Towards IoT Sensor Service, IoT Actuator Service and Web Service modules for a middleware supporting end-user driven collaborative service composition.

By

Md. Raihan Uddin
(142-35-705)
&
Aziha Kamal
(142-35-693)

A thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Software Engineering

Department of Software Engineering
DAFFODIL INTERNATIONAL UNIVERSITY

Fall – 2018

APPROVAL

This **Thesis** titled on "**Towards IoT Sensor Service, IoT Actuator Service and Web Service modules for a middleware supporting end-user driven collaborative service composition.**", submitted by **Md. Raihan Uddin (142-35-705**) & **Aziha Kamal (142-35-693)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Prof. Dr. Touhid Bhuiyan — **Chairman**
Professor and Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Dr. Md. Asraf Ali — **Internal Examiner 1**
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Manan Binth Taj Noor — **Internal Examiner 2**
Lecturer
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Dr. Md. Nasim Akhtar — **External Examiner**
Professor
Department of Computer Science and Engineering
Faculty of Electrical and Electronic Engineering
Dhaka University of Engineering & Technology, Gazipur

DECLARATION

It hereby declere that this thesis has been done by **us** under the supervission of **K. M. Imtiaz-Ud-Din, Assistant Professor,** Department of Software Engineering, Daffodil International University. It also declere that nithor this thesis nor any part of this has been submitted elesewhere for award of any degree.

Md. Raihan Uddin
ID: 142-35-705
Batch: 14th
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University

Aziha Kamal
ID: 142-35-693
Batch: 14th
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University

Certified by:

K. M. Imtiaz-Ud-Din
Assistant Professor
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University

ACKNOWLEDGEMENT

This document presents our Bachelor of Science Thesis “Towards actuator and sensor service modules for a middleware supporting end-user driven collaboration service composition”. Appreciatively, we received advice from various people to whom we want to explicit our acknowledgment towards in this section.

Our sincerest thanks to our supervisor and mentor K.M. Imtiaz-Ud-din, Assistant Professor, Department of Software Engineering, Daffodil International University for providing us to a chance to do this. Without his encouragement, guidance and motivation we would not be able to realize the structure and got an output.

We also thankful to our mentors Khandker M. Qaiduzzaman, Kaushik Sarker, Md. Anwar Hossen, and Shohel Arman for all the support to execute this milestone.

Grateful to the Ambient Intelligence Lab group members especially Shuvo Proshad Sarnokar, Md. Sayeed Bin Muzahid, Antara Saha who always work and help with us in this research project.

We also thankful to Professor Dr. Touhid Bhuiyan, Professor & Head, Department of Software Engineering, Daffodil International University, and all the teacher in our department.

We also thank to researchers for their works which help us to learn design and implement our research project.

Last but not least, we want to thank almighty Allah for giving us patience. We are also thankful to teachers, family-members, friends, seniors, juniors for providing their effective support and prayers.

TABLE OF CONTANTS

Contents

LIST OF FIGURES

ABBREVIATION & ACRONYMS

- **IoT-** Internet of Things
- **IDE-** integrated development environment
- **MQTT-** MQ Telemetry Transport
- **HTTP-** Hypertext Transfer Protocol
- **API-** Application Programming Interface
- **UI-** User Interface
- **MCU-** Micro Controller Unit
- **HTML-** Hyper Text Markup Language
- **CSS-** Cascading Style Sheets
- **JS-** JavaScript
- **SQL-** Structured Query Language
- **ACID**- Atomicity, Consistency, Isolation, Durability

ABSTRACT

Modern software systems are increasingly characterized by uncertainties in the running context and user needs. These uncertainties and needs are difficult and challenging to predict at design time. While existing end-user driven service composition tools provide support for building composite IOT based services in order to meet user needs at run time, an environment that will enable end-users to create a collaboration of both web services and IOT based services in order to meet users requirement at while is yet to exist to the best of our knowledge. We therefore propose a novel architecture to address this gap. More specifically our contribution lies in designing and implementing a pluggable IOT based sensor and actuator environment that can be used to support composition of end-user driven collaboration of both IOT based services and other web services. We have also provided the validation of our designed architecture using a proof of concept scenario.

CHAPTER 1

INTRODUCTION

Currently people are surrounded by technology and this technology tries to develop our quality of life and also tries to facilitate our daily activities more and more. However, there are situations where technology is impossible to handle or people have lack of knowledge how to use it. Recent years IOT based services are increased and the integration of IOT services through web are gradually started according to Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., & Tamburrelli, G. (2011). Also many types of web services are rapidly developed to make our life easy. After collaborating both IOT based and web services, creating a complex ecosystem and for this ecosystem, creating uncertainties in the running context and user needs at run time. For this reason, we are trying to adapt the technology to the people's need by proposing a self-adaptive system where end-users have no need to programming knowledge for creating a service. An intelligence user interface where end-users give their requirements and dynamically a service can be discovered by adding service based adaptation. Users are able to create a composite service by using multiple actions.

Our proposed system architecture follows the distributed service and agent architecture and also service oriented architecture. Our proposed system has three modules Sensors, Actuators and Middleware where every module is independent. That's why user uses a module for multiple purposes. For example, collecting data from a temperature sensor in a certain area, we can automatically on and off air conditioner, measuring temperature, humidity and also collecting few months temperature and analysis that data set, we can assume a fixed day temperature. Another interesting part of our research is, our system is pluggable that means we will add any sensor and service easily just consider some installation process.

1.1 Motivating Scenario

With the advance of technology, it has become an essential and harmonizes part of human's life. Modern technology consists of many hardware and software components together to make our life easier. But still End-users who have no knowledge about programming language, they are not able to customize a software or service as their requirements and needs. For changing any action, service or any other parts of the application, need to hire related expert person. Let's see some real scenarios.

Scene 1: Mr. Rahim bought an automated water pump for his house building to climb up water in his house tank. For controlling this water pump, Mr. Rahim gets an application that has a user interface. A sensor service is integrated with the water pump and using this sensor, Mr. Rahim can know the water layer in the tank using the user interface and also on and off water pump through user interface. Mr. Rahim can customize the service according to his needs.

1.2. Objective

Our aspiration is to build a system where creating a collaboration of both IoT based service and web service to meet the user needs at run time and also sensor and actuator modules are pluggable, just follow some instruction such as a new software installation process type. This application is planned specially for the end-user who has no programming knowledge. They can easily customize the system as for the requirements they needs. End-users can make a composite service using multiple services without any programming knowledge. They can easily add a sensor or actuator easily to follow some process.

Our main goal is:

-Creating composite service using Iot based service and web service.

-Pluggable service.

1.3. Outline of the Thesis

In chapter 2 deigned with literature review and related technology. Here briefly describes the related research work in this area and which technologies we use are also described.

In chapter 3 describes the proposed system architecture and how it works.

In chapter 4 cover the implementation of the research project following the proposed architecture.

In chapter 5 describes proof of concept with a scenario and show an implementation with practical works with picture.

In chapter 6 concludes the thesis work by specifying contribution, limitation and future work.

CHAPTER 2

LITERATURE REVIEW

We survey the state of the art in the research areas addressed in this paper. To the best of our observation, a few works have been done in this area. These are adaptable user interface, multi agents system, peer to peer applications, distributed agent architecture, service oriented architecture, run time adaptation, sensor networks, ubiquitous computing, run time self-adaptation.

2.1 Related Works

A few works has done in composite service collaboration. One of the most noted works in this era is personal application in the IoT through visual end-user programming by Valsamakis, Y., & Savidis, A. (2017), permit end-users with an interface to easily and instantly modify IoT based services as their needs.

Kovatsch, M., Mayer, S., &Ostermaier, B. (2012), developed a system where logic and user interface are loosely coupled. Here logic means IoT based services are stored in server and if any functionality changes in the service, interface auto updated.

Noura, M., Heil, S., &Gaedke, M (2018), presents an architecture where some functionality is designed for end-user. The architecture works with existing IoT services which are managed by end-user. Here end-user first set their goal and it makes a service via translating that stored in a server, communicate through HTTP.

A group has worked at the area of integrated framework for adaptation of context-aware applications that empowers end-users to meet their own collections of services at run time and to fulfill specific goals according to Poladian, V., Sousa, J., Garlan, D., & Shaw, M. (2004). Another paper presents distributed agent architecture for end user and only one user can use it at a time according to Hassan, A., & Reza, F. (2018).

From above, what are missing, an architecture that enables end-user state the purpose and corresponding actions by creating composite service using both IoT based services and web services at their needs without any programming knowledge. Our next chapter converge our planning architecture.

2.2. Related Technology and framework

Here we cover which framework and technology we use to build our research project.

