Autonomous service creation framework for pluggable IoT Sensors and Actuators

Abu Sayeed Bin Mozahid

(151-35-843)

Md.Ariful Islam

(151-35-937)

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

Spring-2019

APPROVAL

This **Thesis** titled on **"Autonomous service creation framework for pluggable IoT Sensors and Actuators."**, submitted by **Abu Sayeed Bin Mozahid (151-35-843)** & **Md.Ariful Islam (151-35-937)** 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

Mohammad Khaled Sohel **Internal Examiner 2**
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Prof Dr. Mohammad Abul Kashem **External Examiner**
Professor
Department of Computer Science and Engineering
Faculty of Electrical and Electronic Engineering
Dhaka University of Engineering & Technology, Gazipur

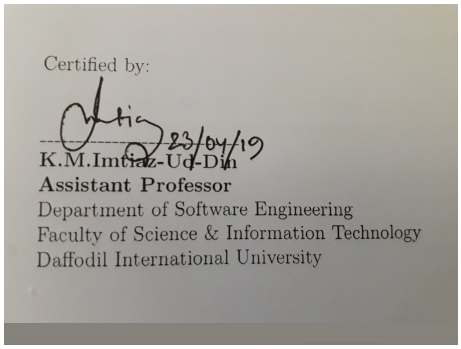
DECLARATION

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.

Abu Sayeed Bin Mozahid
ID: 151-35-843
Batch:16th
Department of Software Engineering
Faculty of Science & Information Technology

Daffodil International University

Md. Ariful Islam
151-35-937
Batch:16th
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University



ACKNOWLEDGEMENT

This document presents our Bachelor of Science Thesis "**Autonomous service creation framework for pluggable IoT Sensors and Actuators**". 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 **Kaushik Sarker** and **Md. Anwar Hossen** for all the support to execute this milestone.

Grateful to the Ambient Intelligence Lab group members especially **Diganta Protic Biswas**, **Avee Chakraborty**, **Md. Ashiqur Rahman** who always work and help with us in this research project.

We also thankful to **Professor Dr. Touhid Bhuiyan Head of 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.

Contents

List of Figures

ABSTRACT

In the advancements of technology the interest in Internet of Things domain has increased remarkably, specially in the end user kind. People are using various smart systems in order to improve their lifestyle. Therefore end users without any technical experience started using those systems. But there are no such end-user friendly smart systems where users who don't know any aspects of programming can control and add different services to the existing system based on their needs. In this work we had proposed an architecture where end-users without programming knowledge can control over the services. On the other hand service composition and adaptation at runtime is necessary in order to satisfy the users specifications. We showcase a prototypical implementation of our architecture in the domain of smart home.

Chapter 1

INTRODUCTION

Last two decades, Peoples have addicted to the internet. They are so interested in any kind of product which is spontaneously connected with internet. In the early age Internet of things (IoT), people had two options to using IoT device: Owner has to write program for their device or can use rigid IoT hardware device where no need any kinds of programing. But static IoT hardware device do not solution for dynamically work. It is okay when owner know the programing knowledge but researcheres are so worried about End-user who don't have any programming idea. Nowadays there are many home automation tools exist such as IFTTT, openHAB, Zapier, Node-RED and StreamBase Studio where users can interact with their devices within their home to perform different tasks according to Mahda Noura, Sebastian Heil and Martin Gaedke(2018). But no one concerns about hardware program where end-user will write or customize a program for their device to connect and modify their own services based on their needs in any IoT system. In this aspects we are trying to strictly using technology for creating self-adaptive system where any sensor or service can adapt to the system with owner desire and an intelligent user-interface can discover services automically.

We propose an architecture which supports microservices and distributed service and end-user gets ease to customize his hardware.To address this we have identified four major components:

Sensor or Actuator Service: A device that can be used to measure a physical property of the real world or can perform an operation or control a system.

Web Service: Supports RESTful APIs

Middleware: Computation Engine where Service adaptation and task triggering actions performed.

User Interface: An Intelligent GUI to support users to make service composition based on their needs.

Two groups have worked in this concept. Group A handle Sensor or Service and Web Service components and Group B handle Middleware and User Interface. We are working with Group A thats why rest to thesis we will be talking about sensor or service and web service.

1.1 MOTIVATING SCENARIO

Last year, Ambient Intelligence Research Lab represented an architecture where all components were distributed and end-user can compose any service with other a third party service such as an actuator service can perform with a weather API by Md. Raihan Uddin, Aziha Kamal, Shuvo Prosad, Antara Saha(2018). We have extended their approach and improving their given architecture. By a real scenario, we will describe where we have contributed and how to improve their architecture.

Scene 1: Assume there are two shops and 4 owners. One shop owner name is Akash and another shop is run by three brothers named Rahim, Akbar, Jihad. Akash bought an automated water pump system for his shop to automatic filling the tank when it's empty. Seeing this another shop owners wanted the system too but 3 of them wants the services in different ways.

Rahim wants to control the door of their shop, when any person comes closer to the door, it will open automatically. Akbar wants to control the lights and fan inside the shop, when there is nobody left on the shop lights and fans will automatically turn off and if there is someone enters through the door lights and fans will turn on automatically. Jihad wants to check the weather status and based on that he wants those lights and fans to be turned on or off.

With these thoughts they bought the required sensors and actuators but they are unable to integrate with the system so they called a professional to help with the system. Then the

expert fix the system based on their requirements and gave them an advice, not to change their wifi SSID and password. If they change those then they have call the expert again because of the complexity of the system.

1.2 OBJECTIVE

Our ambition is to build an intelligent system where users can collaborate on different IOT based services and web services and can configure the services based on their needs and the services can be adapted at run time.Each service is loosely coupled.Users only have to connect the sensor or actuator with his computer and our intelligent system will give him/her a interface to enter some pieces of information regarding the sensor or actuator and the system will automatically create the service against the sensor or actuator. And after creating the service user can customize the service and can use based on their needs. That's where our main objectives are met, This can be defined as:

1. Plug and play services
2. Runtime collaboration of web and IOT based services
3. Self-adaptive microservices

1.3 OUTLINE OF THE THESIS

Chapter 2 deigned with literature review and related technologies. Here briefly describes the related research work in this area and which technologies we are using.

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

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

In chapter 5 describes evaluation 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 investigate the state of the art in the research areas addressed in this paper. To the best of our knowledge, almost all the researchers concerned about handling middleware where end users will compose any service without any programming knowledge but they are not thinking about Hardware programming for end-users and how will they connect and register with the system to create composite services.

2.1 RELATED WORKS

A few works have done in the context of hardware such as sensors and actuators. Among them, the most remarkable one is personal application in the IoT through visual end-user programming by Valsamakis, Y., & Savidis, A. (2018), they used RFID tags for identifying different sensors and actuators.

Actinium:A Restful runtime container for scriptable internet of things applications by Matthias Kovatsch,Martin Lanter,Simon Duquennoy, they proposed a runtime environment that exposes the scripts themselves through a RESTful interface and also indicated WSN programming as the network is built upon a simple set of actions: periodic sensing,alarm triggering and actuation.

A Multi-Agent-Based Intelligent Sensor and Actuator Network Design for Smart House and Home Automation by Qingquan Sun , Weihong Nikolai Kochurov, Qi Hao, and Fei Hu. According to them based on the contextual data from the sensor a smart house can provide

services, such as energy efficiency control, security monitoring and medical assistance.

2.2 RELATED TECHNOLOGY AND FRAMEWORK

Microservice architecture: Microservice architecture is an architectural style that structures an application as a collection of services that are :

- Maintainable and testable
- Loosely coupled
- Can be deployed independently

Its an architecture that structures the application as a set of loosely coupled, collaborating services.

RESTful APIs: An API is an application programming interface.Its a set of rules that will allow different programs to communicate with each other.By creating API on the server, we are allowing clients to communicate with the server.

REST gives a set of rules that we have to follow when creating the APIs. RESTful Web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations.

In our system the clients and server exchange representations of resources by using a standardized interface and protocol HTTP.

Wireless Sensor Network(WSN): Wireless Sensor network is a network that can certainly cover a huge number of spatially distributed, little, battery-operated, embedded devices that are networked to caringly collect, process, and transfer data to the operators, and it has controlled the capabilities of computing & processing.

