DEPARTMENT OF INFORMATION ENGINEERING
MSc. ARTIFICIAL INTELLIGENCE & DATA ENGINEERING

Air Sensor Detector

STUDENT:
Michael Asante

COURSE:
Internet Of Things

A.Y. 2021-2022

Github: `https://github.com/Mickey374/IoT_Air_Sensor`

Contents

1 Introduction **1**

2 Application Workflow **2**

3 System Architecture **2**
3.1 CoAP Network . . . 3
3.1.1 Fan Actuator . . . 3
3.1.2 Window Actuator . . . 4
3.2 MQTT Network . . . 4
3.2.1 Temperature-Humidity-CO2 . . . 4
3.2.2 Outside temperature . . . 4
3.3 Database . . . 5
3.4 Data Encoding . . . 5

4 Deployment and Execution **6**
4.1 Collector . . . 6
4.2 Command Line Interface Commands . . . 6
4.3 CoAP Network Implementation . . . 6
4.4 MQTT Network implementation . . . 7

5 Flow of Execution **9**
5.1 Window Flow . . . 9
5.1.1 Open Window Flow . . . 9
5.2 Fan Flow . . . 9
5.2.1 Open Fan Flow . . . 9

6 Data Visualization-GRAFANA **11**

1 Introduction

With the average breathing rate of a normal adult being 12 to 16 rates per minute and the volume (depth of inhalation) being around 500-600ml of air, there are a lot of compositions of air which are harmful to the health of humans. As the world population is increasing to 8 billion, the amount of chemicals, industrial waste and hazardous substances emitted into the atmosphere tends to affect our health on the long run.
The current most-contributing sources of air pollution include emissions from factories, cars, open burning of wastes, pesticides and even commercial and household products. This tends to increase global warming and affect the inhabitants.
The harmful impact of small airborne particles, also known as fine particular matter (PM2.5) that can penetrate deep into the bloodstream and lungs, is becoming increasingly clear. According to the World Health Organization, an estimated 7 million people die annually from air pollution related diseases.
The goal of this project is to create an air detection system that measures these gases and also automates some procedures as a counter measure.The implementation of the project therefore combines sensors and actuators for measuring, storing and performing a counter action within a simulated environment context. The sensors will therefore monitor temperature level, humid air and carbon monoxide in the simulated atmosphere while the actuators will initiate air and filtering control action.
Among other components of the application is the availability of control logic programs which can be initialized by the application user to achieve a specific task giving much flexibility and ease of use to the actor.

2 Application Workflow

The simulation of this application is a house with individuals(eg.elderly people). The logic of the application measures certain components of the Air Quality Index(AQI) to ensure they do not inhale poisonous gases. There are sensors which monitor the temperature, humidity and carbon dioxide. The sensors collect and send the data to a border router, then to a controller which initiates an actuator to close the windows and start a fan(gas extractor) within the environment.
Once the fan starts, the valve managing it changes state into a fanning state. The concept of recharging fans was used in this case, which ensures the fans(extractors) can be recharged once their battery level is low.

3 System Architecture

The overall procedural flow of the IoT solution is represented on *Figure 1*.