2.2.1. Service Oriented Architecture

An approach for distributed system architecture that implies loosely coupled services. It includes a collection of services in a network which communicate with each other. Each service is well defined, self-contained that provide separate functionality. It replaces new business requirement with plugging new services or upgrading existing services. This is a platform that is language independent. Figure 2.1 shows the service oriented architecture.

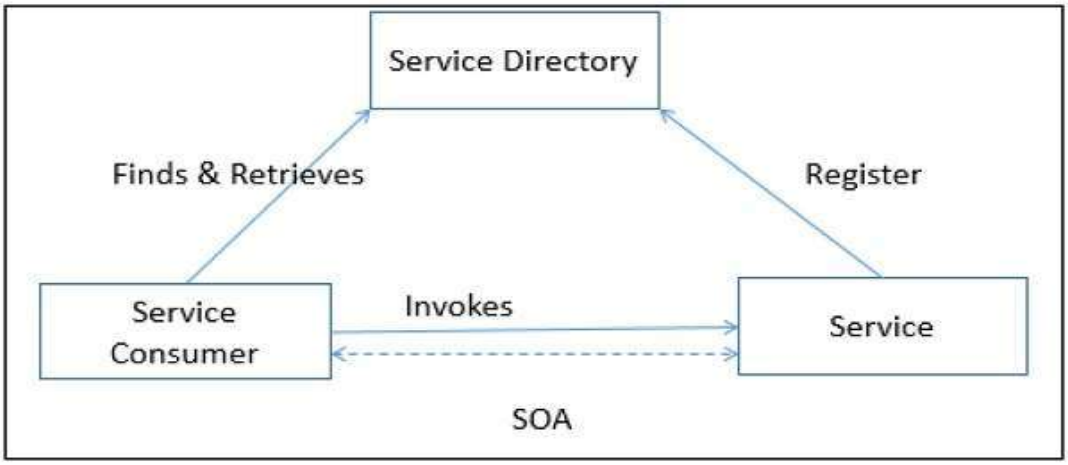

Figure 2.1: Service oriented architecture

2.2.2. Distributed Agent Architecture

All modules in distributed architecture such as sensors, actuators, middleware are independent and isolated. In our proposed system, we follow this architecture. Here one module can be used in various purposes. If one module can damage anyhow, other modules have no problem.

2.2.3. CloudMQTT Protocol Server

CloudMQTT protocol server is a MQTT broker. MQTT protocol connects with gadgets, sensors and subscribes and publishes packets. CloudMQTT server is a medium where, we store our packets or data through MQTT protocol. We are using this service to implement our research project.

2.2.4. Internet Protocol

Internet protocol is the principal communication protocol which data is sent from one device to another device over the internet according to Valsamakis, Y., & Savidis, A. (2017). Each packet that travels through internet is treated as an independent unit of records besides any relation to any other unit of records. Two types of internet protocol are available. The most uses internet protocol is IPV4 another is IPV6. IPV4 protocol provides 4.3 billion addresses and another one IPV6 provides 85,000 trillion addresses. IPV6 is the 21st century protocol.

2.2.5. Wireless Fidelity (Wi-Fi)

Wi-Fi is a local area networking (LAN) technology blueprinted to supply in-building broadband coverage. This is based on IEEE 802.11 specification. There are several specifications in 802.11 family such as 802.11a, 802.11b, 802.11g. It allows

computers and other smart gadgets to communicate over a wireless signal. It provides high speed connection without any cables. Now a day, office, home, hotels, cafes, airports and also a city is covered by Wi-Fi.

CHAPTER 3

DESCRIPTION OF PROPOSED SYSTEM

3.1. Proposed System Architecture

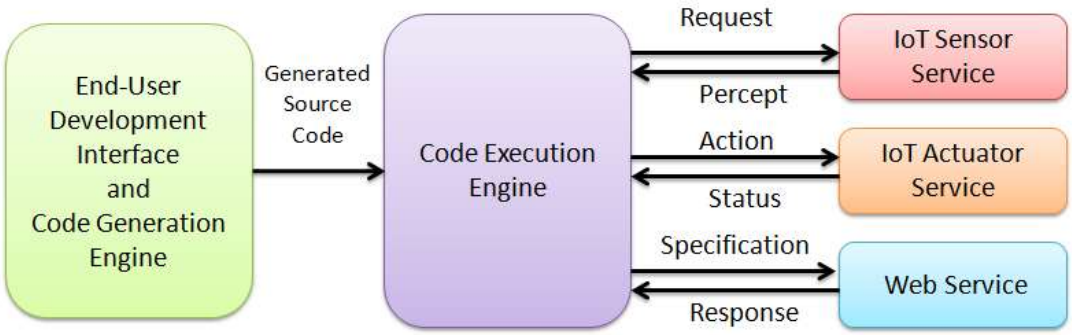

Figure 3.1: Proposed System Architecture

3.1.1 End-User Development Interface and Code Generation Engine

End-user development interface refers a graphical interface for general user so that they can easily use complex service without knowing programming. They can customize their service through their own way when needs without helping of a tech person. Some approaches are developed in this domain of end-user development. Most popular approaches are blockly, spreadsheet, scratch etc.

3.1.2. Code Execution Engine

This is a product layer arranged among applications and working frameworks. It is normally utilized in distributed frameworks where it disentangles programming improvement by doing the following:

- Hides the complexities of distributed applications
- Hides the heterogeneity of equipment, working frameworks and conventions

- Gives uniform and abnormal state interfaces used to make interoperable, reusable, and convenient applications
- Gives an arrangement of regular administrations that limit duplication of efforts and upgrades coordinated effort between applications.

3.1.3. IOT Sensor Service

Sensors are advanced gadgets that are every now and again used to distinguish and react to electrical or optical signs. A Sensor changes over the physical parameter into a signal which can be estimated electrically.

There are many types of sensors which are commonly used in IOT, like-

- Temperature
- Blood pressure
- Humidity
- Speed
- Soil moisture

In our proposed system architecture, IoT sensor service and code engine execution modules are communicated with each other through request and percept.

3.1.4. IoT Actuator Service

An actuator is a gadget that moves or controls some component. An actuator transforms a control motion into mechanical action, for example, an electric engine. An actuator binds a control framework to its condition. Different types of actuators, like-

- Comb drive
- Hydraulic piston
- Electric motor
- Relay

In our proposed system architecture, IoT actuator service and code engine execution modules are communicated with each other through status and action.

3.1.5. Web Service

Web service implies a product framework or architecture that backings applications to speak with one another viably through the web. It very well may be utilized in any equipment or programming stages and furthermore can be utilized in any programming dialects independently. That is the things that make web benefit based applications loosely coupled.

It is used in service-based model architecture and it tends to be considered a first favorable position or purpose behind utilizing web services. Service-based model means web services happen at the web server and any applications can ask for information from it and can expend those for the different users. It enables us to convey among different applications.

Diverse applications are composed of various programming languages and that is the reason they can't speak with one another yet utilizing web services we can speak with various applications by utilizing open conventions. Figure 3.2 shows the working process of web service.

It is a stage and programming language free and it doesn't depend on conventions. We can actualize the web services utilizing minimal effort correspondence frameworks.

Most often-used types of web service:

- SOAP
- XML-RPC
- JSON-RPC
- REST

Simply we can say that a system consumes service from another software system. There is an entity called request data such as weather conditions from server or service provider respond with the requested data.

