End user driven collaboration service composition framework for pluggable IoT sensor and actuator services

Avee Chakraborty

(151-35-924)

Diganta Protic Biswas

(151-35-960)

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 **"End user driven collaboration service composition framework for pluggable IoT sensor and actuator services."**, submitted by **Avee Chakraborty (151-35-924)** & **Diganta Protic Biswas (151-35-960)** 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

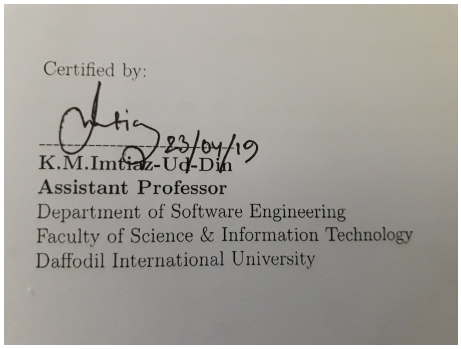
DECLARATION

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.

Diganta Protic Biswas
ID: 151-35-960
Batch:16th
Department of Software Engineering
Faculty of Science & Information Technology

Daffodil International University

Avee Chakraborty
151-35-924
Batch:16th
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University



ACKNOWLEDGEMENT

This document presents our Bachelor of Science Thesis "**End user driven collaboration service composition framework for pluggable IoT sensor and actuator services**". 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 **Abu Sayeed Bin Mozahid**, **Md. Ariful Islam**, **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 recent years,technologies in the era of Internet of things have experienced a great advancement and end user services are more capable than before. people demands to take full control over those services and personalize them. while there have been many works in end user driven service composition,to the best of our knowledge, none gives the end user the ability to build distributed dynamic composition services by building a collaboration of any 3rd party services and Iot based services. Existing work involves in iot based system support or only web based system support. Our work is step towards building a distributed end-user driven environment in order to solve the existing problem. More specifically our team's contribution lies in designing a software architecture, implementing that architecture that supports end user to control their own iot based services and 3rd party web services through any possible service composition dynamically and distributively.

Chapter 1

INTRODUCTION

1.1 MOTIVATION

Over the last decade there has been rapid progress in internet of things and end user driven services that assist us in our everyday life has increased significantly.But a less progress a has been made of combining those two technologies.The advent of new technologies such as various sensors,actuators, dynamic web frameworks,3rd party web services,simple ui-ux have fueled this growth in services. Furthermore this development has also enhanced the possibility for the realization of a end user driven services combing those technologies.Typically end users do not have a programming background but they wants to take full control over these services and personalize them to fulfill their needs. End user wants to make their own services by customizing a existing service or by composing two or more services together to fulfill their needs. end-user can not customize a service as their requirement changes over time, they software professionals to do that change for them.however those iot based services and end user driven service working independently ,could not collaborate with each other according to the end user's needs.For that reason,our motive is to design and develop an architecture,where their will be iot based services,3rd party web services and end user friendly interface can make collaborative services by compositing them together distributively, which will fulfill end user's requirement.

1.2 GOAL

The goal of this thesis is to devise, document, and present a novel approach with which an end user will design and develop composite services based on the user's need. These preferences can be updated dynamically at runtime and communicate with the project components distributively. The service composition can be adapted according to end user's need. In order to fulfill this objective, we work towards developing an intelligent software system that supports enough flexibility for end users in defining the service composition, while ensuring both the end user control,3rd party web services and iot services. A task description to realize these goals includes the following set of milestones:

- Review related work in end user service composition and the adaptation
- Review the end user-friendly ui-ux
- Review the support for developing service against user's need
- Develop a mechanism for end user service composition to include:
  - concepts related distributive service works
  - concepts related distributive architecture for distributively communicate different parts of the core component
- Implement the concepts developed above
- Test the validity of the concept and the implementation using a simple scenario and by conducting a user survey
- Document the work by writing the thesis report

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.

1.3 THESIS BACKGROUND

we are working on a lab called Ambient Intelligence Lab and the motivation for this thesis stems from our previous team's work. their objective was to build a end user friendly system where services are modular and easily pluggable without having the need of a software professional to customize and control the system. Their system was designed by many

microservices and a easily understandable user interface which provides the ability to end user to build and change their services without having the knowledge of coding. End users were able to create their own service by themselves by compositing IoT based services and web services. But there was a lot of limitations to build such thing. This project's limitations are our goal.

1.4 RESEARCH METHOD

In our thesis life cycle, we followed a systematic research method in order to achieve our goal in a precise way. the method consists of different sequential approaches, helped us to easily maintain the whole research life cycle. we have started to find the problem.when ever we found our problem, then we started analysing and reviewing another's paper. after getting our analysis report and the other's paper review report we started to designing our proposed solution.Then our proposed solution was analysed and reviewed again. We made an exit point to go again analysis and state of art state to review again if our proposed solution is not perfect. If our proposed solution is perfect then we went through the implementing our solution state,testing the solution. At the last,we found our final prototype.