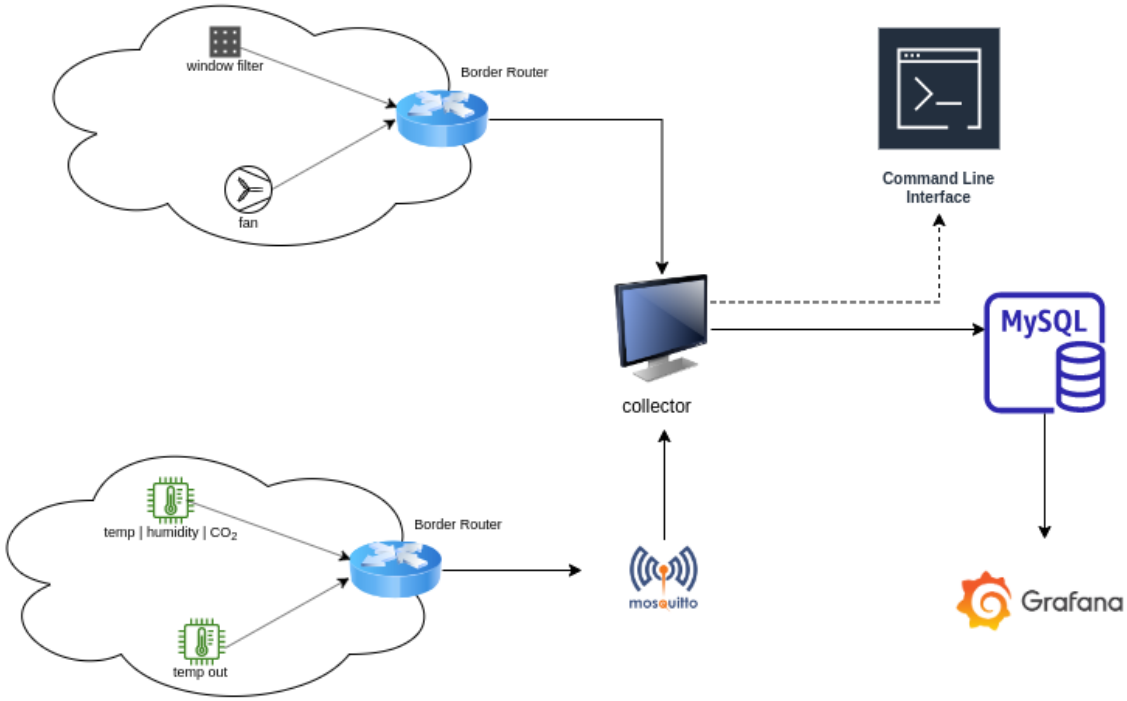

Figure 1: System Architecture for Air Sensor Detector

From the diagram, the IoT architecture consists of 2 parts. The first part uses IoT devices implementing an MQTT framework to retrieve the data while the other part consists uses a COAP framework to communicate with the actuators. The code for both the MQTT and COAP is flashed on nRF52840 dongles. These frameworks are connected to a border-router which enables them to communicate with their external environment. The MQTT network is deployed on the osboxes VM.
The sensors used in the application were temperature-humidity-Co2 and outside temperature while the actuators are windows and fan(gas extractor). The collector(implemented with Python) collects the data from COAP and MQTT and stores it in a MySQL database.

Grafana is used to connect to the MySQL table to visualize all the tables and the data collected. A control logic is also used within the collector to either enable or disable the actuators as well as modify the external environment based on the data collected and the parameter intervals on the sensors. In addition, some commands are exposed to ensure that the user has maximum control over the application through activities such as "help", "activate", "filter", "logs", "simulate", "change params", "exit".

3.1 CoAP Network

The CoAP network on the sensors exposes resources for the application to communicate. Since CoAP provides a request/response interaction model similar to HTTP between application endpoints, it supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and content negotiation.

3.1.1 Fan Actuator

This actuator is used to trigger the Fan acting as a gas extractor. Based on the scope of value recorded, the application communicates to the actuators to start or stop, as well as charge it. The figure below explains the counter behaviour and different state in which this actuator can be. Also the light emitting diode(led) for all possible states are highlighted.

Figure 2: Fan Actuator that opens and closes to extract poisonous gases within a house which can be harmful to one's health.

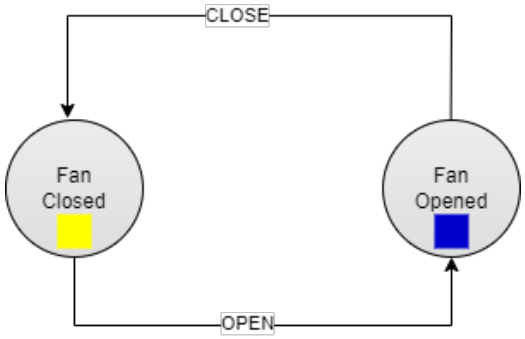

Figure 3: Fan Actuator with led colors based on state.

3.1.2 Window Actuator

This actuator controls the opening and the closing of the windows. Based on the recorded values at a given time, a message is sent to the window actuator to either close or open the windows. Mostly this happens in the cases where poisonous gases will start moving into the house through the windows. The figure below highlights the various states the application can be in as well as the led in those different scenarios.

