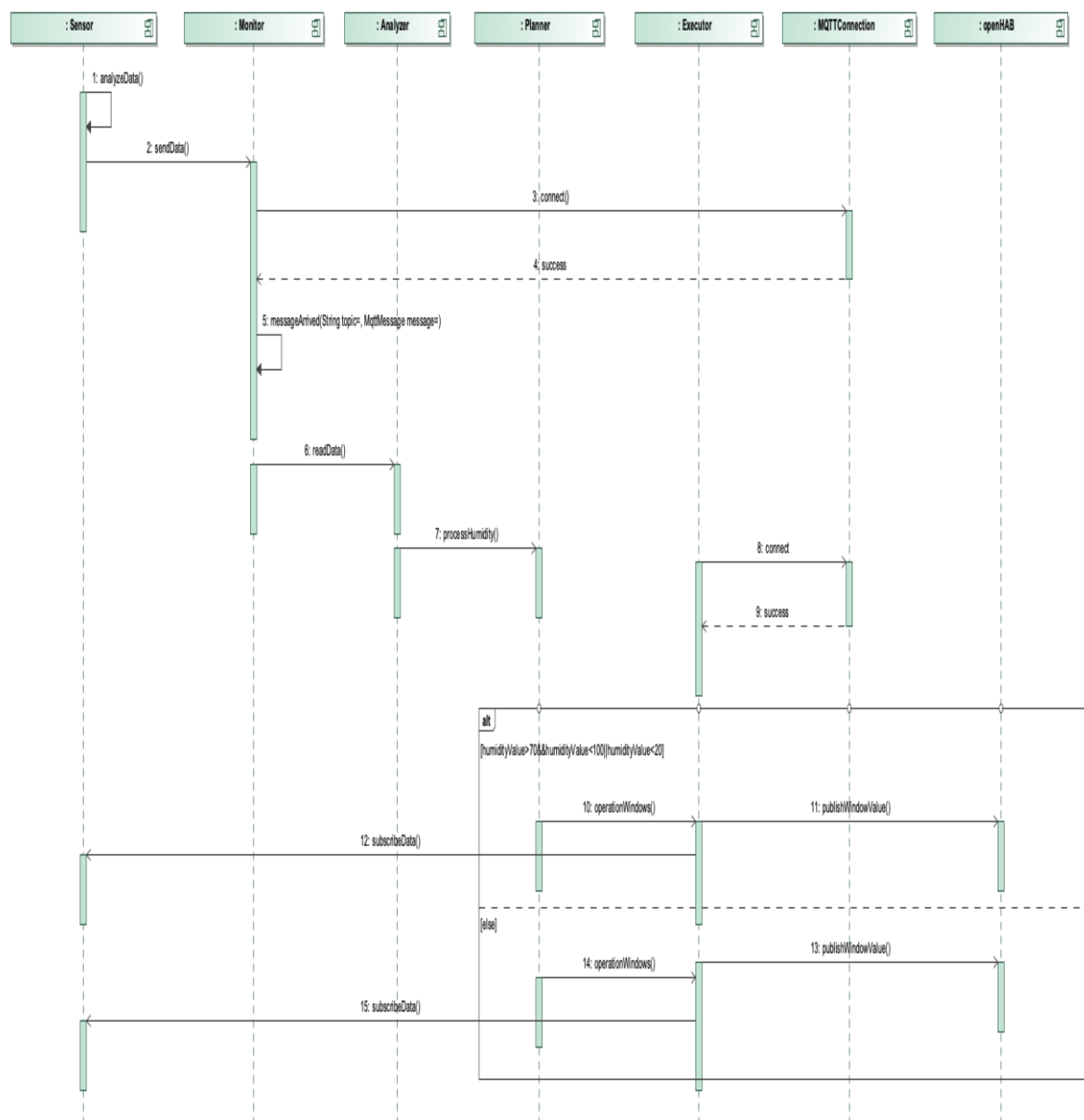
The following sequence shows how our system works during the presence of CO2 in a room. Once CO2 is sensed beyond some level, the lights can be switched on/off based on time of the day.



SD3: Working of Humidity sensor

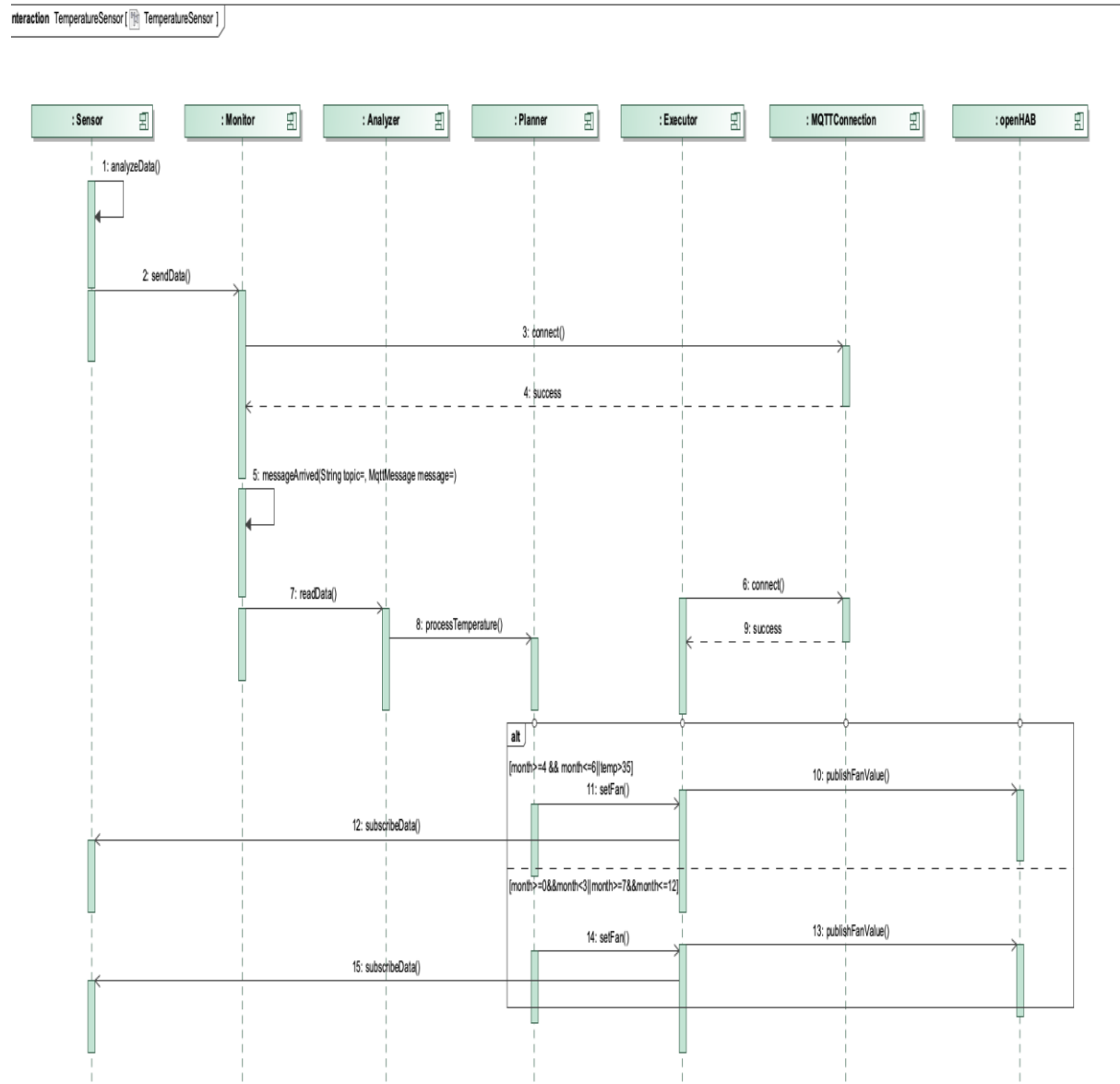
The following sequence shows how our system works during the presence of humidity in a room. Humidity is measured and according to its level, windows are opened or closed.