Distributed Service Architecture: All the components such as sensors, actuators, and web services are independent and isolated.They can function asynchronously.If any of the service is not working properly the other one will not stop to function.

Arduino-cli: Command Line Interface(CLI) is a tool that launched by arduino for compile and uploading arduino code from using windows command prompt or Linux and MAC Terminal.

Chapter 3

DESCRIPTION OF PROPOSED SYSTEM

3.1 Proposed System Architecture

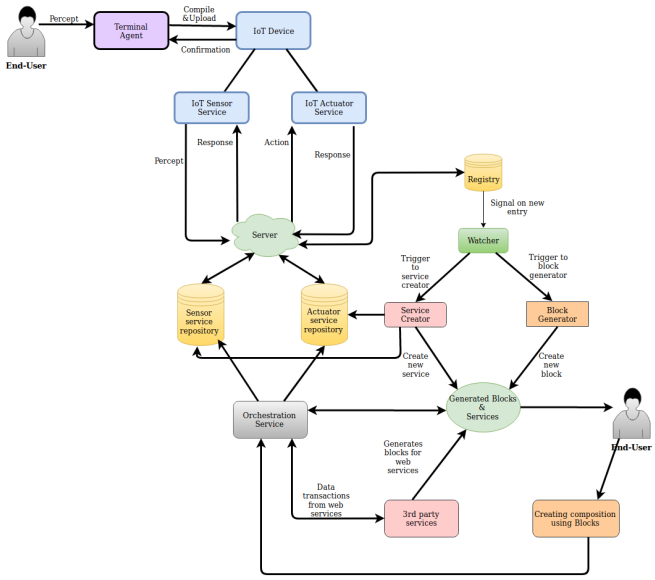


Figure 3.1: Proposed System Architecture

Figure 3.1 have described our propose architecture where users can compose their services based on their specific needs. As we are group A, we will describe Sensor service, Actuator service, Terminal Agent, Server, Orchestration Service.

End-user Interface and composite service generation engine: The user interface provides the means to coordinate the user inputs to interact with the system. End users can easily create composite services without any programming knowledge just by drag and drop. This is where the goals are issued.

Terminal Agent: The terminal agent is responsible for gathering information about the new sensor or actuator from the user and generate customize program suitable for the IoT device and connect them with the system to create service against them.

Sensor Service: Sensor can be defined as a physical object that precepts a change in the environment and responds to that change with an electrical signal output. The role of sensor service is to simultaneously monitor the environment for any state changes. If occurred the data is sent to the server and those data can be processed, analyzed, reported or can be used to trigger another event. Consider a situation where users would like their water pump to be switched on when the water level in the tank goes below a certain level. That's where the role of the sensors comes in. The sensor will measure the level of water. Our system is designed such a way that once the sensor learns that the water has gone down a particular level, it will send an alert to the controller (Middleware). The controller, in turn, would trigger an actuator to switch on the water pump.

Actuator Service: Actuator is a device that can control certain materials.Its a mechanism to turning energy into motion. It's generally used to apply force on something. In our example above, the actuator would apply force to switch on the motor of the water pump. Actuators can create linear, oscillatory or rotatory motion based on how they are designed. It can trigger various devices into operation based on the dynamics of the data.

We can classify actuators as either binary or continuous devices based upon the stable states available in their output. For example, a relay is a binary actuator as it holds two stable states, either energized and latched or de-energized and unlatched. Whether the motor is a continuous actuator because it can rotate through a full 360-degree motion.

In our architecture, we used those three types of actuators. One of them used for automated door and another two used for water tank and light fan control.

Server: As we mentioned before that we are extending previous works done by Md. Raihan Uddin, Aziha Kamal, Shuvo Prosad, Antara Saha(2018) under Ambient Intelligence Lab, they used CloudMQTT Protocol Server for connecting with gadgets, sensors and to publish and subscribe packets. We have replaced it with HTTP server for exchanging and storing data from our various IoT based devices.

The core architectural principles to the web can be defined as Representational State Transfer(REST). It shares a similar goal as web services. Web services are for increasing interoperability between the parts of a loosely coupled distributed applications, whether REST is for achieving this in a more lightweight manner seamlessly integrated to the web.

In our architecture, we proposed a system where applications are realized through scripts running on the server accessing the initial functionality of IoT based services through their RESTful interfaces. REST uses URIs for identifying different services on the web, resources can be linked, bookmarked, cached,searched for and the results are directly visible on the browser. The lightweight and ubiquitous aspects of REST makes it ideal to build APIs for our system. HTTP has been designed as a high-level application protocol and we used a RESTful interface over HTTP. For supporting heterogeneity of clients our primary protocol is HTTP/JSON based. REST makes sure that important concepts of an application scenario are represented as URIs.We can get the content of an object using an HTTP GET method. For modifying an object we can use a POST, PUT, or DELETE method.As we mentioned previously, RESTful access uses the following four HTTP verbs to determine which action to take on a particular object:

- GET: Retrieves the current state of the object
- PUT: Sets the current state of the object
- POST: Creates a new object
- DELETE: Deletes the object

Joseph Benharosh had decribed full restful api working process in his website by Figure 3.2 diagram.

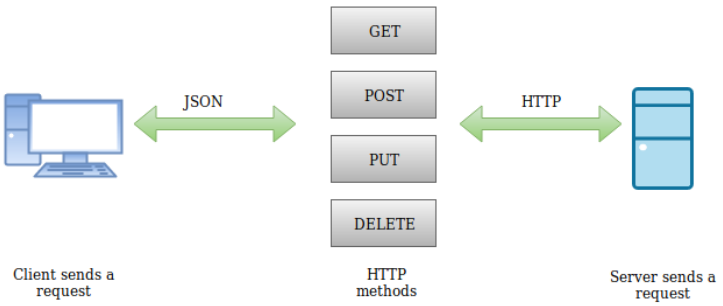


Figure 3.2: RESTful API working process

Orchestration Service: Orchestration Service details.....

Chapter 4

PROOF OF CONCEPT

4.1 IoT Sensor, Actuator and Web Service Working Module

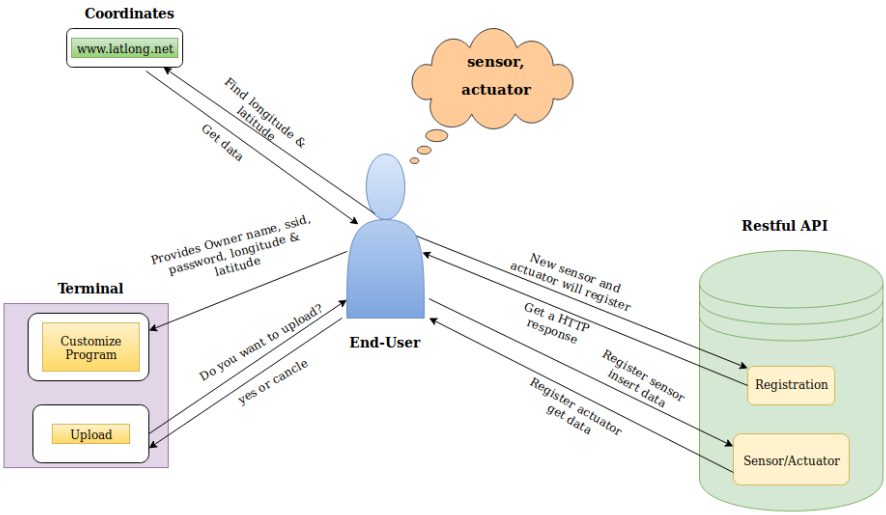


Figure 4.1: Sensor, Actuator and Web Service Working Module

Figure 4.1 depict that End-User will buy a sensor or actuator service from market and to our system for download user manual. That manual will describe how end-user will connect

their iot device with system.

4.1.1 Coordinates

For position of the sensor or actuator users has to enter lattitude and longitude before conecting to the system and our system does not provide any services for aquiring coordinates. So users has to find coordinates from another websites or apps such as www.latlong.net.

4.1.2 Terminal

After collecting all information, user will download device program customize tool from our system and provide some information. Those are

- Owner Name
- Sensor Name
- WiFi SSID and password
- Lattitude and longitude

Firstly users have to enter their name in the feild of Owner name,For entering sensor name our terminal agent will show examples of naming conventions for sensor name and users has to follow the exact same fashion. Then agent will ask for wifi SSID and password,users have to enter exactly same as their internet configuration.Lastly it will ask for coordinates and users have to gather them(lattitude,longitude) from other apps or websites. After entering these informations our terminal agent will automatically generate a customized program to connect the iot device to our system.