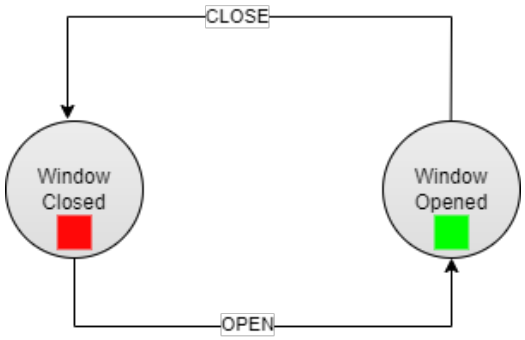

Figure 4: Window Actuator actions and led states.

3.2 MQTT Network

The main aspect of the Air Sensor Detector is the collection of recorded values of temperature, humidity and Co2. The environment under context has a ventilation system that operates outside the environment to be able to record outside temperatures since it can give a trigger decision for the actuators(especially, the windows) to act fast. The MQTT network on these sensors collect and generate, through a simulation, all the values that are used by the CoAP network.

3.2.1 Temperature-Humidity-CO2

The sensors for temp-hum-CO2 sense the values for temperature, humidity and CO2 within house. This ensures that, any temperature changes are recorded for the period of context. In unison to the different actuators, reasonable values are generated. If the fan actuator has started, the temperature will decrease for the humidity will increase. Moreover, the sensors also implements a day and night as a way to inculcate the overall period within this environment.

3.2.2 Outside temperature

This sensor node senses the external temperature within the application. In addition to this, on-going changes are detected. The focus was on only temperature because atmospheric poisonous gases in a way affect the temperature in totality.

3.3 Database

All data flowing within this application are stored within a MySQL database which is in addition connected to a Grafana dashboard to give a visual representation of the data. The database ("air_sensor_iot") is simply structured with different tables explained below:

- node_data
  Describing this table outlines the fields such as temperature, humidity, CO2 and the node_id the data is recorded from.

- actuator_fan
  Describes the status of the fan with the current timestamp as well.

- actuator_window
  Describes the status of the window with the current timestamp an action was triggered.
  No dependency exist between the tables. It is also important to note that node_id implemented as a field within some of the tables are important because of multiple devices which will might be recording similar field values within various times.

  The various tables as a diagram has been represented in the figure below.

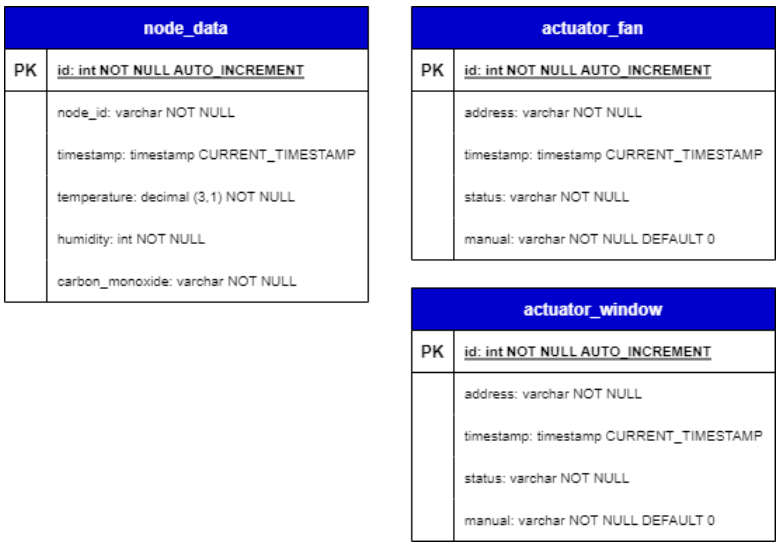

Figure 5: Database Schema for Air Sensor Detector.

3.4 Data Encoding

As a way to include a data-encoding mechanism, JSON data structure was used to transmit all data to the collector. The choice of this mechanism is due to the fact that the sensors have limited resources and XML is a complex structure for this project scope. JSON is less verbose and more flexible.

4 Deployment and Execution

4.1 Collector

The collector represents the main part of the application. The collector basically collects data from the sensors or receives information from actuators. Moreover, a simple CLI that permits users have control the application. The main intuition behind this is to enable parameters to be set to simulate different use-cases within different homes. After the system parameters are set, the application works autonomously. The collector is fully implemented in the language of Python and the modules of CoAPThon3 and Paho library was used. Threads are used to handle and manage the CoAP and MQTT network within the main file.