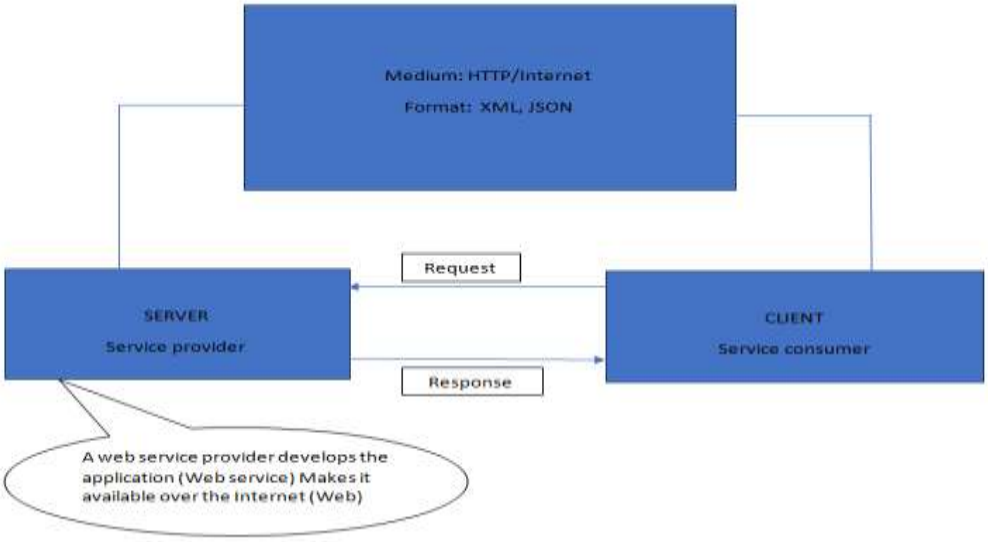

Figure 3.2: Web Service working process

CHAPTER 4

IMPLEMENTATION

4.1. IoT sensor service, IoT actuator Service and Web Service Working Module

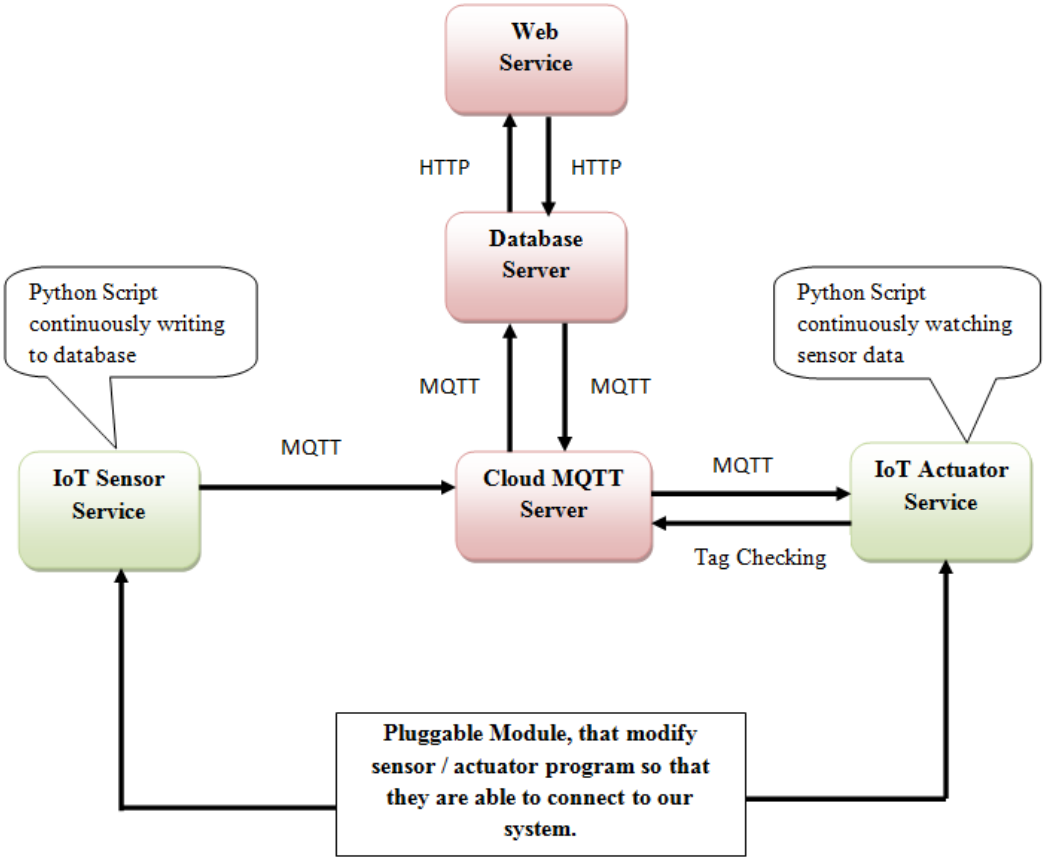

Figure 4.1: IoT Sensor Service, IoT Actuator Service and Web Service Working Module

4.1.1. Pluggable Module

Every sensor and actuator follows some basic rules to collect data and execute an action. In our pluggable platform, any sensor or actuator program can be modified for connecting our existing system. In figure 4.2 shows that how a sensor program is

looks like. It follows the basic structure for retrieve sensor data. It has three part and they are variable and pin declaration part, setup part and loop part.

```
const String MAC = "/S07USWTC23";
int sensorValue = A0;
int value;
void setup() {
  // put your setup code here, to run once:
    Serial.begin(115200);

}

void loop() {
  // put your main code here, to run repeatedly:
     value = analogRead(sensorValue);
     Serial.println(value);

}
```

Figure 4.2: Sensor connecting program

In our pluggable platform we identified this basic structure and based on this we inject the connection programming code to the file and create a new file which contains every necessary connection related code for connecting our existing system. In figure 4.3 shows that how we identified the basic structure of a sensor code.

```
for($x = 0; $x < sizeof($arr); $x++){
    // echo "x\n".$a."\n </br> x";
    if(strpos($arr[$x],"setup")){
        for($i = $x+1 ; $i < sizeof($arr); $i++){
            if($arr[$i]=="}"){
                break;
            }
            $setup = $setup . $arr[$i];

        }
        break;
    }
}
```

Figure 4.3: Basic setup part identified code

Identifying the basic structure the pluggable codes inject the necessary connection related code to the existing file and create a new file. In the new file all the necessary code are written and then the file is ready to communicate with the system. Figure 4.4 show that the connection related code of the pluggable system.

```
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.println("Connecting to WiFi..");
}
Serial.println("Connected to the WiFi network");

client.setServer(mqttServer, mqttPort);
client.setCallback(callback);

while (!client.connected()) {
  Serial.println("Connecting to MQTT...");

  if (client.connect("ESP8266Client", mqttUser, mqttPassword )) {

    Serial.println("connected");

  } else {

    Serial.print("failed with state ");
    Serial.print(client.state());
    delay(2000);

  }
```

Figure 4.4: Connection Related Code

4.1.2. IoT Sensor Service

In IoT sensor service we represent a sensor module which contains any type sensor and NodeMCU. In this we use a python script which continuously writing to the database through CloudMQTT. Figure 4.5 shows the python script that continuously writing to the database.

```
def on_message(self, client, userdata, msg):
    print(msg.topic + " " + str(msg.payload))

    messg = str(msg.payload).replace("'","")
    sensorValue = str(messg).replace("b","")
    print("sensor value",sensorValue)
    self.s.insertSensorTable(sensorValue,msg.topic)


def mqttSubscribe(self):
    self.client.on_connect = self.on_connect
    self.client.subscribe("/#")
    self.client.on_message = self.on_message
    try:
        self.client.loop_forever()
        print("Okay")
    except KeyboardInterrupt:
        print("Closing...")
```

Figure 4.5: Python Script subscribe data through MQTT

4.1.3. IoT Sensor Service to CloudMQTT Server

IoT sensor service connects with CloudMQTT through MQTT protocol. Figure 4.6 shows that, how data send from sensor module to CloudMQTT server for using the sensor data for further working process.

```
void callback(char* topic, byte* payload, unsigned int length) {

  Serial.print("Message arrived in topic: ");
  Serial.println(topic);

  Serial.print("Message:");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }

  Serial.println();
  Serial.println("-----------------------");

}

void loop() {
   String payload = "";
   payload += value;
   payload.toCharArray(topicValue,10);
   MAC.toCharArray(mac,15);

   if (!client.connected()) {
  reconnect();
 }
  client.loop();
  client.publish(mac, topicValue);
delay(2000);
}
```

Figure 4.6: IoT Sensor Service to CloudMQTT Server

4.1.4. CloudMQTT Server

We use a CloudMQTT server for receive data from sensor and send data to actuator, also for communicated with the database. Figure 4.7 shows the MQTT connection code. For connecting to CloudMQTT server, user needs server name, port number, username and password.

```
def __init__(self, server="m10.cloudmqtt.com", port=13964,
             username="hvgihpll", password="CTMRYdz5vSx4"):
    self.server = server
    self.port = port
    self.username = username
    self.password = password
    self.client = mqtt.Client()

def mqttConnection(self):
    try:
        self.client.username_pw_set(self.username, self.password)
        self.client.connect(self.server, self.port)
        # print("Connection with MQTT")
    except self.client.Error as e:
        print(e)
        print("Could not connect to the MQTT broker....")
        print("Closing....")
        sys.exit()

def on_connect(self, client, userdata, flags, rc):
    print("Connedted -Result code: " + str(rc))
```

Figure 4.7: CloudMQTT Server Connection Code

4.1.5. CloudMQTT Server to Database

CloudMQTTserver passes the collected sensor data to database. The database sensor tablecontains four columns and they are sensor, data, sensor value and time. There is also an ID column which is auto generated. Figure 4.8 shows that how sensor data pass to the database.

```
def insertSensorTable(self, value,topic):
    sql = "INSERT INTO `Sensor`.`SensorValue`\
    (`sensor`,`data`,`SensorValue`.`time`) \
    VALUES(%s,%s,CURRENT_TIMESTAMP)"

    args = (topic,value)
    self.db.dbConnection()
    self.db.executeData(sql, args)
```