IoT device customize code:

```
    def create_client_file(self,
                           type,
                           owner,
                           ssid,
                           password,
```

```
6                           device_name,
7                           longitude,
8                           latitude):
9
10          if(type == ``digital_sensor"):
11              conn_file = self.digital_sensor()
12          elif(type ==``analog_sensor"):
13              conn_file = self.analog_sensor()
14          elif(type == ``ir_sensor"):
15              conn_file = self.ir_sensor()
16          elif(type == ``ultrasonic_sensor_hc_sr04"):
17              conn_file = self.ultrasonic_sensor_hc_sr04()
18          elif(type == ``soil_moisture_sensor"):
19              conn_file = self.soil_moisture_sensor()
20          elif(type == ``actuator"):
21              conn_file = self.actuator_default_file()
22
23          folder_path = self.create_dir(device_name)
24          print(``After Path: ")
25
26          print(``Owner: "+owner)
27          print('\n')
28          print(``SSID: "+ssid)
29          print('\n')
30          print(``PASSWORD: "+password)
31          print('\n')
32          print(``DEVICE: "+device_name)
33          print('\n')
34          print(``Logitude: "+longitude)
35          print('\n')
36          print(``Latitude: "+latitude)
37          print('\n')
38
39          for line in conn_file:
```

```
            line = line.replace(``owner",owner)
            line = line.replace(``wifi_name",ssid)
            line = line.replace(``wifi_password",password)
            line = line.replace(``device_name",device_name)
            line = line.replace(``sensor_long",longitude)
            line = line.replace(``sensor_lati",latitude)

            with open(os.path.join(folder_path,(device_name+``.ino")),
                      ``a") as new_file:
                new_file.write(line)
```

Tag

Our terminal program will combine the given informations by users to create an unique tag for the connected iot device.With this tag users can identify the service against that iot device as well as our system.

Device tag creating Program:

```
void create_tag()
{   sensor_tag = `` ";
    sensor_tag += sensor_name;
    sensor_tag +=``_";
    sensor_tag += longitude;
    sensor_tag +=``_";
    sensor_tag += latitude;
    sensor_tag +=``_";
    sensor_tag += owner;
    Serial.print(``Sensor Tag:");
    Serial.println(sensor_tag);
}
```

Upload

After generating the suitable program for specific iot device our terminal program will ask to user: Do you want to upload the program? y/n. If user press 'y' then program will check the operating system whether its windows or linux.Based on the operating system it will generate commands for compiling and uploading program to the device.After successfully uploading the iot device will be connected to the system and end users can easily create composite services using the device.

Device tag creating Program:

```
    def upload_command(self,sensor_name):
        print('*******Upload********')
        command1= 'arduino-cli compile --fqbn esp8266:esp8266:nodemcu'
        command1 += ' '
        command1 += os.getcwd()
        command1 += '/sketchbook/'
        command1 += sensor_name
        print('Command-1: '+command1)

        command2 = 'arduino-cli upload -p /dev/ttyUSB* --fqbn'
        command2 += ' '
        command2 += 'esp8266:esp8266:nodemcu '
        command2 += os.getcwd()
        command2 += '/sketchbook/'
        command2 += sensor_name
        print('Command-2: '+command2)

        cmd = [command1,command2]

        for c in cmd:
            self.terminal_task(c)
            print('****Command****')
        print(''END")
```

4.1.3 Context sensing:

The role of context sensing is to monitor the status of the IoT devices found at runtime and inform the corresponding components upon state change. Sensor services always monitor the status of the connected physical products.

4.1.4 RESTful API

When the IoT device is connected with our system, device will provide its tag to the server as a json data.

Sensor tag as a json data code:

```
void service_registry_json()
{
  registry_encoder[``name"] = sensor_tag;
  registry_encoder[``service_type"]= ``sensor";
  registry_encoder.prettyPrintTo(service_registry_buffer,sizeof(
   service_registry_buffer));
  Serial.println(service_registry_buffer);
}
```

Server will take that json format and deserialze the data. After deserialze data, server will check that with its registry table. If that tag is unique, server will save it otherwise it will feedback 400 HTTP response. Every device will follow this process at least three times after device can get power. If the service is already registered then the server will create a row regarding the service in the sensor or actuator registry.

We have three APIs integrating with the server.Such as:

1. serveraddress/api/serviceregistry
2. serveraddress/api/sensors
3. serveraddress/api/actuators

Every IoT device have two APIs. Service registry API is common for every device and sensors API for sensors and actuators API will exist in actuators devices.

Service registry code:

```
class ServiceRegistryApi(viewsets.ModelViewSet):
    queryset = Service_registry.objects.all()
    serializer_class = ServiceRegistrySerializer
```

Service registry deserialzer code:

```
class ServiceRegistryApi(viewsets.ModelViewSet):
    queryset = Service_registry.objects.all()
    serializer_class = ServiceRegistrySerializer
```

Sensor data inserting code:

```
class LowerSensorApi(viewsets.ModelViewSet):
  queryset = LowerSensor.objects.all()
  serializer_class = LowerSensorSerializer
```

Actuator data inserting code:

```
class ActuatorApi(viewsets.ModelViewSet):
  queryset = Actuator.objects.all()
  serializer_class = ActuatorSerializer
```

Chapter 5

EVALUATION

To evaluate our concept we adopted an environment where users would like to open his door when anyone comes closer to the door and turn on the light or fan when needed based on the context.Our main goal was to prove that the implemented support for self-adaptive service composition at runtime based on user requirements works as expected.There are many services can be created by the user and we kept the functionalities as simple as possible.

When instantiated our system will provide a graphical user interface with some informations such as registered services, actuators, instructions and service compostion engine. We followed the steps below:

1. User connects a sensor with his computer and by following the instructions user has to run terminal agent application. Terminal agent will take some information from the user and generate a customized program for connecting their IoT device with our system. Then it will ask end-user to upload the program to microcontroller. If uploading is successful the IoT device is configured with our system.

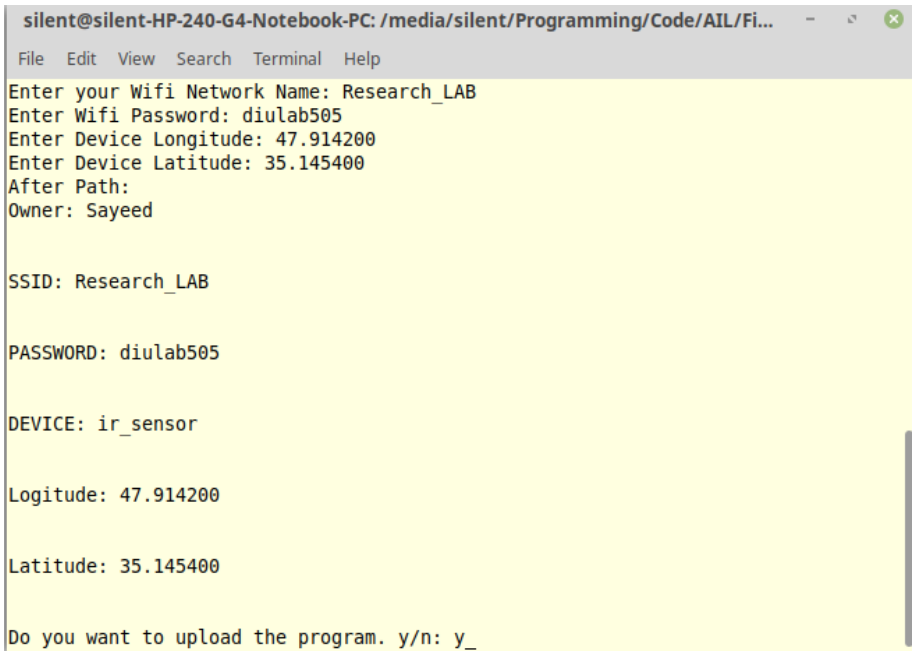


Figure 5.1: Terminal Agent procedure

2. If uploading is successful a service will be created based on the IoT device on the blocks. Our system will automatically place that block in an appropriate place whether its a sensor service or actuator service.