4.2 Command Line Interface Commands

The following are the commands implemented in the collector

- help: Shows all possible commands that can be used within the application.
- activate Command to communicate to sensors to start.
- filter**
- logs Command to look for the logs of the sensors.
- simulate command to show the mode of operation of the Air Sensor detector.
- change params command to change the values of humidity, temperature, CO2.
- exit command to close the application.

4.3 CoAP Network Implementation

The server is started on the main python file by the thread that handles the CoAP network. After the initialization of the CoAP server, the application registers itself to the resources that manage the actuators.

- Class ResExample
  This python class implements a function to get the addresses of the actuators and to observe the resources they expose.
- Class sendPost
  This class manages the post method to communicate directly with the resources exposed by the actuators.
- Class sensor
  This class manages the observation of resources. The constructor of the class is called on the render_GET(resExample class). Within the constructor of the application, a connection with the database as well as some fields are set to handle communication with the sensors. The observer functions then implements all the

logic that are behind the observation of all the resources exposed. If a message is triggered by a sensor, for instance, start fan, the application communicates with the MQTT network to notify the event.

4.4 MQTT Network implementation

The client is started on the main file by the thread that handles the MQTT network for the node_data

- FILE filter_extractor.py
  This class of methods are used to manage the sensors that are engaged in fan extraction process. Basically, the constructor is called on the main file when the thread starts. After the constructor is created, then a connection between the broker and the database is established. The method mqtt_client contains an mqtt.Client() which initiates the application on client side and then makes a reference to the on_connect and on_message methods defined within the same class.

  on_connect: subscribes a client to the topic of interest.

  on_message: handles messages that arrive for the client. The methods retrieves the fields of the JSON data structure and saves it in a table within the database. In addition, there is a checkActuator control check to see if the fan extractor needs to be charged.

  checkExtractorMode: This method checks if the level is within a given threshold. If not, it triggers the charge for the fan extractor.

  onCharge: This method is used to charge the fan extractor.The status of the actuator is checked before initiating this command.

  offCharge: This method checks and stop the charge of the extractor.

  communicateToSensors: This method is used to send a post to the actuator to either on or off the charge.

  executeLastState: This method retrieves the last state of the correspondent actuator.

- FILE value_collector.py
  The Class in this file is used to manage the sensor connected to the fan(extractor). The constructor is called in the Main file when the thread is instantiated. Once the constructor is created, the connection with the database and the broker is established. There is also a call to the mqtt.Client() to initialize the client as well as an on_connect and on_message. There are some parameters that are passed to the constructor. Explained below are some list of methods that exists within the Class.

  on_connect: This method subscribes the client to the topic of interest.

  on_message: This method handles the messages arriving as data to the client. In addition, the JSON data structure received is written into the database.

  checkActuatorFan:This method checks if the last values reported by the user are within the accepted scope for the values.

startFan: This method is used to start the fan extraction process. There are control logic to check if the actuator can initiate the fan process.

stopFan: This method is used to stop the fan extraction process. There are control logic to check if the actuator can initiate the fan process.

shouldStartFan: This method checks if the assigned values are accepted and the actuator is to start.

checkActuatorWindows: This method checks if the most recent values reported are acceptables based on user defined values.

openWindow: The method that triggers the window to open. There are some loop logic to control actuator action.

closeWindow: The method that triggers the window to close. There are some loop logic to control actuator action.

communicateToSensors: This method is used to send a post to the actuator and change the status and to start or stop the charge.

executeCurrentState: This method is used to look for the most current state of the actuator.

- FILE addresses.py
  The Class for this file is used to store all the addresses of the nodes within the network. The addresses stored are used by the Collector to communicate with all the nodes.

- FILE sensor.py
  The class in this file waits for the connection of the COAP sensor and calls the observation for them. The main variables under this context are the **fans** and the **window** which are set to "1" once the connection is established.

- FILE sendPost.py
  The Class defined in here is used to communicate to the CoAP sensors when needed. The **getClient()** is used to retrieve the client that wants to communicate to the Controller.

- FILE globalStatus.py
  This class contains some list of functions used to get the current state of the application as well as current simulation.