Interaction HumiditySensor[HumiditySensor]



SD4: Working of Temperature sensor

The following sequence shows how our system works based on temperature in a room. Temperature is measured with the help of thermostat and according to its level, based on the month of the year, whether its summer and winter and based on high and low temperature, the fans will be switched on/off.



From code to application

My project is available here:

<https://github.com/niharikagouda/autonomousproject/blob/master/BuildingAutomation.zip>

Tools used

OpenHab 2 - <https://www.openhab.org/>

Eclipse Mosquitto - <https://mosquitto.org/>

Eclipse Java 2019-20





MQTTBox

TextEditors to update demo.sitemap and demo.items

Software installation

To install the project you need to follow these simple steps:

- Download repository <https://github.com/niharikagouda/autonomousproject/blob/master/BuildingAutomation.zip> and unzip.
- Import the following projects into an eclipse workspace and also add mqtt-paho jar in the pom file:
 - BuildingAnalyzer
 - BuildingExecutor
 - BuildingMonitor
 - BuildingPlanner
- Download the openHAB folder and bundles folder.
- Bundles should be plugin project, ensure the five jars are mentioned as target in the file.

	org.apache.felix.gogo.command_1.0.2.v20170914-1324.jar
	org.apache.felix.gogo.runtime_1.1.0.v20180713-1646.jar
	org.apache.felix.gogo.shell_1.1.0.v20180713-1646.jar
	org.eclipse.equinox.console_1.3.200.v20181115-0906.jar
	org.eclipse.osgi_3.13.300.v20190218-1622.jar
	org.eclipse.paho.client.mqttv3-1.2.4.jar

- We have three sensors; hence mention we have three osgi configuration which runs with the main sensor api class and simplehomesensor class along with the required sensor class.
- Right click on the six plugin projects you will get options to export as deployable plugins and fragments, then we can see these projects as bundles, download them to one folder and add that folder in addons folder of openHAB-runtime.
- Start Mosquitto.
- Start OpenHab via start.bat or start.sh (depending on your operating system). Reach the BasicUI interface.
- For changing the basic UI interface, we need to add items in the file demo.sitemap and demo.items file.
- In the openHAB paper UI, make required configurations as bindings and give the topics for the publisher and subscriber and set their format, maximum and minimum values.
- For the first execution of the project start the bundles as right-click and execute as run configurations or osgi-framework. We have three sensors, hence there are three configurations of osgi.
- After running the three bundles we should be able to see in the eclipse console the execution messages of the respective applications.
- Run the 'MonitorRunner.java' class as it has the main class as a java application on eclipse to start the simulator.
- Next on MQTTBox, add subscriber and their topics to inspect the values.
- According to the values, the light, fan and windows will be switched on/off and open/close respectively.
- In openHAB we can also see the graphs and today's current date in the openHAB window.

- Also, in paper ui, we can see the given input values of sensor as well as the output values in results tab.
- The results are achieved as per the planner which also has some knowledge regarding what the appropriate climatic conditions should be to run it better.