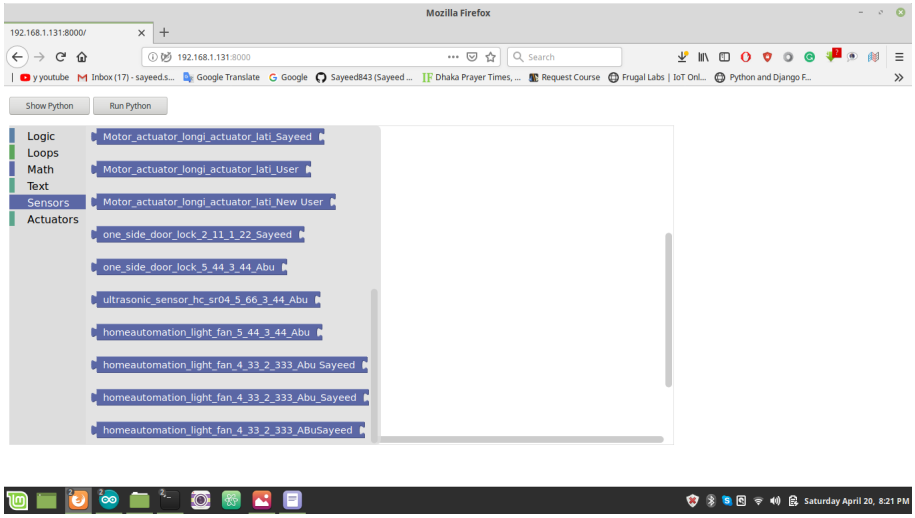


Figure 5.2: Generated Blocks

3. Now, the user can create service composition by drag and drop from the blockly blocks. They can customize and set conditional steps using the blocks.

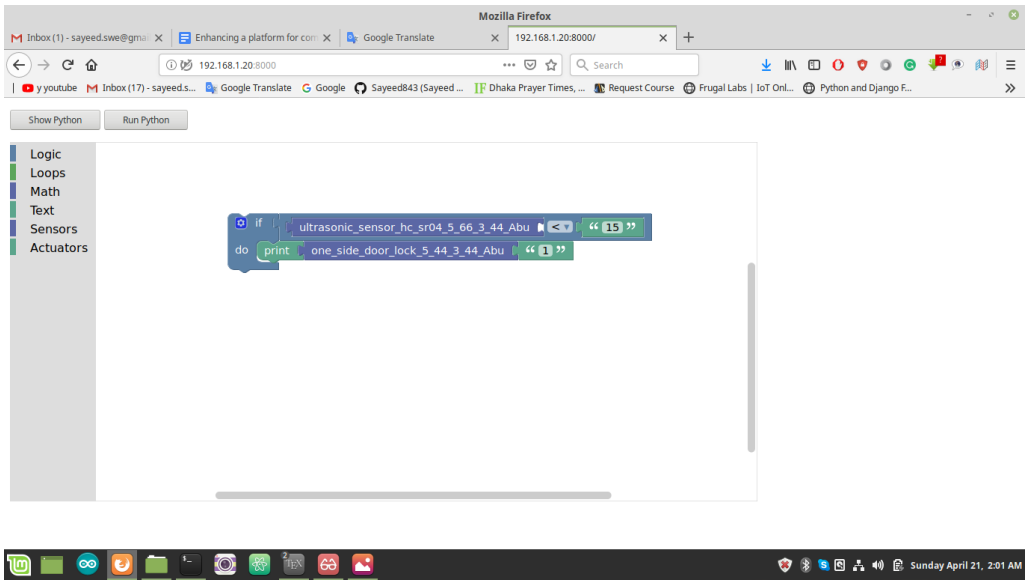


Figure 5.3: Door unlock program

After clicking run the program will send a command to the corresponding actuator for opening the door.

Figure 5.4: Door opens

4. For evaluating the concept of pluggability to create new services end-users can reuse the same sensor for multiple actuator services.

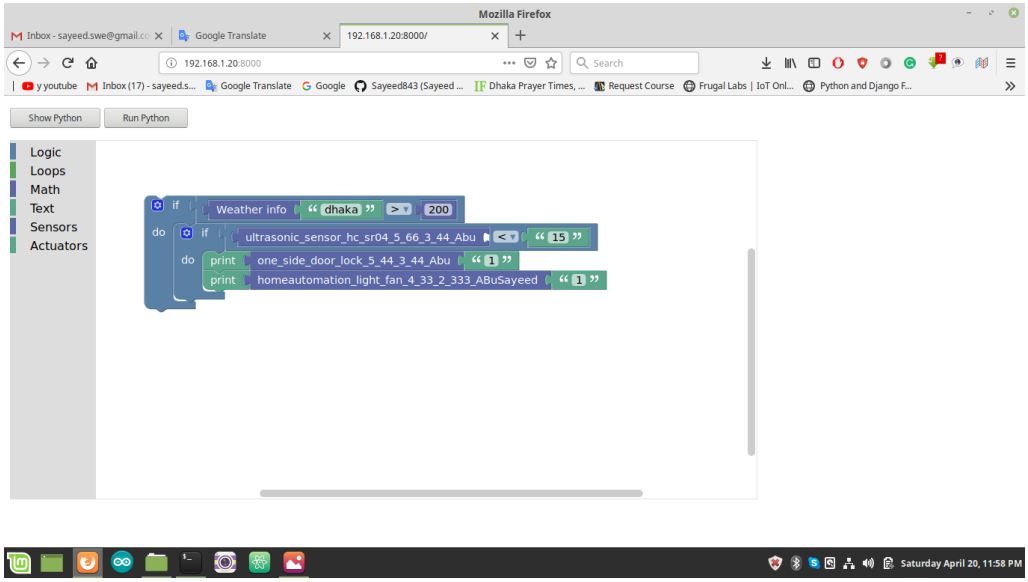


Figure 5.5: Multiple actuator service creation

5. User can unlock the door and turn on the light using the same sensor. We have set an exact range for percepting obstacles. If the sensor finds out that there is someone closer to it then it can trigger two actuator actions, one for opening the door and another is for turning on the light.

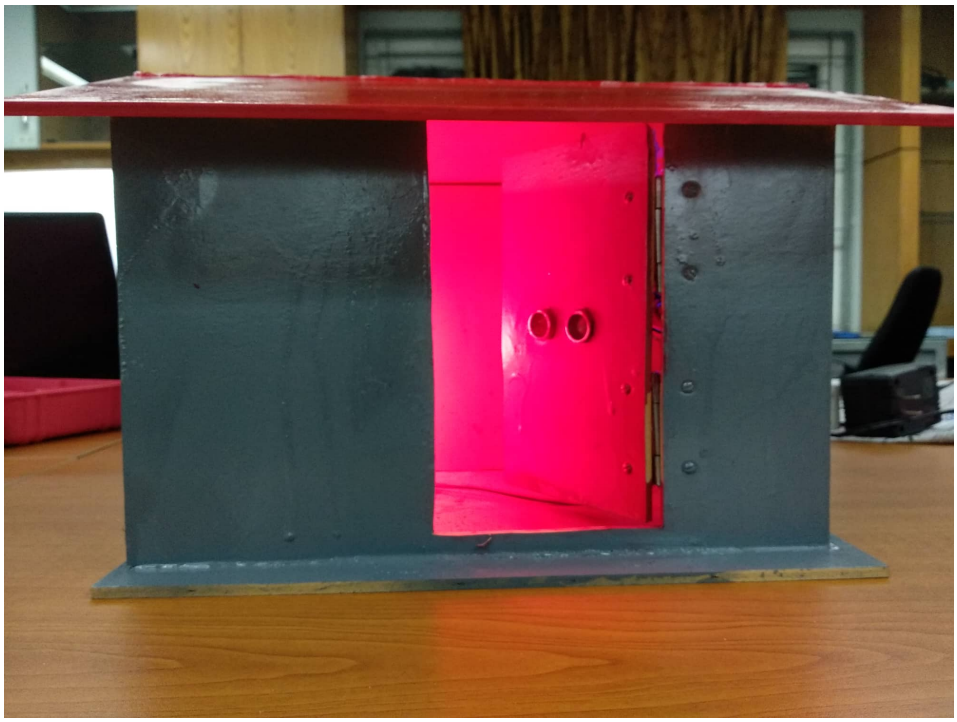


Figure 5.6: Multiple actuator actions

6. Other than IoT based services our architecture can support third party web services such as user can monitor weather conditions. With this service user can create a composition using IoT based services. We can compose weather api and sensor services data together and trigger actuator actions based on that data.