Figure 4.8: CloudMQTT Server to Database

4.1.6. Database to Web Service

Sensor data sent from database to web service through HTTP protocol. Figure 4.9 shows that how sensor data sent from database to web service.

```
$app->get('/api/lowerlastvalue', function (Request $request, Response $response, array $args) {
   $query="SELECT * FROM sensorvalue WHERE sensor='/S07LSWTC23' ORDER BY time DESC limit 1";
    try{
        //get db object
        $db=new db();
        //connect
        $db=$db->connect();
        $stmt=$db->query($query);
        $sensorvalue=$stmt->fetchALL(PDO::FETCH_OBJ);
        $db=null;
        echo json_encode($sensorvalue);
    }
    catch(PDOException $e){
      echo '{"error": {"text": '.$e->getMessage().'}';
    }
});
```

Figure 4.9: Code of Database to Web Service

4.1.7. Web Service to Database

Using the sensor value user take an action, this action again carries by HTTP protocol from web service to database. Figure 4.10 shows that how action sent from web service to database.

```php
$app->put('/api/action/update/{id}', function(Request $request, Response $response){
    $id = $request->getAttribute('id');

    $action = $request->getParam('action');

 $query = "UPDATE actuator SET
                action  = :action
                WHERE id = $id";


    try{
        // Get DB Object
        $db = new db();
        // Connect
        $db = $db->connect();

        $stmt = $db->prepare($query);

       $stmt->bindParam(':action',$action);

        $stmt->execute();

        echo '{"notice": {"text": "Temperature Updated"}}';
    }

    catch(PDOException $e){
        echo '{"error": {"text": '.$e->getMessage().'}';
    }
});
```

Figure 4.10: Code of Web Service to Database

4.1.8. Database to CloudMQTT Server

The action then carried from database to CloudMQTT through MQTT protocol. Figure 4.11 shows the retrieval process of action value from database.

```python
    def rSensor(self):      # retrieve Sensor table data
        sql = "SELECT `actuator`.`atag` ,\
        `actuator`.`action` \
        FROM `restapps`.`actuator` \
        ORDER BY `actuator`.`time` DESC LIMIT 1;"
        args = ()
        self.db.dbConnection()
        self.db.executeData(sql, args)
        return (self.db.getCursor())
```

Figure 4.11: Action Retrieval Code from Database

4.1.9. CloudMQTT to IoT Actuator Service

Action value passes from database to IoT actuator service through MQTT protocol. Figure 4.12 shows that how the action value passes from database to IoT actuator through MQTT protocol.

```
    def mqttLoopForever(self):
        self.client.on_connect = self.on_connect
        data = self.s.rSensor()


        for d in data:
            topic = d[0]
            value = d[1]
            print("Actuator Tag: " + str(topic) + "    Action:" + str(v

            self.client.publish(str(topic), str(value))
            sleep(1.0)
            self.mqttLoopForever()

            try:
                self.client.loop_forever()
                print("Okay Man")
            except KeyboardInterrupt:
                print("Closing...")
```

Figure 4.12: Code of CloudMQTT to IoTActuator Service

4.1.10. IoT Actuator Service

In IoT actuator service, the entire time actuator module checks the action value for specific actuator through tag. If the IoT actuator service get a value for a specific actuator then it retrieve data from database and give an action. Figure 4.13 shows that how IoT actuator service connected with the database.

```
//*************************** Reconnect ******************************//
void reconnect(){
  while(!client.connected()){
    Serial.println("Connecting to MQTT");

    if (client.connect("ESP8266Client", mqttUser, mqttPassword)){
      Serial.println("Connected");
      client.subscribe("/ac0wtc23");
//       client.subscribe("/#");
    }else{
      Serial.print("Failed with State");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
//*************************** <Reconnect> ******************************//
```

Figure 4.13: Connection Code of IoT Actuator Service

CHAPTER 5

PROOF OF CONCEPT

Our main target is to make a pluggable platform where we can plug any sensor or actuator to receive or send data. To achieve our target we make a pluggable platform where any sensor or actuator device programming code can be modified for connecting our existing system.

For connecting in our existing system every sensor or actuator need some specific connection and output related information which is provided by our pluggable platform. So every sensor and actuator needs to be modified before to connect in our existing system.

In our existing system sensor data is readable and according to sensor data action can be performable by the actuator.

For proof of our concept, we use a scenario that is described in chapter 1 and section 1.1 in scene 1.

5.1. Experimental setup

5.1.1. Arduino IDE

The Arduino IDE is a cross-platform application which is written using Java programming language. It runs on Windows, MacOS and Linux. It is used to write and upload code to Arduino board. The Arduino IDE supports C and C++ programming languages using special rules of code structuring. Figure 5.1 shows the Arduino IDE interface. It supplies a software library from wiring project, which provides many common input and output procedures.

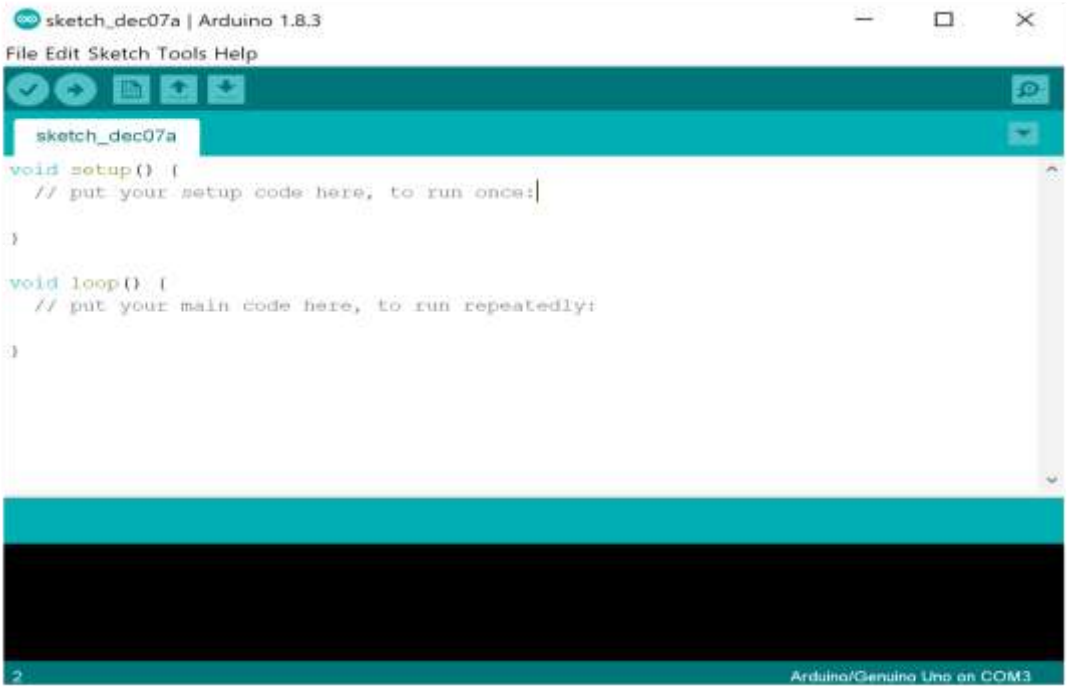

Figure 5.1: Arduino IDE interface

Client composed code just requires two fundamental capacities, for beginning the draw and the primary program loop. The Arduino IDE utilizes the program avrdude to change over the executable code into the content record in the hexadecimal encoding that is stacked into the Arduino board by a loader program in the board's firmware.

5.1.2. Apache HTTP Server

The Apache HTTP Server, conversationally called Apache is a free and open-source cross-stage web server which discharged under the terms of Apache License 2.0. Apache is created and kept up by an open network of engineers under the sponsorship of the Apache Software Foundation.

Apache bolsters an assortment of highlights many actualized as arranged modules which expand the center usefulness. These can extend from validation plans to supporting server-side programming dialects, for example, Perl, Python, Tcl, and PHP.

5.1.3. MySQL

MySQL is the popular open source relational database management system. It is a cost-effective delivery of dependable, superior and versatile Web-based and installed database applications. It is an incorporated exchange sheltered, an ACID-agreeable database with full confer, rollback, crash recuperation, and column level locking capacities.

The MySQL database provides the following features:

- Elite and Scalability to meet the requests of exponentially information burdens and clients.
- Self-recuperating Replication Clusters to enhance versatility, execution, and accessibility.
- Online Schema Change to meet changing business prerequisites.
- Execution Schema for observing client and application-level execution and asset utilization.
- SQL and NoSQL Access for performing complex questions and straightforward, quick Key-Value activities.
- Enormous Data Interoperability utilizing MySQL as the operational information store for Hadoop and Cassandra.