Using the Simulator

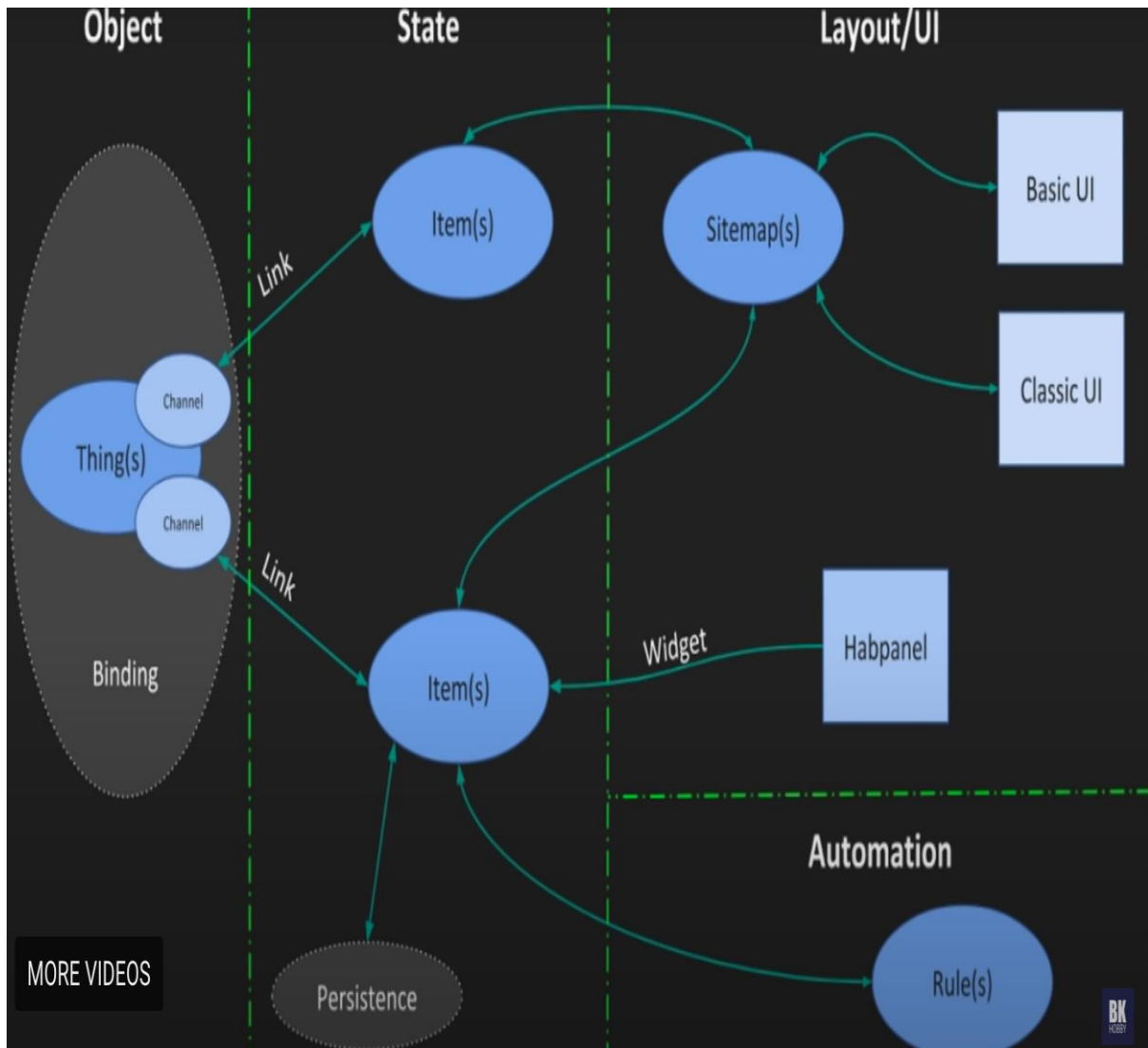


Fig: Openhab Links

Analyzer code

```
1. package it.univaq.disim.se4as.building.analyzer;
2.
```

```

3. import it.univaq.disim.se4as.building.planner.Planner;
4.
5. public class Analyzer {
6.
7.     private Planner planner = new Planner();
8.
9.     public String processTemperature(int temperature) {
10.         return planner.setFan(temperature);
11.     }
12.
13.
14.     public String processCO2(int co2Value) {
15.         return planner.setLight(co2Value);
16.     }
17.
18.
19.     public String processHumidity(int humidityValue) {
20.         return planner.operationWindows(humidityValue);
21.     }
22.
23.
24. }

```

Executor code

```

1. package it.univaq.disim.se4as.building.executor;
2.
3. import org.eclipse.paho.client.mqttv3.MqttClient;
4. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
5. import org.eclipse.paho.client.mqttv3.MqttException;
6. import org.eclipse.paho.client.mqttv3.MqttMessage;
7. import org.eclipse.paho.client.mqttv3.MqttPersistenceException;
8.
9. public class Executor {
10.     private final String broker = "tcp://localhost:1883";
11.     private final String clientId = ""; // private final String clientId = "MQTT Ex
ecution";
12.     private final int qos = 2;
13.     private MqttClient sampleClient = null;
14.
15.     public Executor() {
16.         try {
17.             sampleClient = new MqttClient(broker, clientId);
18.             MqttConnectOptions connOpts = new MqttConnectOptions();
19.             connOpts.setCleanSession(true);
20.             sampleClient.connect(connOpts);
21.         } catch (Exception e) {
22.             System.out.println("Exception occurred: "+e);
23.         }
24.
25.     }
26.
27.     public void publishFanValue(String fanVal) throws MqttPersistenceException, Mqt
tException {
28.         MqttMessage message = new MqttMessage(fanVal.getBytes());
29.         message.setQos(qos);
30.         sampleClient.publish("sm/fan", message);
31.
32.     }
33.     public void publishLightValue(String lightVal) throws MqttPersistenceException,
MqttException {
34.         MqttMessage message = new MqttMessage(lightVal.getBytes());

```

```

35.         message.setQos(qos);
36.         sampleClient.publish("sm/light", message);
37.     }
38.
39.     public void publishWindowValue(String windowVal) throws MqttPersistenceException, MqttException {
40.         MqttMessage message = new MqttMessage(windowVal.getBytes());
41.         message.setQos(qos);
42.         sampleClient.publish("sm/window", message);
43.     }
44.
45.
46. }

```

Monitor Code

```

1. package it.univaq.disim.se4as.building.monitor;
2.
3. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
4. import org.eclipse.paho.client.mqttv3.MqttCallback;
5. import org.eclipse.paho.client.mqttv3.MqttClient;
6. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
7. import org.eclipse.paho.client.mqttv3.MqttException;
8. import org.eclipse.paho.client.mqttv3.MqttMessage;
9.
10. import it.univaq.disim.se4as.building.analyzer.Analyzer;
11. import it.univaq.disim.se4as.building.executor.Executor;
12.
13. public class Monitor implements MqttCallback {
14.     private final String broker = "tcp://localhost:1883";
15.     private final String client = "";
16.     private MqttClient mqttclient = null;
17.
18.     private Analyzer analyzer = new Analyzer();
19.     private Executor executor = new Executor();
20.
21.     public Monitor() throws MqttException {
22.         System.out.println("Monitoring service executing");
23.         this.mqttclient = new MqttClient(broker, client);
24.
25.         MqttConnectOptions connOpts = new MqttConnectOptions();
26.         connOpts.setCleanSession(true);
27.
28.         this.mqttclient.connect(connOpts);
29.         this.mqttclient.setCallback(this);
30.
31.         for (Topics topic : Topics.values()) {
32.             this.mqttclient.subscribe(topic.getTopic());
33.         }
34.     }
35.
36.     public void disconnect() throws MqttException {
37.         this.mqttclient.disconnect();
38.         System.out.println("Monitoring service disconnect.");
39.     }
40.
41.     public void connectionLost(Throwable cause) {
42.         // TODO Auto-generated method stub{
43.         System.out.println("Monitoring service lost MQTT connection.");
44.         cause.printStackTrace();
45.         System.exit(0);
46.     }
47. }
48.