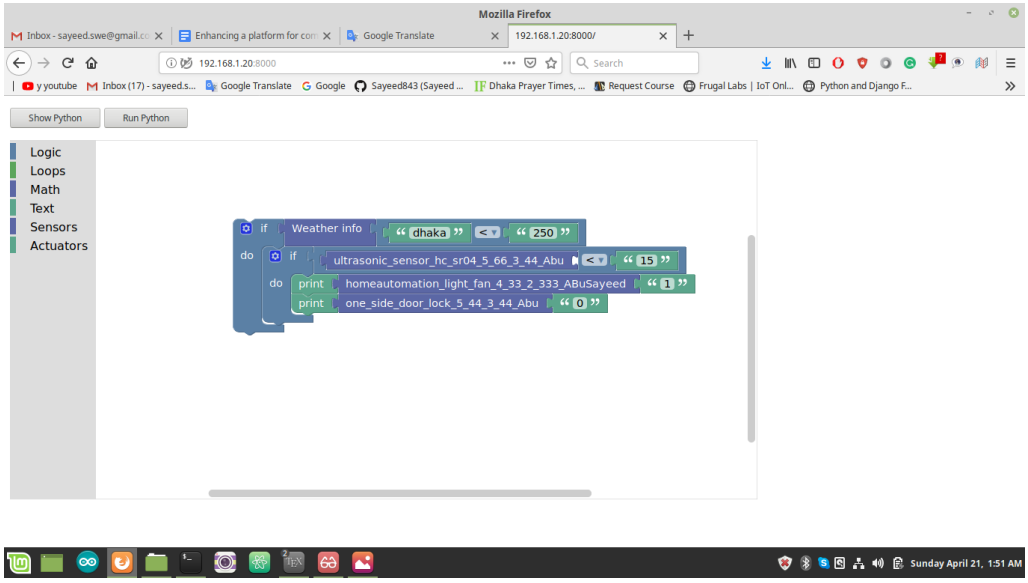


Figure 5.7: Actuator service creation with weather api and sensor

7. Suppose we have a situation where the weather condition is really bad and user needs to turn on the light. He can create a service composition with weather api and ultrasonic sonar sensor and can trigger the actuator for turing on the light and closing the door.

Figure 5.8: Actuator service action with weather api and sensor

Chapter 6

CONCLUSION

We presented in this paper a platform we are building for the Internet of Things, where end-users can easily compose their services based on their necessities. Our architecture can excite end-user programming in a new way. They can control their services through service composition and runtime adaptation.If consumers want to add new services we are providing the mechanism to connect the IoT device to our platform for creating a iot based service and that makes our architecture pluggable. Services are loosely coupled if any of them failed another one can run smoothly. A proof of concept validated our approach.

In future we would like to add more functionalities to our architecture so that end users can create more complex service composition and to make our system more effective and intelligent. We want to analyze end-users previous data to automatically discover and identifying specific services for better decision making and performance.

Chapter 7

REFERENCE

1. Mahda Noura, Sebastian Heil, & Martin Gaedke (2018). GrOWTH: Goal-Oriented End User Development for Web of Things Devices. *Proceeding of the International Conference on Web Engineering.*

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

3. Matthias Kovatsch, Martin Lanter , Simon Duquennoy (2012) .Actinium: A RESTful runtime container for scriptable Internet of Things applications. *Proceeding of the 3rd IEEE International Conference on the Internet of Things.*

4. Qingquan Sun , Weihong Nikolai Kochurov, Qi Hao, and Fei Hu (2013) .A Multi-Agent-Based Intelligent Sensor and Actuator Network Design for Smart House and Home Automation. *Proceeding of Journal of Sensor and Actuator Networks.*

5. Kashif dar, Amirhosein Taherkordi, Roman Vitenberg, Roman Rouvoy and Frank Eliassen (2011). Adaptable service composition for very-large-scale Internet of Things systems. *Proceedings of the Workshop on Posters and Demos Track.*

6. Joseph Benharosh. What is REST API? in plain English. *Describe on his blog www.phpenthusiast.com at July 31, 2018.*

Appendix A

SOURCE CODE

Terminal Agent:
System_connection.py

```
import os
import shutil

class Arduino_File():

    def __init__(self):
        path = os.getcwd() +"/sketchbook"
        if os.path.exists(path):
            print("Exist")
            shutil.rmtree(path)

    def create_dir(self,device_name):
        folder_path = os.getcwd() +''/sketchbook/"+device_name

        if not os.path.exists(folder_path):
            os.makedirs(folder_path)
        return folder_path

```

```
19
20 ######### Sensor Defualt Code #################
21     def digital_sensor(self):
22         with open (``File/digital_default.ino",``r+") as f:
23             f = f.readlines()
24         return f
25
26     def analog_sensor(self):
27         with open (``File/analog_default.ino",``r+") as f:
28             f = f.readlines()
29         return f
30
31     def ir_sensor(self):
32         with open (``File/ir_sensor_default.ino",``r+") as f:
33             f = f.readlines()
34         return f
35
36     def ultrasonic_sensor_hc_sr04(self):
37         with open (``File/ultrasonic_sensor_hc-sr04_default.ino",``r+") as f:
38             f = f.readlines()
39         return f
40
41     def soil_moisture_sensor(self):
42         with open (``File/soil_moisture_sensor_default.ino",``r+") as f:
43             f = f.readlines()
44         return f
45
46
47 ######### Sensor Defualt Code #################
48
49 ########## Actuator Defualt Code ###############
50
51     def water_tank_motor_default_file(self):
52         with open (``File/water_tank_motor_default.ino",``r+") as f:
```

```
53              f = f.readlines()
54          return f
55
56
57      def one_side_door_lock_default_file(self):
58          with open (``File/one_side_door_lock_default.ino",``r+") as f:
59              f = f.readlines()
60          return f
61
62      def homeautomation_light_fan_default_file(self):
63          with open(``File/one_side_door_lock_default.ino",``r+") as f:
64              f = f.readlines()
65          return f
66
67 ########### Actuator Defualt Code ################
68
69
70      def create_client_file(self,
71                             type,
72                             owner,
73                             ssid,
74                             password,
75                             device_name,
76                             longitude,
77                             latitude):
78
79
80          if(type == ``digital_sensor"):
81              conn_file = self.digital_sensor()
82          elif(type ==``analog_sensor"):
83              conn_file = self.analog_sensor()
84          elif(type == ``ir_sensor"):
85              conn_file = self.ir_sensor()
86          elif(type == ``ultrasonic_sensor_hc_sr04"):
```

```
87              conn_file = self.ultrasonic_sensor_hc_sr04()
88          elif(type == ``soil_moisture_sensor"):
89              conn_file = self.soil_moisture_sensor()
90          elif(type == ``one_side_door_lock"):
91              conn_file = self.one_side_door_lock_default_file()
92          elif(type == ``water_tank_motor"):
93              conn_file = self.water_tank_motor_default_file()
94          elif(type==``homeautomation_light_fan"):
95              conn_file = self.homeautomation_light_fan_default_file()
96
97          folder_path = self.create_dir(device_name)
98          print(``After Path: ")
99
100         print(``Owner: "+owner)
101         print('\n')
102         print(``SSID: "+ssid)
103         print('\n')
104         print(``PASSWORD: "+password)
105         print('\n')
106         print(``DEVICE: "+device_name)
107         print('\n')
108         print(``Logitude: "+longitude)
109         print('\n')
110         print(`Latitude: "+latitude)
111         print('\n')
112
113         for line in conn_file:
114             line = line.replace(``owner_name",owner)
115             line = line.replace(``wifi_name",ssid)
116             line = line.replace(``wifi_password",password)
117             line = line.replace(``device_name",device_name)
118             line = line.replace(``sensor_long",longitude)
119             line = line.replace(``actuator_long",longitude)
120             line = line.replace(``sensor_lati",latitude)
```

```
121             line = line.replace(''actuator_lati",latitude)
122
123             # self.delete_file(sensor_name)
124             #os.path.join(path, ''MakeFile")
125             with open(os.path.join(folder_path,(device_name+''.ino")), "a") as
    new_file:
126                 new_file.write(line)
127
128     def delete_file(self, file_name):
129         if os.path.exists(file_name+''.ino"):
130             os.remove(file_name+''.ino")
131
132 if __name__ == ''__main__":
133     a = Arduino_File()
134
135
136     print(''1: Sensor")
137     print(''2: Actuator")
138
139     choose = input(''Enter your choose: ")
140     if (choose == ''1"):
141         print(''1: Soil Moisture Sensor")
142         print(''2: Ultrasonic Sensor HC-SR04")
143         print(''3: IR Sensor")
144         print(''4: Analog Sensor")
145         print(''5: Digital Sensor")
146
147         choice = input(''Enter your choice: ")
148
149         owner = input(''Sensor Owner Name:")
150         device = input(''Sensor Name: ")
151         ssid = input(''Enter your Wifi Network Name: ")
152         password  = input(''Enter Wifi Password: ")
153         longitude = input(''Enter Device Longitude: ")
```

```
154         latitude = input(``Enter Device Latitude: '')
155         device = device.replace(``-'',``_'')
156         device = device.replace(`` '',``_'')
157 
158         if(choice == `1'):
159             #Soil Moisture Sensor
160             a.delete_file(device)
161             a.create_client_file(device,
162                                  owner,
163                                  ssid,
164                                  password,
165                                  device,
166                                  longitude,
167                                  latitude)
168             # main(device)
169         elif(choice == `2'):
170             #Ultrasonic Sensor HC-SR04
171             a.delete_file(device)
172             a.create_client_file(device,
173                                  owner,
174                                  ssid,
175                                  password,
176                                  device,
177                                  longitude,
178                                  latitude)
179             # main(device)
180         elif(choice == `3'):
181             #IR Sensor
182             a.delete_file(device)
183             a.create_client_file(device,
184                                  owner,
185                                  ssid,
186                                  password,
187                                  device,
```

```
188                                 longitude,
189                                 latitude)
190             # main(device)
191         elif(choice == '4'):
192             #Analog Sensor
193             a.delete_file(device)
194             a.create_client_file(device,
195                                 owner,
196                                 ssid,
197                                 password,
198                                 device,
199                                 longitude,
200                                 latitude)
201             # main(device)
202         elif(choose == '2"):
203             #Digital Sensor
204             a.delete_file(device)
205             a.create_client_file(device,
206                                 owner,
207                                 ssid,
208                                 password
209                                 ,device,
210                                 longitude,
211                                 latitude)
212             # main(device)
213     elif(choose=='2"):
214         owner = input(''Sensor Owner Name:")
215         device = input(''Actuator Name: ")
216         ssid = input(''Enter your Wifi Network Name: ")
217         password  = input(''Enter Wifi Password: ")
218         longitude = input(''Enter Device Longitude: ")
219         latitude = input(''Enter Device Latitude: ")
220
221         a.delete_file(device)
```

```
222         a.create_client_file(``actuator",
223                              owner,
224                              ssid,
225                              password,
226                              device,
227                              longitude,
228                              latitude)
229         # main(device)
```