5.1.4. Brackets (Text Editor)

'Brackets' is a source code editorial manager with an essential spotlight on web development. Created by Adobe Systems, it is free and open-source programming authorized under the MIT License and is presently kept up on GitHub by Adobe and other publicly released designers. Figure 5.2 shows the Brackets interface. It is composed of JavaScript, HTML, and CSS. Brackets are cross-stage, accessible for MacOS, Windows, and most Linux distros. The principle motivation behind sections is its live HTML, CSS, and JS altering usefulness.

Figure 5.2: Brackets (Text Editor) Interface

5.1.5. Postman

Postman is a great HTTP customer for testing web services. Postman makes it easy to test, make and document APIs by empowering customers to quickly amass both essential and complex HTTP requests. Postman is available as both Google Chrome Packaged App and a Google Chrome in-program application. Figure 5.3 shows the Postman Interface.

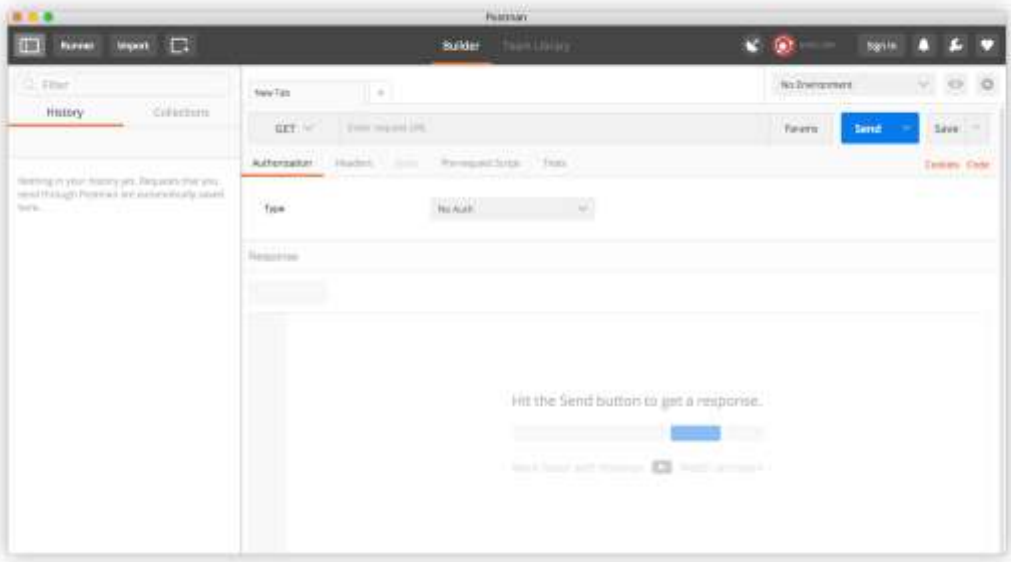

Figure 19: Postman Interface

Postman has a perfect and natural UI, with most key highlights available inside a single tick. The expectation to absorb information for utilizing the program is low, most clients ought to have the capacity to begin building and testing API calls rapidly. One central explanation behind Postman's convenience is its robotization capacities. It mechanizes the way toward making API request for and testing API reactions, enabling engineers to set up an extremely proficient work process.

5.2. Practical Experiment

We make a sensor module which is actually any kind of sensor and a NodeMCU module. For collecting sensor data we need this sensor module because we sending data through WiFi to our database. In the figure 5.4, we use a soil moisture sensor and NodeMCU module for our experiment. We are using two soil moisture sensors to read upper water level and lower water level in formation. After collecting this information we are sending the sensor data through Wifi using NodeMCU to our database.

Figure 5.4: Sensor and NodeMCU module for receiving data

In the figure 5.5, showing that the sensor module can read data and it can usable for further experiment.

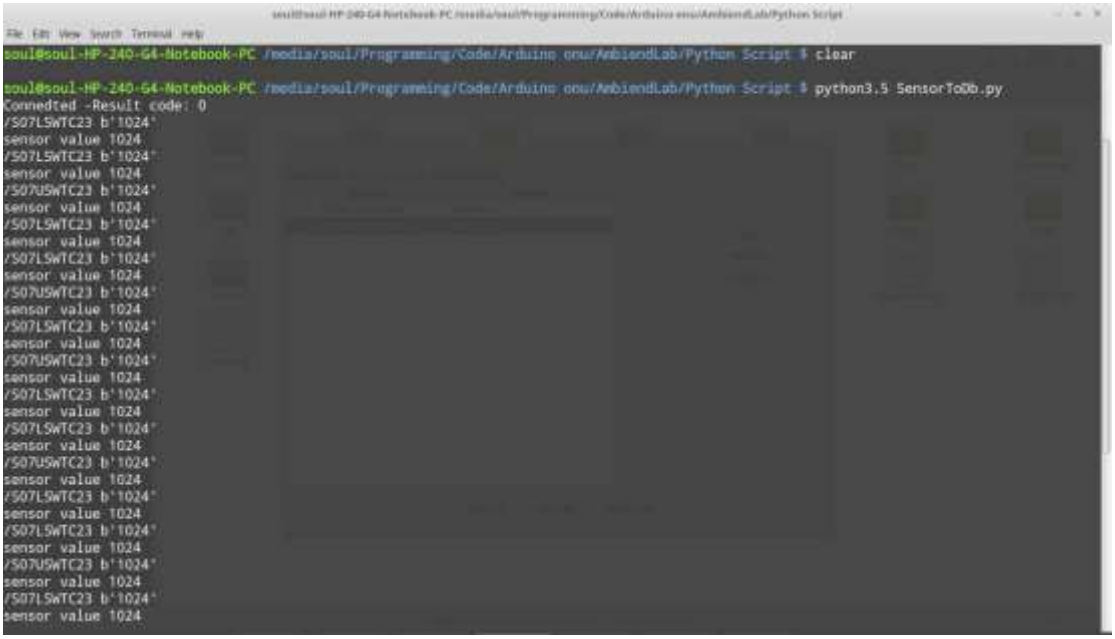

Figure 5.5: Sensor data in the local terminal

After that our next challenge is sending this data to anywhere through API. For this, we use the MQTT protocol for publishing our sensor data to the database.

In figure 5.6, the database is containing four fields (id, sensor, data, and time). The id is the auto-generate field which is created automatically if any sensor data send to the database. The sensor field contains the tag of the sensor. From this tag value, the sensor data can be identified which sensor data is it. The data field shows the value of the sensor which is collected by the sensor module and the time field is automatically changed with each sensor data updated.

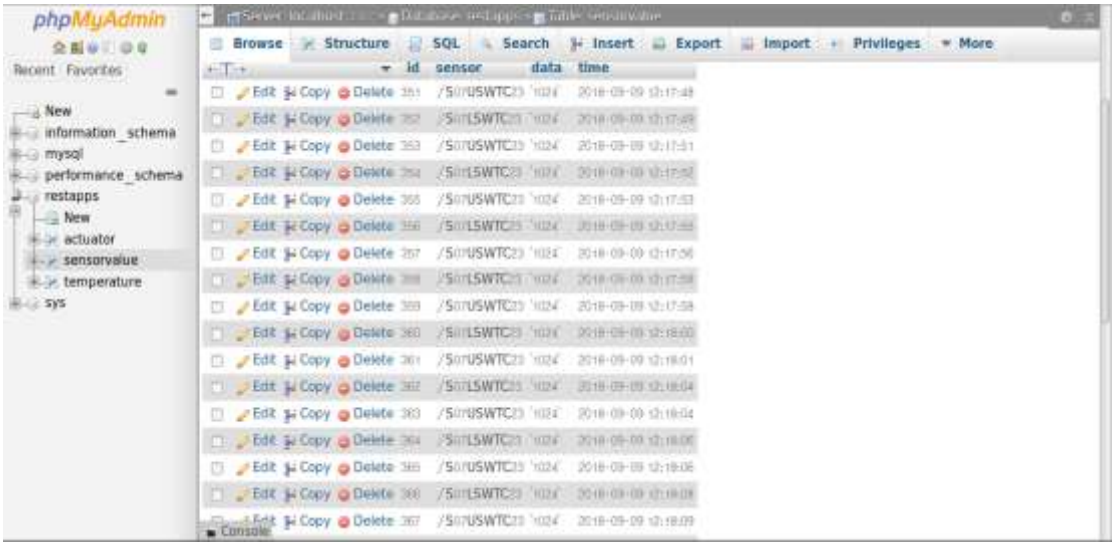

Figure 5.6: Publishing sensor data to the database through MQTT Protocol.