```

```

49.     public void deliveryComplete(IMqttDeliveryToken token) {
50.         System.out.println("Delivery complete. Token: " + token);
51.
52.     }
53.
54.     public void messageArrived(String topic, MqttMessage message) throws Exception
55.     {
56.         // TODO Auto-generated method stub
57.         String[] parts = topic.split("/");
58.         switch (parts[1]) {
59.             case "temperature":
60.                 String temperature = new String(message.getPayload());
61.                 String fanVal = analyzer.processTemperature(Integer.parseInt(temperatur
62. e));
63.                 System.out.println("Niharika.tmp "+fanVal);
64.                 executor.publishFanValue(fanVal);
65.                 break;
66.             case "co2":
67.                 String co2 = new String(message.getPayload());
68.                 String lightVal = analyzer.processCO2(Integer.parseInt(co2));
69.                 System.out.println("Niharika.co2 "+ lightVal);
70.                 executor.publishLightValue(lightVal);
71.                 break;
72.             case "humidity":
73.                 String humidity = new String(message.getPayload());
74.                 String windowVal = analyzer.processHumidity(Integer.parseInt(humidity))
75. ;
76.                 System.out.println("Niharika.humi "+windowVal);
77.                 executor.publishWindowValue(windowVal);
78.                 break;
79.
80.         }
81.
82.     }
83.
84. }

```

Topics.java

```

1. package it.univaq.disim.se4as.building.monitor;
2.
3. public enum Topics {
4.
5.     WEATHER_TEMPERATURE("weather/temperature"),
6.     WEATHER_HUMIDITY("weather/humidity"),
7.     WEATHER_CO2("weather/co2");
8.
9.
10.     private final String topic;
11.
12.     private Topics(String topic) {
13.         this.topic = topic;
14.     }
15.
16.     public String getTopic() {
17.         return topic;
18.     }
19.
20. }

```

MonitorRunner code

```
1. package it.univaq.disim.se4as.building.monitor.runner;
2.
3. import it.univaq.disim.se4as.building.monitor.Monitor;
4.
5. public class MonitorRunner {
6.     public static void main(String[] args) throws InterruptedException {
7.         try {
8.             new Monitor();
9.             Thread.sleep(10000);
10.        } catch (Exception e) {
11.            e.printStackTrace();
12.            Thread.sleep(2000L);
13.            main(null);
14.        }
15.    }
16. }
```

Planner code

```
1. package it.univaq.disim.se4as.building.planner;
2.
3. import java.util.Calendar;
4.
5. public class Planner {
6.
7.     Calendar cal = Calendar.getInstance();
8.
9.     public String setFan(int temperature) {
10.
11.         int month = cal.get(Calendar.MONTH);
12.
13.         // summer season
14.         if (month >= 4 && month <= 6) {
15.             if (temperature > 35) {
16.                 return "On";
17.             } else
18.                 return "Off";
19.         }
20.         // winter season
21.         else {
22.             if (temperature > 20 && temperature < 24) {
23.                 return "On";
24.             } else
25.                 return "Off";
26.         }
27.     }
28. }
29.
30. public String setLight(int co2Value) {
31.     int hour = cal.get(Calendar.HOUR);
32.     //At night, light will be switched on.
33.     if (hour >= 17 && hour <= 24) {
34.         return "On";
35.     } else if (co2Value > 1000) {
```

```

36.         return "On";
37.     } else
38.         return "Off";
39. }
40.
41. public String operationWindows(int humidityValue) {
42.     if (humidityValue > 70 && humidityValue < 100) {
43.         return "Close";
44.     } else if (humidityValue < 20) {
45.         return "Close";
46.     } else
47.         return "Open";
48. }
49. }

```

Sample bundle code

Sensor.api code

```

1. package it.univaq.disim.se4as.services.sensor.api;
2.
3.
4. public class Data {
5.     String unit;
6.     int val;
7.     String brand;
8.     int accuracy;
9.
10.    public String getBrand() {
11.        return brand;
12.    }
13.    public void setBrand(String brand) {
14.        this.brand = brand;
15.    }
16.    public int getAccuracy() {
17.        return accuracy;
18.    }
19.    public void setAccuracy(int accuracy) {
20.        this.accuracy = accuracy;
21.    }
22.    public String getUnit() {
23.        return unit;
24.    }
25.    public void setUnit(String unit) {
26.        this.unit = unit;
27.    }
28.    public int getVal() {
29.        return val;
30.    }
31.    public void setVal(int i) {
32.        this.val = i;
33.    }
34.
35. }

```

Isensor code

```

1. package it.univaq.disim.se4as.services.sensor.api;
2.
3. public interface ISensor {
4.