terminal.py

```
import os
import subprocess
import shutil
from System_connection import Arduino_File



class Configration():
    download_link =``https://downloads.arduino.cc/"
    download_link +=``arduino-cli/arduino-cli-latest-linux64.tar.bz2"

    def __init__(self, file_name):
        self.file_name = file_name

        if not os.path.exists(os.getcwd()+``/sketchbook/"):
            os.makedirs(os.getcwd()+``/sketchbook/")

        self.file_path = os.getcwd() +``/sketchbook/"+self.file_name


    def admin_mode(self):
        admin_cmd = ``sudo su"
        # print(``Admin Password\n")
        self.terminal_task(admin_cmd)

    def download_arduino_alpha(self):
        file = self.file_split(self.download_link,"/")
        length = len(file)
        file = file[length-1]
        if not os.path.exists(file):
            download_cmd = ``sudo wget -P "+os.getcwd()+" "
            download_cmd += self.download_link
            print(``Download Link: "+download_cmd)
```

```
34              self.terminal_task(download_cmd)
35              print(``***Download Done***")
36
37      def file_split(sefl,name,mark):
38          file = name
39          file = file.split(mark)
40          return file
41
42      def extract_file(self):
43          file = self.file_split(self.download_link,``/")
44          length = len(file)
45          file = file[length-1]
46          extract_cmd = ``sudo tar xvjf "+str(file)
47          self.terminal_task(extract_cmd)
48
49
50      def file_rename(self):
51          # print(``File Rename: ")
52          new_file=``"
53          file = self.file_split(self.download_link,``/")
54          length = len(file)
55          old_file = file[length-1]
56          print(old_file)
57          old_file_split = self.file_split(old_file,``-")
58          new_file +=old_file_split[0]+``-"+old_file_split[1]
59          print(new_file)
60
61          for os_file in os.listdir(os.getcwd()):
62              if os_file.endswith("linux64"):
63                  old_file = os.path.join(os.getcwd(), os_file)
64                  rename_cmd = ``mv "+str(old_file)+`` "+str(new_file)
65                  print(rename_cmd)
66                  self.terminal_task(rename_cmd)
67
```

```
68
69      def file_move(self,f_name=None):
70          if f_name == None:
71              print(``File Move")
72              new_file=""
73              file = self.file_split(self.download_link,``/")
74              length = len(file)
75              old_file = file[length-1]
76              old_file_split = self.file_split(old_file,``-")
77              new_file +=old_file_split[0]+``-"+old_file_split[1]
78              move_cmd = ``sudo mv "+new_file+`` /usr/bin"
79
80              if not os.path.exists(``/usr/bin/"+new_file):
81                  self.terminal_task(move_cmd)
82              else:
83                  # path =os.getcwd() +``/"+new_file
84                  # print(``PATH "+path)
85                  # shutil.rmtree(path)
86                  os.remove(new_file)
87          elif f_name != None:
88              move_cmd = ``sudo mv "+f_name+`` /usr/bin"
89
90              if not os.path.exists(``/usr/bin/"+f_name):
91                  self.terminal_task(move_cmd)
92              else:
93                  # path = os.getcwd() +``/"+new_file
94                  # print(``PATH"+path)
95                  # shutil.rmtree(path)
96                  os.remove(f_name)
97
98      def terminal_task(self, command):
99          cmd = subprocess.Popen(command,shell = True,
100                                 stdout = subprocess.PIPE,
101                                 stderr = subprocess.PIPE,
```

```
                                stdin = subprocess.PIPE)
        output = str(cmd.stdout.read()+cmd.stderr.read(),''utf-8")
        print(output)
        print(''\n")


    def create_file(self,name):
        with open(name+''.yaml",''a") as f:
            f.write(''board_manager:\n")
            f.write(''  additional_urls:\n")
            f.write(''\t- http://arduino.esp8266.com/stable/
    package_esp8266com_index.json\n")

    def esp8266_installation(self):
        print(''*****Esp8266 Installation -1****")
        command1 = ''arduino-cli core update-index"
        command2 = ''arduino-cli core install esp8266:esp8266"
        cmd = [command1,command2,command1]
        print(''*****Esp8266 Installation -2****")
        print(''CMD: "+ str(cmd))
        for c in cmd:
            print(''ESP8266 Command: "+str(c))
            self.terminal_task(c)



    def upload_command(self,sensor_name):
        print(''*******Upload********")
        command1= ''arduino-cli compile --fqbn esp8266:esp8266:nodemcu"
        command1 += '' "
        command1 += os.getcwd()
        command1 += ''/sketchbook/"
        command1 += sensor_name
        print(''Command-1: "+command1)
```

```
135
136         command2 = ``arduino-cli upload -p /dev/ttyUSB* --fqbn"
137         command2 += `` "
138         command2 += ``esp8266:esp8266:nodemcu "
139         command2 += os.getcwd()
140         command2 += ``/sketchbook/"
141         command2 += sensor_name
142         print(``Command-2: "+command2)
143
144         cmd = [command1,command2]
145
146         for c in cmd:
147             self.terminal_task(c)
148             print(``****Command****")
149         print(``END")
150 def main(device):
151     # sensor_name = input(``Sensor Name: ")
152     choose = input(``Do you want to upload the program. y/n: ")
153     if choose == ``y":
154         c = Configration(device)
155
156         #step-1
157         c.download_arduino_alpha()
158         c.extract_file()
159         c.file_rename()
160         c.file_move()
161
162         #step-2
163         c.create_file(``arduino-cli")
164         c.file_move(``arduino-cli.yaml")
165         c.esp8266_installation()
166
167         #step-3
168         c.upload_command(device)
```

```
169     else:
170         print(``Upload cancel.")
171
172
173
174
175 if __name__ == ``__main__":
176     a = Arduino_File()
177
178
179     print(``1: Sensor")
180     print(``2: Actuator")
181
182     choose = input(``Enter your choose: ")
183     if (choose == ``1"):
184         print(``1: Soil Moisture Sensor")
185         print(``2: Ultrasonic Sensor HC-SR04")
186         print(``3: IR Sensor")
187         print(``4: Analog Sensor")
188         print(``5: Digital Sensor")
189
190         choice = input(``Enter your choice: ")
191
192         owner = input(``Sensor Owner Name:")
193         # device = input(``Sensor Name: ")
194         ssid = input(``Enter your Wifi Network Name: ")
195         password  = input(``Enter Wifi Password: ")
196         longitude = input(``Enter Device Longitude: ")
197         latitude = input(``Enter Device Latitude: ")
198         # device = device.lower()
199         # device = device.replace(``-",``_")
200         # device = device.replace(`` ",``_")
201
202         if(choice == `1'):
```

```
203             #Soil Moisture Sensor
204             a.delete_file(``soil_moisture_sensor")
205             a.create_client_file(``soil_moisture_sensor", #device,
206                                  owner,
207                                  ssid,
208                                  password,
209                                  ``soil_moisture_sensor",
210                                  longitude,
211                                  latitude)
212             main(``soil_moisture_sensor")
213         elif(choice == ``2'):
214             #Ultrasonic Sensor HC-SR04
215             a.delete_file(``ultrasonic_sensor_hc_sr04")
216             a.create_client_file(``ultrasonic_sensor_hc_sr04",#device,
217                                  owner,
218                                  ssid,
219                                  password,
220                                  ``ultrasonic_sensor_hc_sr04",
221                                  longitude,
222                                  latitude)
223             main(``ultrasonic_sensor_hc_sr04")
224         elif(choice == `3'):
225             #IR Sensor
226             a.delete_file(``ir_sensor")
227             a.create_client_file(``ir_sensor",#device,
228                                  owner,
229                                  ssid,
230                                  password,
231                                  ``ir_sensor",
232                                  longitude,
233                                  latitude)
234             main(``ir_sensor")
235         elif(choice == `4'):
236             #Analog Sensor
```

```
237             a.delete_file(``analog_sensor")
238             a.create_client_file(``analog_sensor",#device,
239                                  owner,
240                                  ssid,
241                                  password,
242                                  ``analog_sensor",
243                                  longitude,
244                                  latitude)
245             main(``analog_sensor")
246         elif(choose == ``5"):
247             #Digital Sensor
248             a.delete_file(``digital_sensor")
249             a.create_client_file(``digital_sensor",#device,
250                                  owner,
251                                  ssid,
252                                  password,
253                                  ``digital_sensor",
254                                  longitude,
255                                  latitude)
256             main(``digital_sensor")
257     elif(choose==``2"):
258         print(``1: Water Tank Motor ")
259         print(``2: One side Door Lock")
260         print(``3: Homeautomation Light,Fan etc")
261 
262         choice = input(``Enter your choice: ")
263 
264 
265         owner = input(``Actuator Owner Name:")
266         #device = input(``Actuator Name: ")
267         ssid = input(``Enter your Wifi Network Name: ")
268         password  = input(``Enter Wifi Password: ")
269         longitude = input(``Enter Device Longitude: ")
270         latitude = input(``Enter Device Latitude: ")
```

```
271
272         if(choice == '1'):
273             #Analog Sensor
274             a.delete_file(''water_tank_motor")
275             a.create_client_file(''water_tank_motor",#device,
276                                  owner,
277                                  ssid,
278                                  password,
279                                  ''water_tank_motor",
280                                  longitude,
281                                  latitude)
282             main(''water_tank_motor")
283         elif(choice == '2'):
284             #Analog Sensor
285             a.delete_file(''one_side_door_lock")
286             a.create_client_file(''one_side_door_lock",#device,
287                                  owner,
288                                  ssid,
289                                  password,
290                                  ''one_side_door_lock",
291                                  longitude,
292                                  latitude)
293             main(''one_side_door_lock")
294
295         elif(choice == '3'):
296             a.delete_file(''homeautomation_light_fan")
297             a.create_client_file(''homeautomation_light_fan",
298                                  owner,
299                                  ssid,
300                                  password,
301                                  ''homeautomation_light_fan",
302                                  longitude,
303                                  latitude)
304             main(''homeautomation_light_fan")
```