5 Flow of Execution

The diagrams below outline the different cases for the actuators.

5.1 Window Flow

5.1.1 Open Window Flow

The concept of Telemetry is used especially for the temperature to measure both room and outside temperature. To trigger the window opening option, the application flow has been illustrated with this sequence diagram.

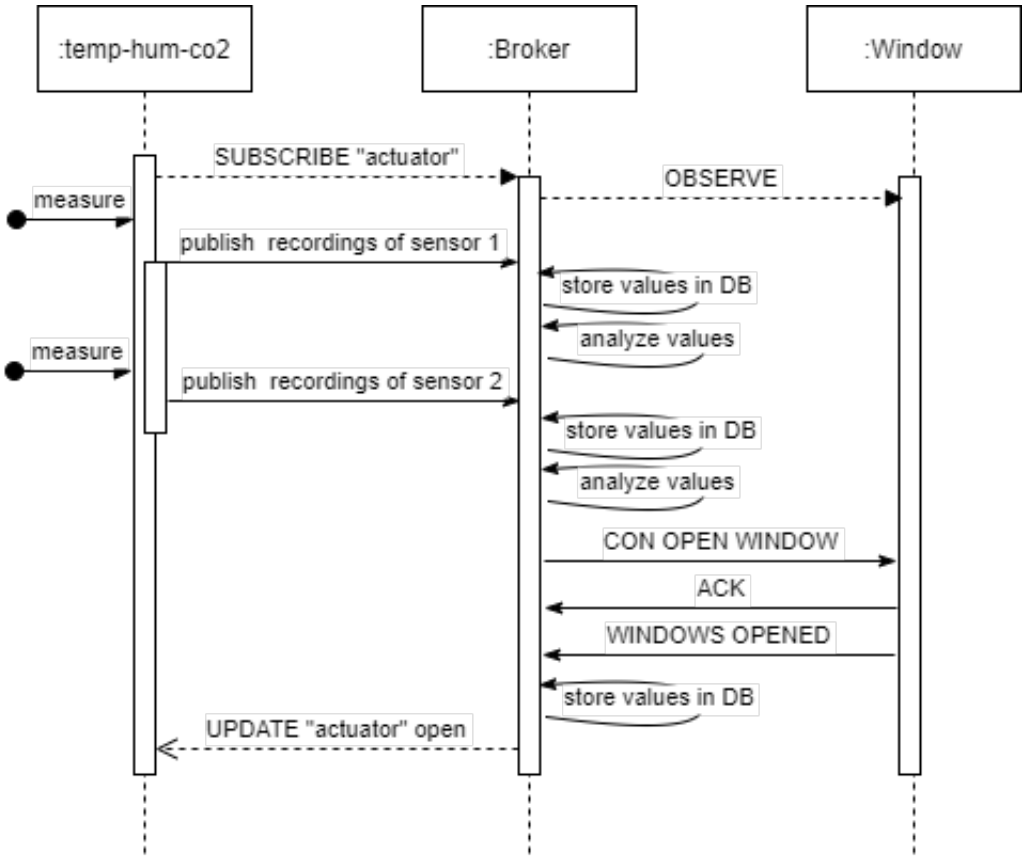

Figure 6: Sequence Diagram for Open Window Actuator.

5.2 Fan Flow

5.2.1 Open Fan Flow

This sensor handles two cases, one is to open the fan acting as an extractor for the room if infiltrated by any gases and other case to charge the fan.