Our next challenge was sending the data from the database to middleware through API. We used the HTTP protocol for sending sensor data from database to middleware. Middleware is the place where a user can use the sensor data for further action. In figure 5.7, Through HTTP protocol we send our sensor data from the database to middleware.

Figure 5.7: Sending sensor data from the database to middleware

In the middleware, the user can perform an action based on sensor data. The user needs to drag and drop sensor value and the action which is performed by the actuator. In figure 5.8, shows how the user interface looks like.

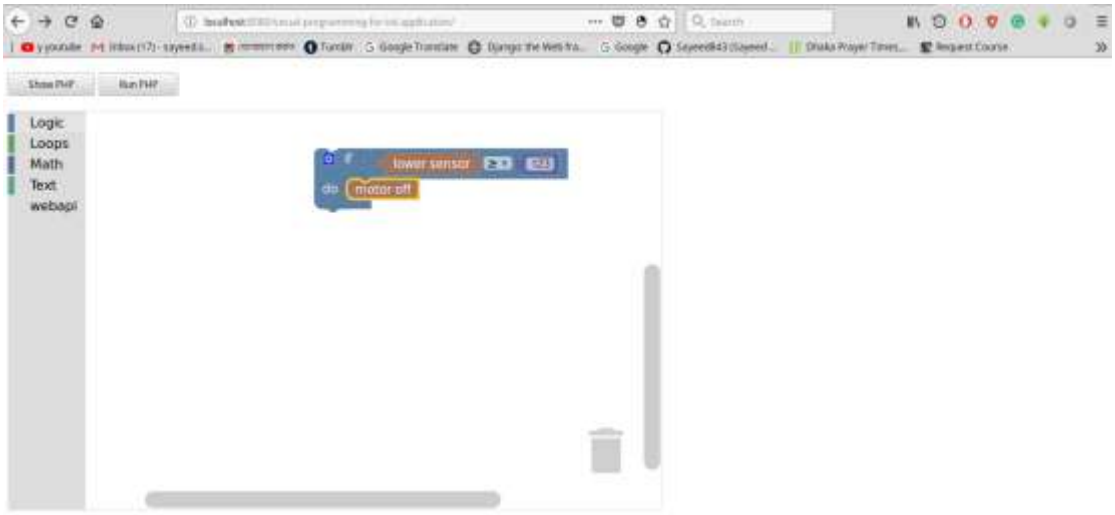

Figure 5.8: Drag and drop user interface

The user need not to know what is happening in the background. Based on the user need the code is auto-generated and a PHP script runs automatically in the background. The action is carried by an API to the database. For carrying this action we used HTTP protocol. In figure 5.9, the auto generated code is showed.

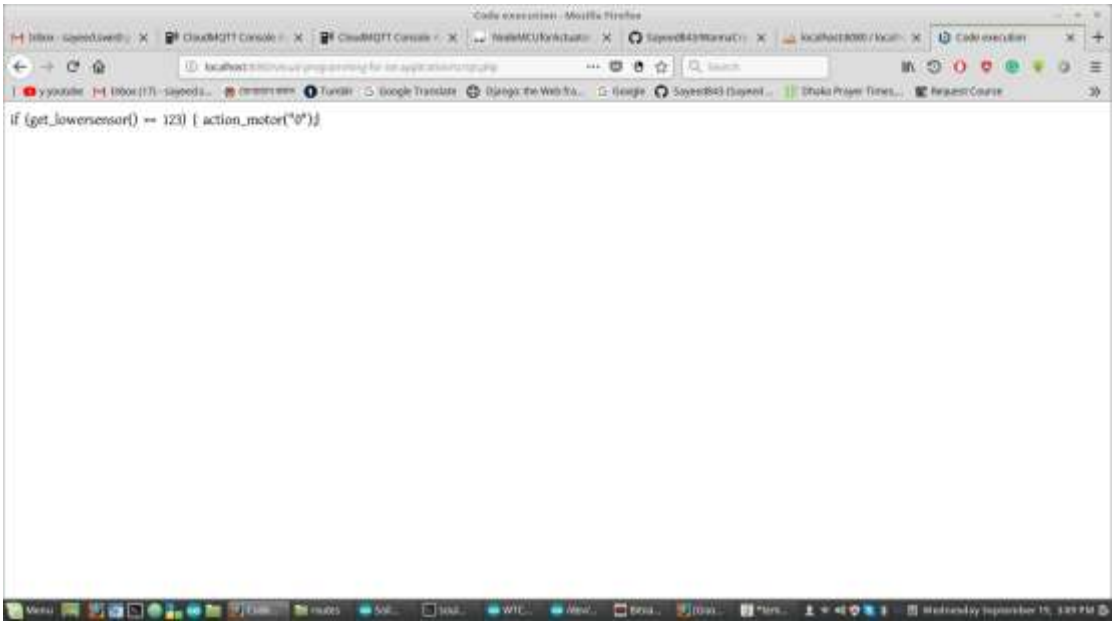

Figure 5.9: Auto-generated code in the background

The whole system is distributed and loosely coupled. The sensor module, middleware, and actuator module are independent because every part of our system is separate from each other. The database contains action value which will be performed by the actuator module. If the action value is False or Zero (0) the motor will be 'OFF' and

if the action value is True or One (1) then the motor will be ‘ON’. Figure 5.10 shows the actuator action value database table.

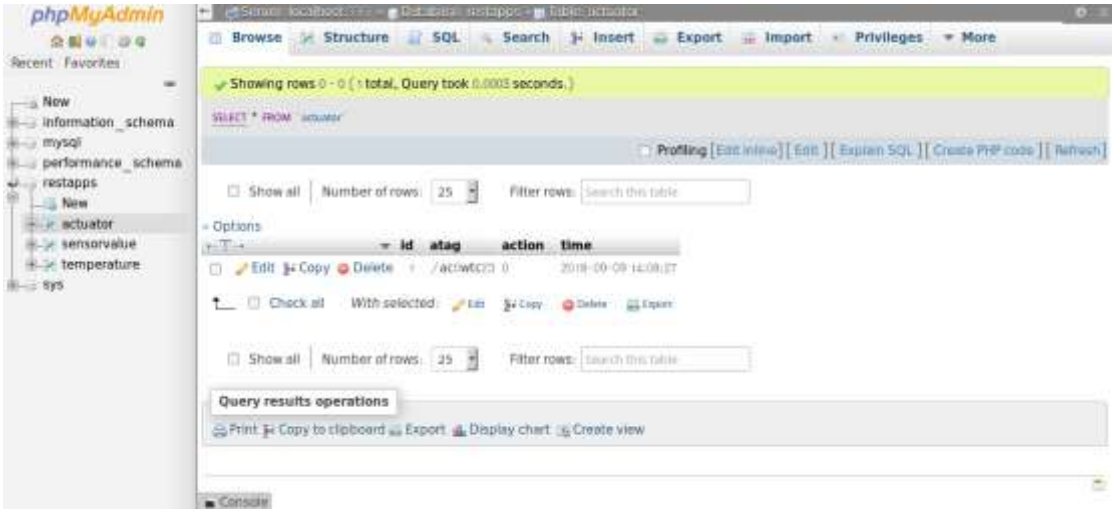

Figure 5.10: Action value carried by HTTP protocol from Middleware to Database

This value is carried from middleware to database through HTTP protocol. When database value updated for actuator module, the action is performed by the relay which can control the motor for switching it ‘ON’ or ‘OFF’. Figure 5.11 shows the ‘OFF’ situation of the actuator. When the database value of the action field will be False or Zero condition the actuator module will look like this. In the ‘OFF’ condition the relay will be ‘OFF’ the motor.