automatic_writter.py

```
def actuator_function_writter(function_name,tag):
  print(``writting actuator_function_writter")
  file = open(``test_file.py",``a+")

  code = ``````
def `````` + function_name+"""(motor_status):
  url = 'http://192.168.1.137:8000/api/actuators/'

  data = {

    ``topic": ```````+tag+"""',
    ``value": motor_status,
    ``time": ``2019-01-24T13:35:24.246226Z",
    ``name": ``1"
        }



  #Call REST API
  response = requests.put(url, data=data)

  #Print Response
  print(response.text)

  """

  file.write(code)

  file.close()



```

```
34
35
36
37 def sensor_function_writter(function_name,tag):
38   file = open(``test_file.py",``a+")
39
40   code = ```
41 def ``````+function_name+"""():
42   empty_list=[]
43   sensor_tag = ``````+tag+"""
44   data = requests.get(``http://192.168.1.137:8000/api/lowersensors/").json()
45   for i in data:
46     if i[`topic']==sensor_tag:
47       empty_list.append(i)
48
49   data = empty_list[-1]
50   print(data)
51
52   data = int(data['value'])
53   return(data)
54 """
55
56   file.write(code)
57
58   file.close()
```

BlockGenerator.py

```
1 #This file include All Block Generator Functions
2 from .models import *
3 import string
4 import random
5 from django.http import HttpResponse
6 from django.shortcuts import render, redirect
7
```

```
#Random Name Generator
# This Function Will be Replaced With Watcherb
def FunctionNameGenerator(size=6, chars=string.ascii_uppercase + string.digits
    ):
    return ''.join(random.choice(chars) for _ in range(size))

#Main Execution Function
def BlockGenerator(nameid, service_type):
    BlockName = nameid
    BlockType = service_type
    BlockUiGenerator(BlockName,BlockType)
    BlockDeclaration(BlockName)
    BlockJsGenerator(BlockName,BlockType)
    BlockJsDeclaration(BlockName)
    print('got it')

# This Function Genearte UI of The Block
def BlockUiGenerator(BlockName, BlockType):
    f= open(''templetes/blocks.js",''a+")
    BlockUiContainer = GetBlockUI(BlockName,BlockType)
    f.write(BlockUiContainer)
    f.close()

# This function Declare block to make it visible to End User
def BlockDeclaration(BlockName):
  f= open(''templetes/blocks_declaration.html",''a+")
  f.write('\n'+''<block type="+''''+BlockName+'" '+"></block>")
  f.close()

#Generate JS file for show code
def BlockJsGenerator(BlockName, BlockType):
    f= open(''static/js/generators/codegenerator-"+BlockName+''.js",''w+")
    BlockJsContainer = GetBlockJs(BlockName,BlockType)
    f.write(BlockJsContainer)
```

```
41      f.close()
42
43  #return Type of Block Ui you want
44  def GetBlockUI(BlockName, BlockType):
45      # if BlockType == 'sensor':
46      #     BlockUi = '\n\n'+''//"+BlockName+'' sensor block start"+'\n'+''
        Blockly.Blocks["+''''''+BlockName+"'"+"] = {''+'\n'+'\t'+"init: function()
        {''+'\n'+'\t\t'+"this.appendDummyInput()''+'\n'+'\t\t'+".appendField(new
        Blockly.FieldLabel(''+''''''+BlockName+"'"+"));''+'\n'+'\t\t'+"this.
        setOutput(true, 'Number');''+'\n'+'\t\t'+"this.setColour(20);''+'\n'+'\t
        '+"}"+'\n'+''}"+'\n'+''//"+BlockName+'' sensor block end"
47      #     return BlockUi
48      if BlockType=='sensor' or 'actuator':
49          BlockUi = ''''''
50              Blockly.Blocks[''''''''+BlockName+"""'] = {
51              init: function() {
52              this.appendValueInput(''to_input")
53              .setCheck(null)
54              .appendField(''''''''+BlockName+"""');
55              this.setOutput(true, null);
56              this.setColour(230);
57              this.setTooltip('');
58              this.setHelpUrl('');
59        }
60      };
61          """
62
63      return BlockUi
64
65  #Greturn type of block js you want
66  def GetBlockJs(BlockName, BlockType):
67      # if BlockType == 'sensor':
68      #     BlockJs = ''Blockly.Python["+''''''+BlockName+"'"+"] = function(Block
        ) {''+'\n'+'\t'+"var code = ''+''''+BlockName+'()"'+";"+'\n'+'\t'+''return
```

```
    [code, Blockly.Python.ORDER_ATOMIC];"+'\n'+'}"
69  #      return BlockJs
70      if BlockType=='sensor' or 'actuator':
71          BlockJs = ''''''
72              Blockly.Python['''''''+BlockName+"""'] = function(block) {
73                var value_to_input = Blockly.Python.valueToCode(block, 'to_input
    ', Blockly.Python.ORDER_ATOMIC);
74                // TODO: Assemble JavaScript into code variable.
75                var code = '''''''+BlockName+"""('+value_to_input+')';
76                // TODO: Change ORDER_NONE to the correct strength.
77                return [code, Blockly.Python.ORDER_NONE];
78              };
79          """
80      return BlockJs
81
82
83  def BlockJsDeclaration(BlockName):
84      f= open(''templetes/blocks_js_declaration.js",'a+")
85      f.write('\n\n<script src='static/js/generators/codegenerator-'+BlockName+
    '.js"></script>')
86      f.close()
```