```

```

5.     public Data getData();
6.     public void setData(Data data);
7.
8. }

```

Mqtt connection code

```

1. package it.univaq.disim_se4as.services.temperatureactivator;
2.
3. import org.eclipse.paho.client.mqttv3.MqttClient;
4. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
5. import org.eclipse.paho.client.mqttv3.MqttException;
6. import org.eclipse.paho.client.mqttv3.MqttMessage;
7.
8. public class MQTTConnection {
9.
10.     private final String broker = "tcp://localhost:1883";
11.     private final String clientId = ""; // private final String clientId = "MQTT Ex
    ecution";
12.     private final int qos = 2;
13.     private MqttClient sampleClient = null;
14.
15.     public MQTTConnection(){
16.         try {
17.             sampleClient = new MqttClient(broker, clientId);
18.             MqttConnectOptions connOpts = new MqttConnectOptions();
19.             connOpts.setCleanSession(true);
20.             sampleClient.connect(connOpts);
21.         } catch (Exception e) {
22.             System.out.println("Exception occurred: "+e);
23.         }
24.     }
25.
26.     public void send(String topic, String content) {
27.         try {
28.             System.out.println("Executor send message (" + topic + "): " + content)
;
29.             MqttMessage message = new MqttMessage(content.getBytes());
30.             message.setQos(qos);
31.             sampleClient.publish(topic, message);
32.         } catch(MqttException me) {
33.             System.out.println("reason " + me.getReasonCode());
34.             System.out.println("msg " + me.getMessage());
35.             System.out.println("loc " + me.getLocalizedMessage());
36.             System.out.println("cause " + me.getCause());
37.             System.out.println("excep " + me);
38.             me.printStackTrace();
39.         }
40.     }
41.
42.
43.     @Override
44.     public void finalize() throws MqttException {
45.         sampleClient.disconnect();
46.         System.out.println("Executor disconnected");
47.     }
48.
49. }

```

Activator. Java


```

1. package it.univaq.disim_se4as.services.temperatureactivator;
2.
3. import org.osgi.framework.BundleActivator;
4. import org.osgi.framework.BundleContext;
5.
6. public class Activator implements BundleActivator {
7.
8.     private static BundleContext context;
9.     private static SensorProbingThread myThread;
10.
11.
12.     static BundleContext getContext() {
13.         return context;
14.     }
15.
16.     /*
17.      * (non-Javadoc)
18.      * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
19.      */
20.     public void start(BundleContext bundleContext) throws Exception {
21.         Activator.context = bundleContext;
22.         myThread = new SensorProbingThread();
23.         myThread.setContext(context);
24.         myThread.start();
25.     }
26.
27.
28.
29.     /*
30.      * (non-Javadoc)
31.      * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
32.      */
33.     public void stop(BundleContext bundleContext) throws Exception {
34.         Activator.context = null;
35.     }
36.
37.
38. }

```

Sensorprobing.java

```

1. package it.univaq.disim_se4as.services.temperatureactivator;
2.
3. import org.eclipse.paho.client.mqttv3.MqttException;
4. import org.osgi.framework.BundleContext;
5. import org.osgi.framework.InvalidSyntaxException;
6. import org.osgi.framework.ServiceReference;
7.
8. import it.univaq.disim.se4as.services.sensor.api.*;
9.
10. public class SensorProbingThread extends Thread {
11.     private volatile boolean active = true;
12.     private BundleContext context;
13.
14.     private MQTTConnection connection = new MQTTConnection();
15.
16.     public void run() {
17.         while (active) {
18.             try {
19.                 printSensorValues();
20.                 Thread.sleep(5000);

```

```

21.         } catch (InvalidSyntaxException | InterruptedException | MqttException
22.         e) {
23.             // TODO Auto-generated catch block
24.             e.printStackTrace();
25.         }
26.     }
27.
28.     public void setContext(BundleContext context) {
29.         this.context = context;
30.     }
31.
32.     private void printSensorValues() throws InvalidSyntaxException, InterruptedException, MqttException {
33.         ServiceReference<?>[] refs = context.getAllServiceReferences(ISensor.class.getName(), null);
34.
35.         if (refs != null) {
36.
37.             for (int i = 0; i < refs.length; i++) {
38.                 ISensor sensor = (ISensor) context.getService(refs[i]);
39.                 if (refs[i] != null) {
40.                     if (sensor != null) {
41.                         // System.out.println("GAS Sensed value" );
42.                         // System.out.println("Unit: " + sensor.getData().getUnit());
43.
44.                         int temp = sensor.getData().getVal();
45.                         if (temp < 20) {
46.                             for (int k = 0; k < 8; k++) {
47.                                 int data = temp + 9 + k;
48.                                 connection.send(Topics.WEATHER_TEMPERATURE.getTopic(), (data)+"");
49.                                 System.out.println(
50.                                     "Temperature Sensed value: " + (data) + sensor.getData().getUnit());
51.                                 Thread.sleep(5000);
52.                                 // System.out.println("Accuracy: " + sensor.getData().getAccuracy());
53.                             }
54.                         } else
55.                         for (int k = 0; k < 5; k++) {
56.                             connection.send(Topics.WEATHER_TEMPERATURE.getTopic(), (temp + k)+"");
57.                             System.out.println(
58.                                 "Temperature Sensed value: " + (temp + k) + sensor.getData().getUnit());
59.                             Thread.sleep(5000);
60.                         }
61.                     }
62.                 }
63.             }
64.         }
65.     }
66.
67. }
68. //connection.finalize();
69. }
70. }

```

TemperatureSensor.java

```

1. package it.univaq.disim_se4as.services.temperaturesensor;
2.
3. import java.util.Random;

```

```

4.
5. import it.univaq.disim.se4as.services.sensor.api.ISensor;
6. import it.univaq.disim.se4as.services.sensor.api.Data;
7.
8. public class TemperatureSensor implements ISensor {
9.     Data data;
10.
11.     public TemperatureSensor() {
12.         Random random = new Random();
13.         int i = random.nextInt(35);
14.         //int accuracy = random.nextInt(100);
15.
16.         data = new Data();
17.         data.setUnit("°Celsius");
18.         data.setVal(i);
19.         //data.setAccuracy(accuracy);
20.     }
21.
22.     @Override
23.     public Data getData() {
24.         // TODO Auto-generated method stub
25.         return this.data;
26.     }
27.
28.     @Override
29.     public void setData(Data data) {
30.         this.data=data;
31.     }
32.
33.
34. }