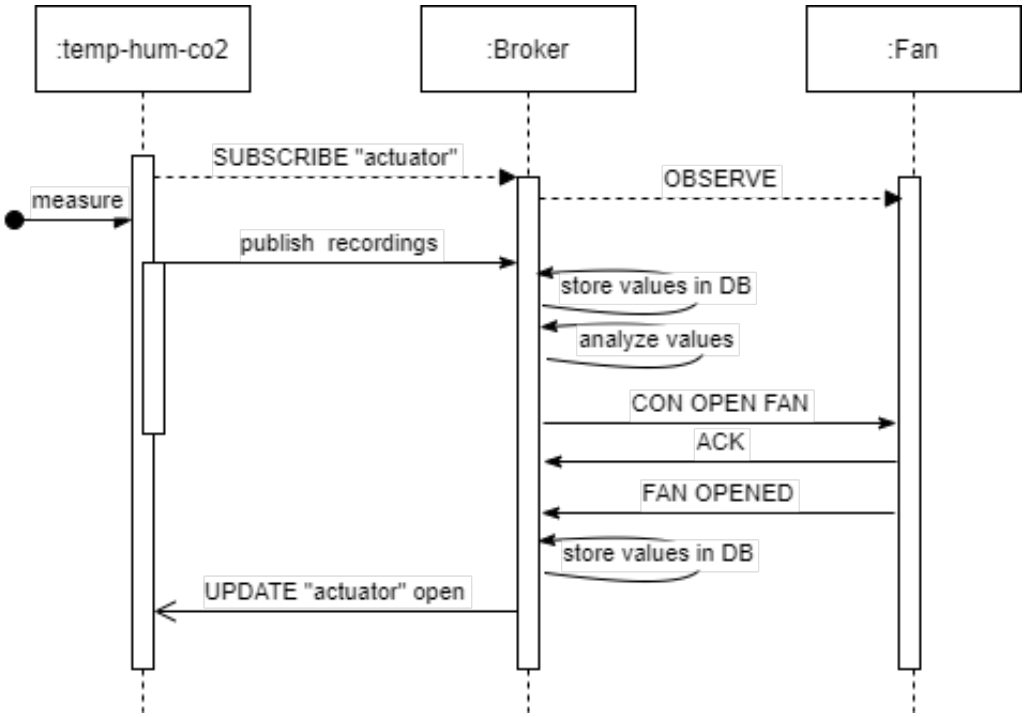

Figure 7: Sequence Diagram for Open Fan Actuator.

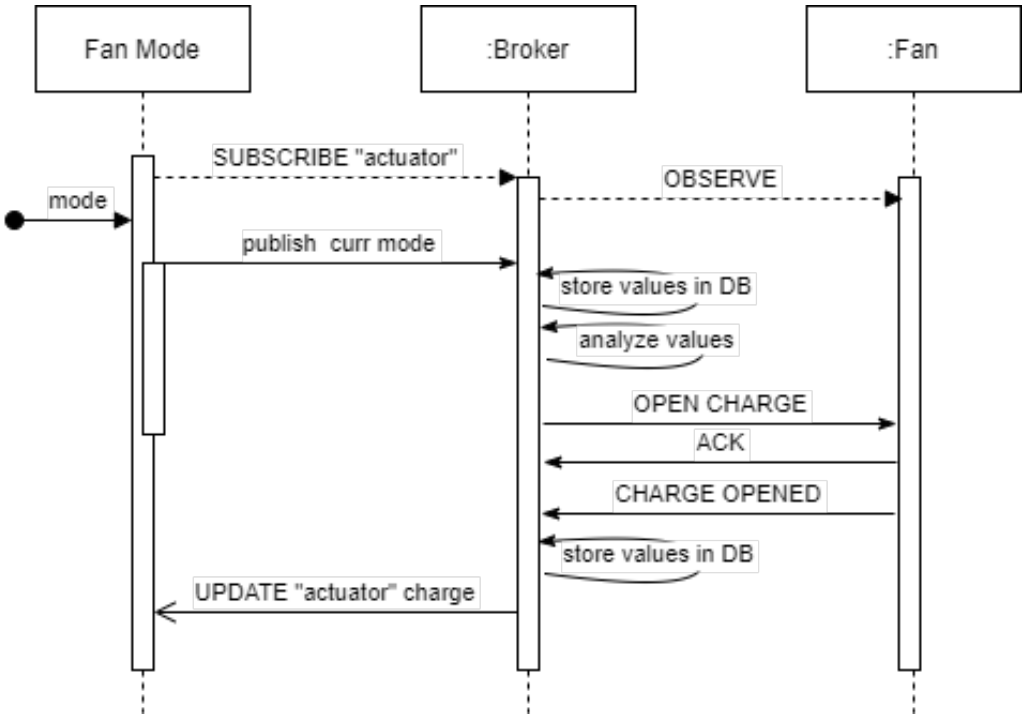

Figure 8: Sequence Diagram for Open Fan Mode.

6 Data Visualization-GRAFANA

Grafana was used to visualize the data stored within the MySQL database.