data_processing_library1.py

```
1  #import requests library for making REST calls
2  import requests
3  import json
4
5
6
7
8  def weather_info(city):
9    api_key = ''bd27419d66c8678613e978ca561ad3f7"
10   url = ''http://api.openweathermap.org/data/2.5/forecast?q="+city+'&APPID={}
      ".format(api_key)
```

```
    data = requests.get(url).json()

    #data = data['weather']
    #print(data['name'])
    #print(json.dumps(data, indent = 4, sort_keys=True))

    data=(data['list'][0]['main']['temp'])
    print(data)
    return data




def action_motor(motor_status):
    url = 'http://192.168.1.137:8000/api/actuators/9/'

    data = {

        ''topic": ''mc101",
        ''value": motor_status,
        ''time": ''2019-01-24T13:35:24.246226Z",
        ''name": ''1"
              }



    #Call REST API
    response = requests.put(url, data=data)

    #Print Response
    print(response.text)

```

```
45
46 def get_lowersensor():
47   empty_list=[]
48   sensor_tag = 'soil_moisture_sensor::2.33::21.22::sayeed'
49   data = requests.get(``http://192.168.1.137:8000/api/lowersensors/").json()
50   for i in data:
51     if i['topic']==sensor_tag:
52       empty_list.append(i)
53
54   data = empty_list[-1]
55   print(data)
56
57   data = int(data['value'])
58   return(data)
59
60
61
62
63 def value_to_print(text):
64   print(text+" pass function")
65   return text
66
67
68 def print_content(what_to_print):
69   what_to_print=what_to_print+" print_content"
70   print(what_to_print)
71
72
73 def soil_moisture_sensor_2_33_1_22_Arif5():
74   empty_list=[]
75   sensor_tag = ``soil_moisture_sensor_2.33_1.22_Arif"
76   data = requests.get(``http://192.168.1.137:8000/api/lowersensors/").json()
77   for i in data:
78     if i['topic']==sensor_tag:
```

```
79        empty_list.append(i)
80
81    data = empty_list[-1]
82    # print(data)
83
84    data = int(data['value'])
85    print (data)
86    return data
87
88 def soil_moisture_sensor_3_22_2_111_User():
89    empty_list=[]
90    sensor_tag = ``soil_moisture_sensor_3.22_2.111_User"
91    data = requests.get(``http://192.168.1.137:8000/api/lowersensors/").json()
92    for i in data:
93      if i['topic']==sensor_tag:
94        empty_list.append(i)
95
96    data = empty_list[-1]
97    print(data)
98
99    data = int(data['value'])
100   return(data)
101
102
103 def Motor_actuator_longi_actuator_lati_Sayeed(motor_status):
104   url = `http://192.168.1.137:8000/api/actuators/'
105
106   data = {
107
108     ``topic": `Motor_actuator_longi_actuator_lati_Sayeed',
109     ``value": motor_status,
110     ``time": ``2019-01-24T13:35:24.246226Z",
111     ``name": ``1"
112         }
```

```
113
114
115
116   #Call REST API
117   response = requests.post(url, data=data)
118
119   #Print Response
120   print(response.text)
```

views.py

```
1  from django.shortcuts import render
2  from rest_framework import viewsets, generics
3  from rest_framework.views import APIView
4  from rest_framework.filters import (
5      SearchFilter,
6      OrderingFilter)
7  from .serializers import *
8  from .models import *
9  from .data_processing_library1 import *
10
11 from django.db.models import Q
12 from django.http import HttpResponse, JsonResponse
13 from django.views.decorators.csrf import csrf_protect
14 import requests
15 from . import automatic_writter
16 from . import BlockGenerator
17
18 from rest_framework.response import Response
19
20
21 # Create your views here.
22 def home(request):
23     action_motor(``0")
24     get_lowersensor()
```

```
    all_service = requests.get(``http://192.168.1.137:8000/api/serviceregistry/").json()
    #print(all_service)
    last_service = all_service[-1]

    print(last_service['id'])
    read_file = open('last-id.txt','r')
    a = read_file.read()
    read_file.close()
    name_id = last_service['name']
    sensor_tag = name_id
    name_id = name_id.replace(``.",``_")

    print(a)
    if int(a) < last_service['id']:
        print('a')
        write_file = open('last-id.txt','w+')
        write_file.write(str(last_service['id']))
        write_file.close()
        if last_service['service_type'] == 'sensor':
            automatic_writter.sensor_function_writter(name_id,sensor_tag)
            BlockGenerator.BlockGenerator(name_id,last_service['service_type'])
        elif last_service['service_type'] == 'actuator':
            automatic_writter.actuator_function_writter(name_id,sensor_tag)
            BlockGenerator.BlockGenerator(name_id,last_service['service_type'])

    return render(request, 'index.html')


class ActuatorApi(viewsets.ModelViewSet):
  queryset = Actuator.objects.all()
  serializer_class = ActuatorSerializer
```

```
56
57 class LowerSensorApi(viewsets.ModelViewSet):
58   queryset = LowerSensor.objects.all()
59   serializer_class = LowerSensorSerializer
60
61
62
63
64 class ServiceRegistryApi(viewsets.ModelViewSet):
65     queryset = Service_registry.objects.all()
66     serializer_class = ServiceRegistrySerializer
67
68
69
70 class ActuatorListApiView(generics.ListAPIView):
71     serializer_class= ActuatorSerializer
72
73     def get_queryset(self):
74         qs=Actuator.objects.all()
75         query = self.request.GET.get(``q")
76
77         if query is not None:
78             qs= qs.filter(
79                 Q(topic__icontains=query)
80                 ).distinct()
81
82         return qs
83
84
85 def getMotorStatus(request):
86     #csrfContext = RequestContext(code)
87     if request.is_ajax():
88         code = request.POST['code']
89         #function_call = request.POST['function_call']
```

```
        #motor_status = get_action_motor_status(function_call)
        print(code)
        exec(code)
    else:
        return HttpResponse('Use ajax format!')

    return JsonResponse({'code': code })
```

serializers.py

```
from rest_framework import serializers
from .models import *

class ActuatorSerializer(serializers.ModelSerializer):
  class Meta:
    model = Actuator
    fields = ('topic','value', 'time', 'name')

class LowerSensorSerializer(serializers.ModelSerializer):
  class Meta:
    model = LowerSensor
    fields = ('topic','value', 'time', 'name')


#service registry serializer
class ServiceRegistrySerializer(serializers.ModelSerializer):
  class Meta:
    model = Service_registry
    fields = ('id','name','service_type')
```