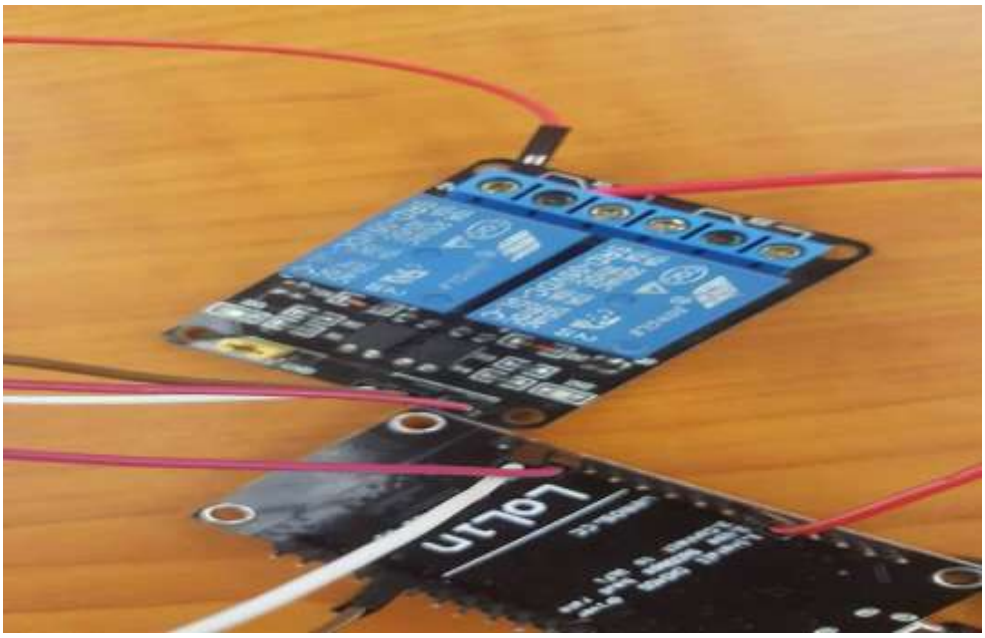

Figure 5.11: Actuator Module (Off Condition)

If the motor is in the ‘ON’ condition, the motor is used to show our experimental output of the actuator module. When database action field value will be True or One (1), the actuator module will be looked like this. In the ‘ON’ condition the relay will be ‘ON’ the motor which is used in our experiment for showing out actuator module output. Figure 5.12 shows the ‘ON’ condition of the actuator module.

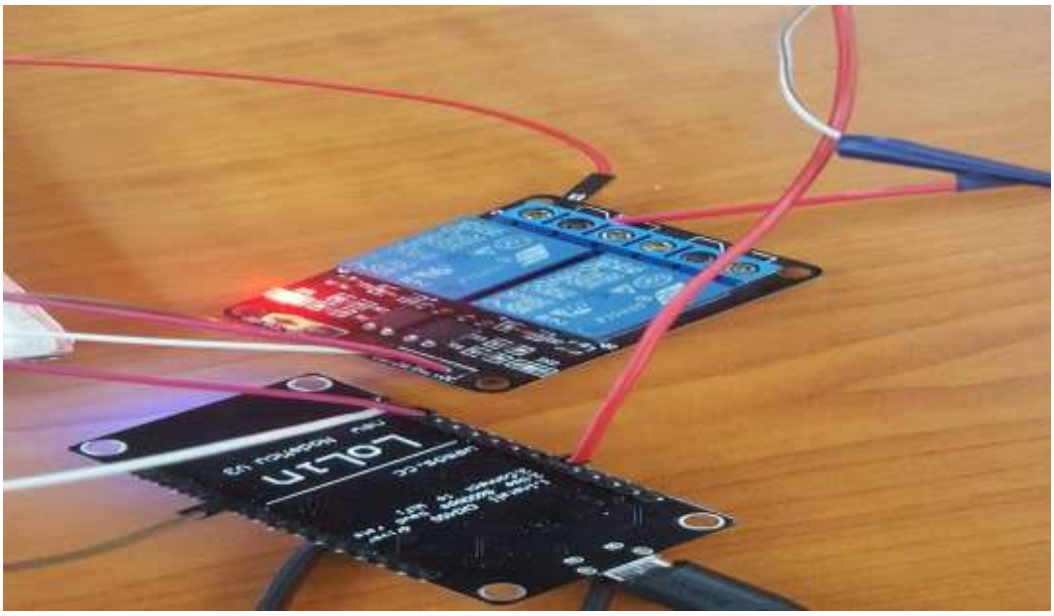

Figure 5.12: Actuator Module (On Condition)

The action values of the actuator module which is performed by the relay for ‘ON’ or ‘OFF’ the motor in our experiment to show the possible outcome. Figure 5.13 shows the local terminal which contains actuator action values.

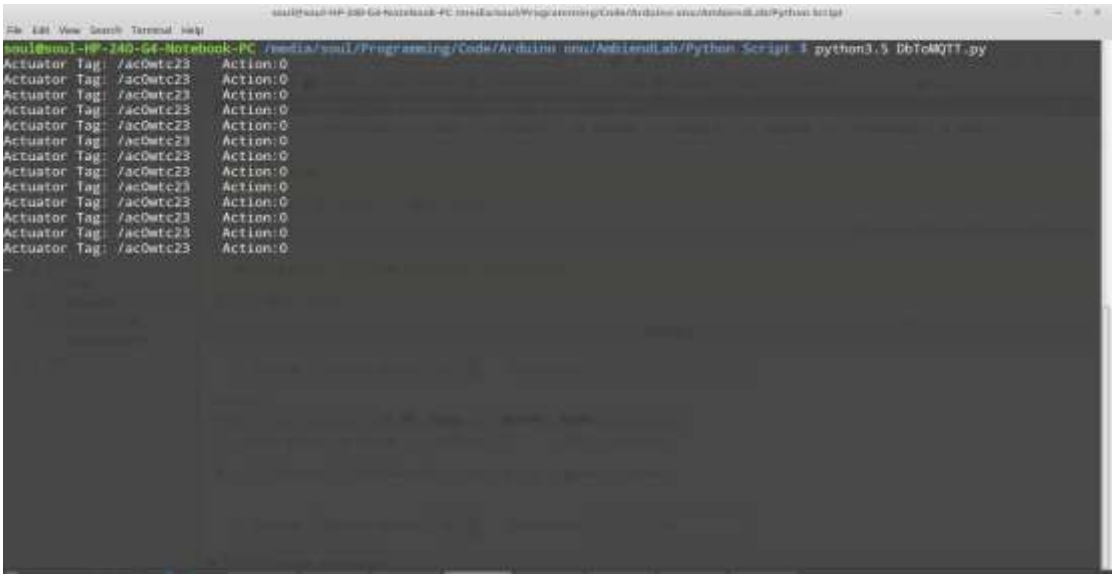

Figure 5.13: Actuator Data in the Local Terminal

5.3. Result Summery

Our main goal is to establish a system architecture which follows the distributed service and agent architecture and also service-oriented architecture. For this, first of all, we make a pluggable platform for all sensor and actuator module. This pluggable platform holds all information about connecting and output information with our system.

Secondly, after collecting sensor data, we send the sensor data to our database through MQTT protocol and the database is situated to another device. After that, we send sensor data from the database to the middleware through HTTP protocol. Middleware is the place where the user can drag and drop a condition and make a decision. After making a decision the code generated automatically in the background. The auto-generated code takes the action to the database through the HTTP protocol.

Finally, the actuator module gives an output based on the action value. As we use a relay to control our motor switch, so if we get an action value 'False' or '0', we turn off the motor and if we get an action value 'True' or '1', the motor switch on condition.

CHAPTER 6

CONCLUSIONS

5.1. Contribution

The main goal of our research project is helped the end-user who has no programming knowledge and all modules are independent. Programming language is independent and if damaged one module, other modules have no problem. In sensor module, if we want to add a new sensor, it's an easy process cause sensor module is pluggable and as it is independent, we use it in multiple application, multiple purposes. For the actuator module, it is also same to sensor module. Also end-user create composite service using IoT based services and web services according to their needs.

5.2. Future work

Our research project has some limitation as like we use Wi-Fi for data transfer from sensor to server that sometimes can interrupt. Sometimes garbage data is also stored and same data stored also from sensor. We are used a third party cloud MQTT protocol server as middleman to send and receive data for creating service, action and it is a worst work for us. As all modules are independent but we are worked in a local area network for checking purpose in deferent device. For data receiving from cloud MQTT server to sensor, we use a recursion function that only run 999 times and then stop. Last one is our data transferring and decision for action is quiet slow.

Present time our proposed architecture is working properly. In future, we will work for efficiency, data filtering. Remove the third party cloud MQTT protocol server and makes our own web socket for data transferring. We will also add machine learning for making more effect full decision. We will work for making a full automated distributed intelligence system for end-user for better performance and service.

Reference

Poladian, V., Sousa, J., Garlan, D., & Shaw, M. (n.d.) (2004). Dynamic configuration of resource-aware services. *Proceedings. 26th International Conference on Software Engineering*.

Salber, D., Dey, A. K., &Abowd, G. D. (1999). The context toolkit. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems the CHI Is the Limit - CHI 99*.

Kovatsch, M., Mayer, S., &Ostermaier, B. (2012). Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*.

Filieri, A., Ghezzi, C., &Tamburrelli, G. (2011). Run-time efficient probabilistic model checking. *Proceeding of the 33rd International Conference on Software Engineering - ICSE 11*.