```

Configuration of openHAB

Updating the demo.items file added new fields such as humidity, CO2, fans and lights, their icons and channels for input and output of the system

```

/* Windows */
String Window_GF_Frontdoor    "Smart Home Window [%s]"    <window> (GF_Corridor, Windows)    ["FrontDoor"]
Contact Window_GF_Kitchen     "Kitchen [MAP(en.map):%s]"      (GF_Kitchen, Windows)    ["Window"]
Contact Window_GF_Living      "Terrace door [MAP(en.map):%s]"  <door> (GF_Living, Windows)    ["Door"]
Contact Window_GF_Toilet      "Toilet [MAP(en.map):%s]"      (GF_Toilet, Windows)    ["Window"]

Contact Window_FF_Bath         "Bath [MAP(en.map):%s]"      (FF_Bath, Windows)    ["Window"]
Contact Window_FF_Bed         "Bedroom [MAP(en.map):%s]"      (FF_Bed, Windows)    ["Window"]
Contact Window_FF_Office_Window "Office Window [MAP(en.map):%s]" (FF_Office, Windows)    ["Window"]
Contact Window_FF_Office_Door  "Balcony Door [MAP(en.map):%s]" <door> (FF_Office, Windows)    ["Door"]

Contact Garage_Door           "Garage Door [MAP(en.map):%s]" <garagedoor> (Garden, Windows)    ["GarageDoor"]

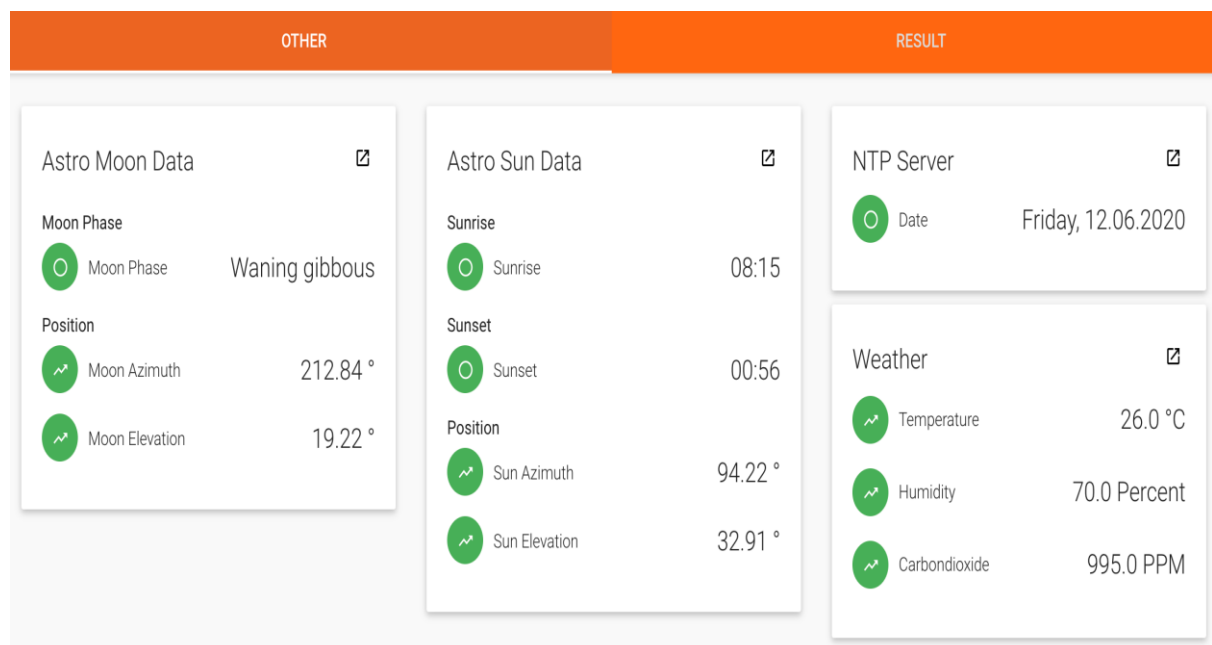
Group Weather_Chart
Number Weather_Temperature    "Temperature [%1f °C]" <temperature> (Weather, Weather_Chart) { channel = "mqtt:topic:7f56c80c:Weather_Temperature" }
Number Weather_Temp_Max       "Todays Maximum [%1f °C]" <temperature> (Weather, Weather_Chart)
Number Weather_Temp_Min       "Todays Minimum [%1f °C]" <temperature> (Weather, Weather_Chart)
Number Weather_Chart_Period    "Chart Period"
DateTime Weather_LastUpdate    "Last Update [%1$ta %1$tR]" <clock>
Number Weather_Humidity        "Humidity [%1f Percent]" <humidity> { channel = "mqtt:topic:7f56c80c:Weather_Humidity" }
Number Weather_CO2             "Carbondioxide [%1f PPM]" <gas> { channel = "mqtt:topic:7f56c80c:Weather_CO2" }
String Lights_SH               "Switch lights [%s]" <light> { channel = "mqtt:topic:abc162bf:Lights_SH" }
String Fan_SH                  "Switch Fan [%s]" <fan> { channel = "mqtt:topic:abc162bf:Fan_SH" }

```

Updating demo.sitemaps with the new items to be displayed

```
1 sitemap demo label="Main Menu"
2 {
3   Frame {
4     Group item=gFF label="First Floor" icon="firstfloor"
5     Group item=gGF label="Ground Floor" icon="groundfloor"
6     Group item=gC label="Cellar" icon="cellar"
7   }
8   Frame label="Weather" {
9     Text item=Weather_Temperature valuecolor=[Weather_LastUpdate=="NULL"="lightgray",Weather_LastUpdate>90="lightgray",>25="orange",>15="green",>5="orange",<=5="blue"] {
10      Frame {
11        Text item=Weather_Temp_Max valuecolor=[>25="orange",>15="green",>5="orange",<=5="blue"]
12        Text item=Weather_Temp_Min valuecolor=[>25="orange",>15="green",>5="orange",<=5="blue"]
13        Text item=Weather_LastUpdate visibility=[Weather_LastUpdate>30] valuecolor=[Weather_LastUpdate>120="orange", Weather_LastUpdate>300="red"]
14      }
15      Frame {
16        Switch item=Weather_Chart_Period label="Chart Period" icon="chart" mappings=[0="Hour", 1="Day", 2="Week"]
17        Chart item=Weather_Chart period=h refresh=600000 visibility=[Weather_Chart_Period==0, Weather_Chart_Period=="NULL"]
18        Chart item=Weather_Chart period=D refresh=3600000 visibility=[Weather_Chart_Period==1]
19        Chart item=Weather_Chart period=W refresh=3600000 visibility=[Weather_Chart_Period==2]
20      }
21    }
22    Text item=Weather_Humidity
23
24    Text item=Weather_CO2
25  }
26 }
27
28 Frame label="Demo" {
29   Text item=CurrentDate
30   Text item=Window_GF_Frontdoor
31   Text item=Lights_SH
32   Text item=Fan_SH
33 }
34 }
35 }
```

Under the framelabel “weather I make all the bindings and give the mqtt connection and the types of values to be returned and max and min of that value. This is for our input values.



Results of our output values are shown.

Control

OTHER

SM: Display

Switch Fan

Off

Switch lights

Off

Smart Home Window

Open

Channels of input

Configuration > Things > Weather

Weather

Generic MQTT Thing

You need a configured Broker first. Dynamically add channels of various types to this Thing. Link different MQTT topics to each channel.

Status: ONLINE

Channels

Weather: Temperature

mqtt.topic:7f56c80c:Weather_Temperature

Number

Weather: Humidity

mqtt.topic:7f56c80c:Weather_Humidity

Number

Weather: Carbon Dioxide Level

mqtt.topic:7f56c80c:Weather_CO2

Number

Configuring temperature channel

Configure channel

Others

MQTT State Topic

weather/temperature

An MQTT topic that this thing will subscribe to, to receive the state. This can be left empty, the channel will be state-less command-only channel.

Absolute Minimum

-10

This configuration represents the minimum of the allowed range. For a percentage channel that equals zero percent.

MQTT Command Topic

weather/temperature

An MQTT topic that this thing will send a command to. If not set, this will be a read-only switch.

Absolute Maximum

100

This configuration represents the maximum of the allowed range. For a percentage channel that equals one-hundred percent.

SHOW LESS

Transform Values

Incoming Value Transformations

Applies transformations to an incoming MQTT topic value. A transformation example for a received JSON would be "JSONPATH:\$device.status.temperature" for a json {device: {status: { temperature: 23.2 }}}}. You can chain transformations by separating them with the intersection character ∩.

Outgoing Value Transformation

Applies a transformation before publishing a MQTT topic value. Transformations are specialised in extracting a value, but some transformations like the MAP one could be useful.

CANCELSAVE

Channels of output

Configuration > Things > SM: Display

SM: Display

Generic MQTT Thing

You need a configured Broker first. Dynamically add channels of various types to this Thing. Link different MQTT topics to each channel.

Status: ONLINE

Channels +

Smart Home: Fan

mqtt.topic:abc162bf:Fan_SH

String

Smart Home: Lights

mqtt.topic:abc162bf:Lights_SH

String

Smart Home: Front Door

mqtt.topic:abc162bf:Window_GF_Frontdoor

String

Configuring light channel

Configure channel

Others

MQTT State Topic

sm/light

An MQTT topic that this thing will subscribe to, to receive the state. This can be left empty, the channel will be state-less command-only channel.

MQTT Command Topic

sm/light

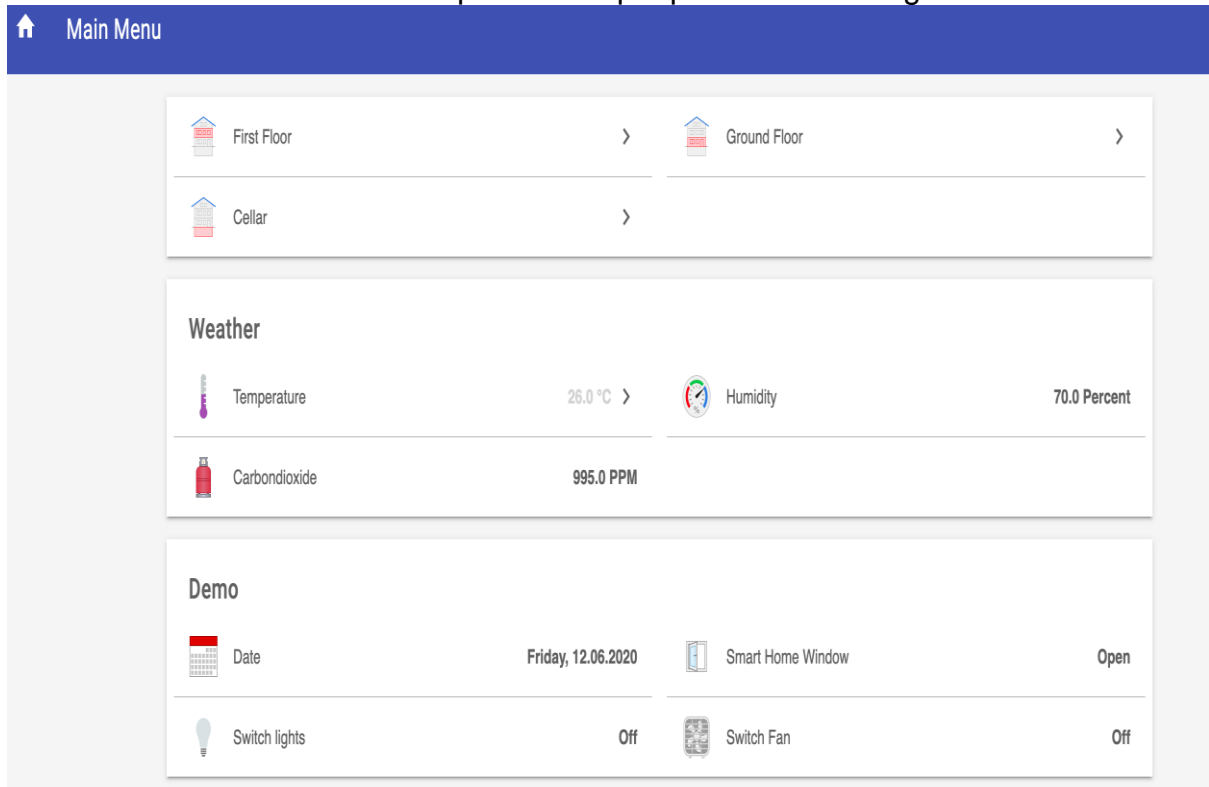
An MQTT topic that this thing will send a command to. If not set, this will be a read-only switch.