Epifani, I., Ghezzi, C., Mirandola, R., &Tamburrelli, G. (2009). Model evolution by run-time parameter adaptation. *2009 IEEE 31st International Conference on Software Engineering*.

Valsamakis, Y., & Savidis, A. (2017). Personal Applications in the Internet of Things Through Visual End-User Programming. *Digital Marketplaces Unleashed,* 809-821.

What is Internet Protocol? - Definition from WhatIs.com. (n.d.). Retrieved from https://searchunifiedcommunications.techtarget.com/definition/Internet-Protocol

What is Internet Protocol? - Definition from WhatIs.com. (n.d.). Retrieved September 28, 2018.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., &Tamburrelli, G. (2011). Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*,*37*(3), 387-409.

Zhang, J., & Cheng, B. H. (2006). Model-based development of dynamically adaptive software. *Proceeding of the 28th International Conference on Software Engineering - ICSE 06*.

Hassan, A., & Reza, F. (2018). Designing simple reflex agent using distributed agent architecture over IOT, Unpublished Undergraduate Thesis, Daffodil International University, Bangladesh.

F., Junaid, M., & S. (2016). Distributed Agent Architecture Using Internet of Things.

Youngblood, G., & Cook, D. (2007). Data Mining for Hierarchical Model Creation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),37*(4), 561-572.

Doctor, F., Hagras, H., & Callaghan, V. (2005). A Fuzzy Embedded Agent-Based Approach for Realizing Ambient Intelligence in Intelligent Inhabited Environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,35*(1), 55-65.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., & Jansen, E. (2005). The Gator Tech Smart House: A programmable pervasive space. *Computer,38*(3), 50-60.

Surie, D., Laguionie, O., & Pederson, T. (2008). Wireless sensor networking of everyday objects in a smart home environment. *2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*.

Noura, M., Heil, S., &Gaedke, M. (2018). GrOWTH: Goal-Oriented End User Development for Web of Things Devices. *Lecture Notes in Computer Science Web Engineering.*

Appendix-A

For Sensor Module:

```
importpaho.mqtt.client as mqtt
import sys
importpymysql as pymysql
from time import sleep
classDBoperation():
def __init__(self, user='root',
password='12345678',
host='127.0.0.1',port=3307,
dbName='restapps'):
self.user=user
self.password=password
self.host=host
self.port=port
self.dbName=dbName
defdbConnection(self):
try:
self.db=pymysql.connect(user=self.use
r,
password=self.password,host=self.host
, port=self.port, db=self.dbName)s
exceptpymysql.Error as e:
print(e)
print("Could Not connect to the
database")
print("Close")
sys.exit()
self.cursor=self.db.cursor()
defexecuteData(self, sql, args):
try:
self.cursor.execute(sql, args)
self.db.commit()
exceptpymymql.Error as e:
self.db.rollback()
print(e)
print("Execute Data into
Database.....Failed")
defgetCursor(self):
cursorList = []
    # print(str(self.cursor.fetchall()))
for cur in self.cursor.fetchall():
cursorList.append(cur)
returncursorList
defreturnCursor(self):
cursorList = []
for cur in self.cursor.fetchall():
cursorList.append(cur)
returncursorList
```

```
def display(self):

for element in self.cursor:

print(element[0])

class Sensor():

db = DBoperation()

definsertSensorTable(self, value,topic):

sql = "INSERT INTO
`Sensor`.`SensorValue`\
(`sensor`,`data`,`SensorValue`.`time`) \

VALUES(%s,%s,CURRENT_TIMES
TAMP)"

args = (topic,value)

self.db.dbConnection()

self.db.executeData(sql, args)
```

Agent Function

```
defrSensor(self):      # retrieve Sensor
table data

sql = "SELECT `actuator`.`atag` ,\

      `actuator`.`action` \

      FROM `restapps`.`actuator` \

      ORDER BY `actuator`.`time`
DESC LIMIT 1;"

args = ()

self.db.dbConnection()

self.db.executeData(sql, args)

return (self.db.getCursor())
```

```
classMqttBroker():

    s = Sensor()

db = DBoperation()


def __init__(self,
server="m10.cloudmqtt.com",
port=17635,

username="ewpjantq",
password="Rfnyu4oiB8WO"):

self.server = server

self.port = port

self.username = username

self.password = password

self.client = mqtt.Client()

defmqttConnection(self):

try:

self.client.username_pw_set(self.usern
ame, self.password)

self.client.connect(self.server,
self.port)

        # print("Connection with
MQTT")

exceptself.client.Error as e:

print(e)

print("Could not connect to the MQTT
broker....")

print("Closing....")

sys.exit()
```

```
defon_connect(self, client, userdata, flags, rc):

print("Connedted -Result code: " + str(rc))

defon_message(self, client, userdata, msg):

print(msg.topic + " " + str(msg.payload))

defmqttLoopForever(self):

self.client.on_connect = self.on_connect

data = self.s.rSensor()

      # print(data)

for d in data:

          # print("Data: "+d)

          # decision = self.s.rSensor(str(d))

print("Actuator Tag: " + str(d[0]) + " Action:" + str(d[1]))

self.client.publish(str(d[0]), str(d[1]))

          # if (decision[0] == 1):

          #     print("Ami Aci" + str(decision))

          #     self.client.publish(str(d), str(decision[0]))

sleep(1.0)

self.mqttLoopForever()

try:

self.client.loop_forever()

print("Okay Man")

exceptKeyboardInterrupt:

print("Closing...")

broker = MqttBroker()

# database.dbConnection()

broker.mqttConnection()

broker.mqttLoopForever()
```

For Web service

```
<?php

use \Psr\Http\Message\ServerRequestInterface as Request;

use \Psr\Http\Message\ResponseInterface as Response;

$app = new \Slim\App(['settings' => ['displayErrorDetails' => true]]);

//Get all sensor data from database

$app->get('/api/sensorvalue', function (Request $request, Response $response, array $args) {

  $query="SELECT * FROM sensorvalue";

try{

      //get dbobjec

      $db=new db();

      //connect

      $db=$db->connect();
```

```
        $stmt=$db->query($query);

        $sensorvalue=$stmt->fetchALL(PDO::FETCH_OBJ);

        $db=null;

echojson_encode($sensorvalue);

    }

catch(PDOException $e){

echo '{"error": {"text": '.$e->getMessage().'}';

    }

});

//get last data from database for Lower sensor

$app->get('/api/lowerlastvalue', function (Request $request, Response $response, array $args) {

   $query="SELECT * FROM sensorvalue WHERE sensor='/S07LSWTC23' ORDER BY time DESC limit 1";

try{

        //get db object

        $db=new db();

        //connect

        $db=$db->connect();

        $stmt=$db->query($query);

        $sensorvalue=$stmt->fetchALL(PDO::FETCH_OBJ);

        $db=null;

echojson_encode($sensorvalue);

    }

catch(PDOException $e){

echo '{"error": {"text": '.$e->getMessage().'}';

    }

});

//get last data from database for Upper sensor

$app->get('/api/upperlastvalue', function (Request $request, Response $response, array $args) {

   $query="SELECT * FROM sensorvalue WHERE sensor='/S07USWTC23' ORDER BY time DESC limit 1";

try{

        //get db object

        $db=new db();

        //connect

        $db=$db->connect();

        $stmt=$db->query($query);

        $sensorvalue=$stmt->fetchALL(PDO::FETCH_OBJ);

        $db=null;

echojson_encode($sensorvalue);

    }

catch(PDOException $e){

echo '{"error": {"text": '.$e->getMessage().'}';
```

```
    }
});
//For Actuator service get data
$app->get('/api/action/{id}', function(Request $request, Response $response){
    $id = $request->getAttribute('id');
    $query = "SELECT * FROM actuator WHERE id = $id";
try{
        // Get DB Object
        $db = new db();
        // Connect
        $db = $db->connect();
        $stmt = $db->query($query);
        $action = $stmt->fetch(PDO::FETCH_OBJ);
        $db = null;
echojson_encode($action);
    } catch(PDOException $e){
echo '{"error": {"text": '.$e->getMessage().'}';
    }
});
//for actuator service  update the field
$app->put('/api/action/update/{id}', function(Request $request, Response $response){
    $id = $request->getAttribute('id');
    $action = $request->getParam('action');
 $query = "UPDATE actuator SET
                                action  = :action
                                WHERE id = $id";
try{
        // Get DB Object
        $db = new db();
        // Connect
        $db = $db->connect();
  $stmt = $db->prepare($query);
      $stmt->bindParam(':action',$action);
        $stmt->execute();
echo '{"notice": {"text": "Temperature Updated"}';
    }
catch(PDOException $e){
echo '{"error": {"text": '.$e->getMessage().'}';
    }
});
```