SHOW MORE

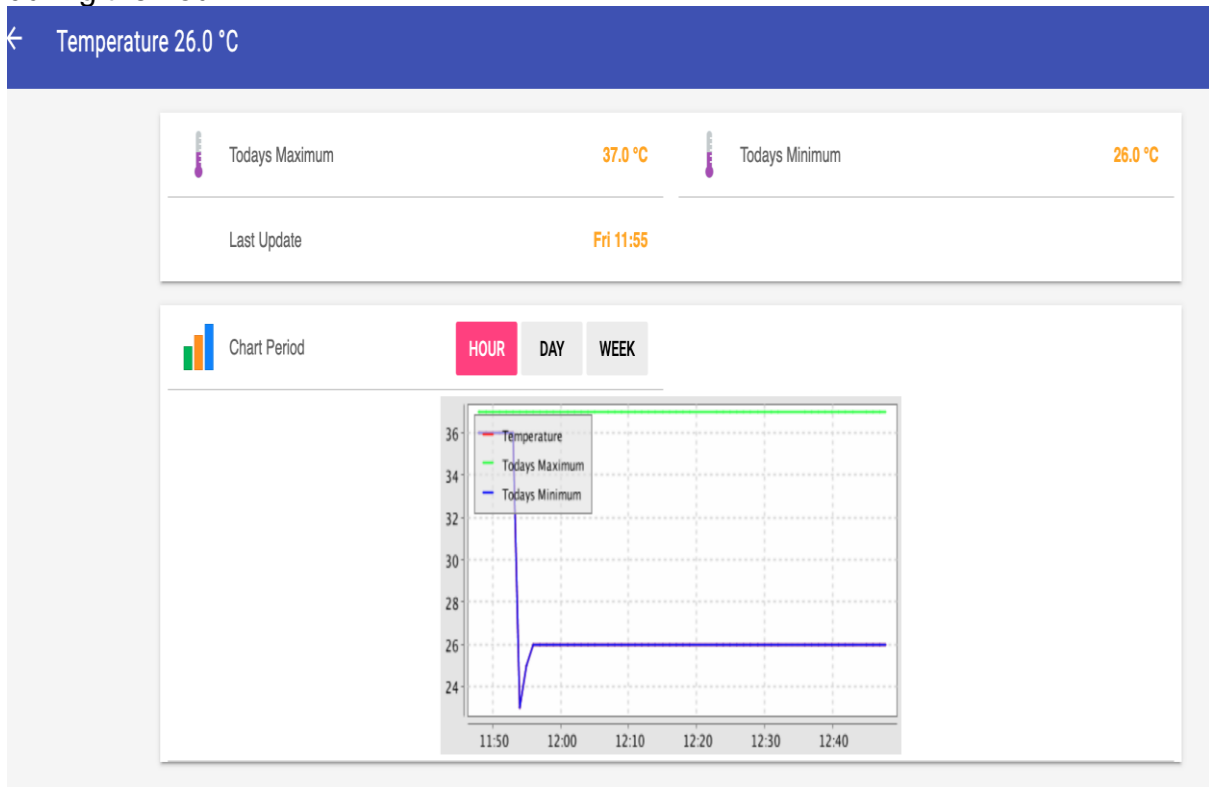
CANCELSAVE

Using openHAB GUI

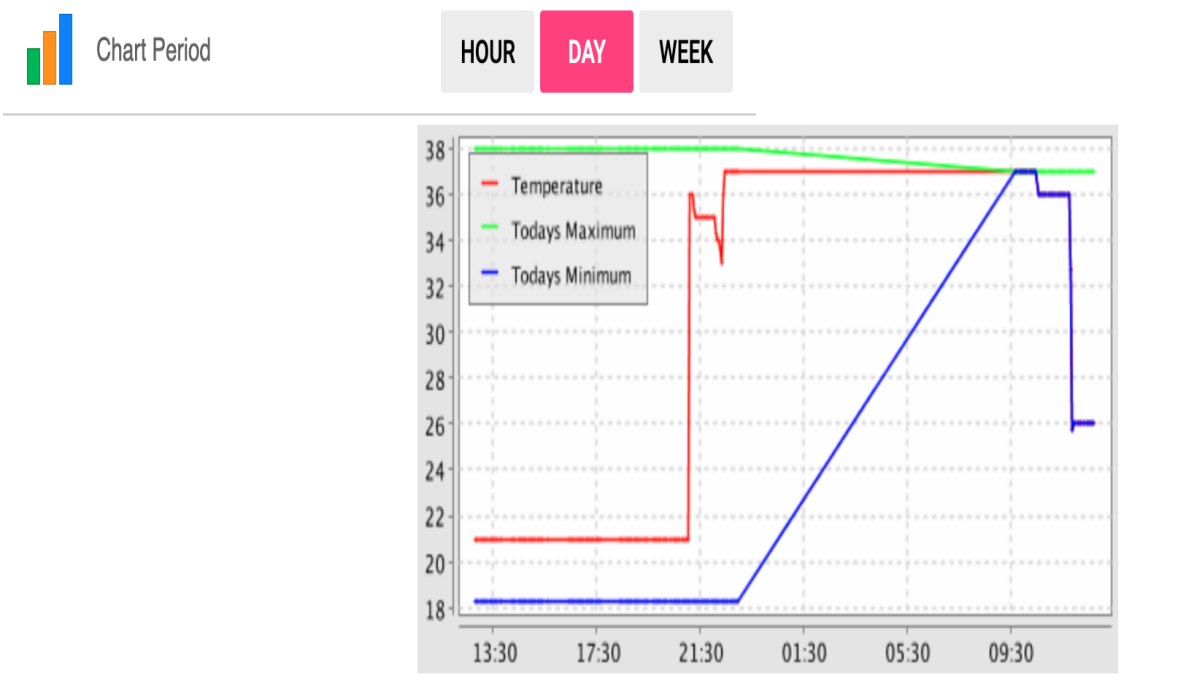
The main menu shows all the input and output parameters along with the date.



In the temperature menu, we also have a chart indicating how was the temperature during the hour.



In the temperature menu, we also have a chart indicating how was the temperature during the day.



In the temperature menu, we also have a chart indicating how was the temperature during the week.