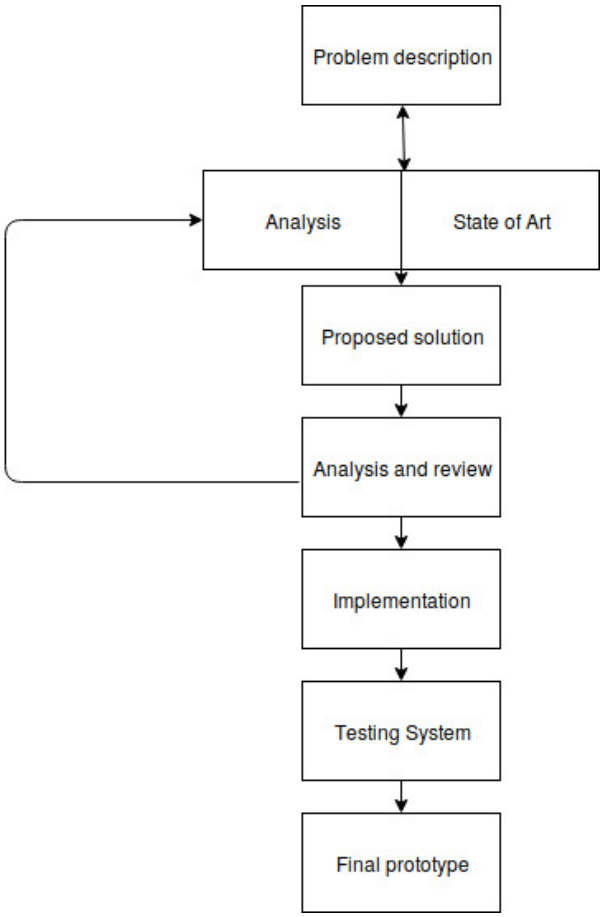


Figure 1.1: Research method

Chapter 2

SCENARIO BASED PROBLEM DESCRIPTION

2.1 SCENARIO

- Scene 1: End Users , Mr X lives in Country A and Mr Y lives in country B . They both have smart water tank refiller . At present they can only control their own devices . If they want to have more than one devices on different locations to pump water they can't do it . As all data resides in one place.
- Scene 2: In addition Mr X buys a hidro sensor , and he wants his water pump to detect the water level and work according the readings. But he can not add the sensor with the system of his . Because it needs help of professional , and the integration of a new component is quiet complex .
- Scene 3: Suppose Mr Y sets up a hydro with the help of professional. But yet he does not have full control upon his new component.As a non technical person he wants full control of his device and use the full extend of every technology .

2.2 ANALYSIS OF SCENARIO

According to the scenario the the points where technological improvement needed are :

- In scene 1 : Users can't control devices if it is distributed in different locations. They can only control those devices which are situated in the same directory.
- In Scene 2 : Users can not add new components themselves. To add a new component they need help of professionals . Even after adding those components those are not immediate functional . This means in this scenario there is a lack of pluggability.
- In Scene 3 : After the addition of new users the system is not ready to be fully controlled by a user . As different devices has different functionalities a common control architecture is missing here .

Chapter 3

STATE OF ART

We can explore more effectively in this path of making an user control friendly and dynamic and adaptive IoT architecture by simply reviewing the literature based on following questions :

- How to simplify and give full control ?
- How to make architecture loosely coupled ?
- How to make things Pluggable ?

3.1 HOW TO SIMPLIFY AND GIVE FULL CONTROL?

A paper (Shaha,A& Sharnaker, S.P, 2018) used the theory of building bridge between user control and runtime adaptation . This paper they discussed a way by which an user can develop and design a composite service at runtime . This allows a user more control over the system. They used a technique that introduces us with a concept "composite composition". This paper helps us to think about the scope of contribution in user control simplification sector . But this paper did not work with adaptive iot devices . To more relative work we can look up to the paper (Towards middleware supporting end-user driven collaborative IoT and web services).

The paper (Shaha,A& Sharnaker, S.P, 2018) also working on the same objective as they are using blockly type blocks to give full control of a system to end user but their proposed architecture is not fully distributed and it does not offer a feature to make a actual composite composition . The proposed architecture also lacs of pluggability . It means if any new component is newly attached with the system that proposed system can't handle the new components.

3.2 HOW TO MAKE ARCHITECTURE LOOSELY COUPLED?

The paper (kovatsch,M., & Lanter,M., & Duquennoy,A.,2012) proposed an architecture that provides web like scripting for low-end devices through cloud based application servers and a consistent restful programming model. Their novel runtime container actinium(AC) exposes scripts, their configuration and their lifecycle management through a fully restful programming interface using the constrained application protocol(CoAP). They endow the Javascript with an API for direct interaction with mote class iot devices the CoapRequest object and means to export script data as web resource. Although, They are not fully distributed and loosely coupled.

3.3 HOW TO MAKE THINGS PLUGGABLE ?

The paper (Hachem,S.,& Teixeira.,2011) addressed that, internet of things are amplified by the anticipated large scale development of devices and services, information flow and involved users in the IoT. The author of that paper focused on addressing those related to scalability, pluggability and heterogeneity of IoT components and the highly dynamic and unknown nature of the network topology. On that paper they showed a service -oriented middleware solution that addresses those challenges using semantic technologies to provide interoperability and flexibility. They said about some aspects of the ontology.The first aspect is the thing aspect ontology described in a device ontology,second aspect consists of real world concepts and functionalities of things, modeled in a domain ontology as mathematical formulas and third is a real world approximation unavailable service and estimated information.

For better results, they need to further investigate the sensor, actuator modeling approach on different levels of detail, as they plan on performing deeper comparison with existing solutions.

3.4 RELATED TECHNOLOGY AND FRAMEWORK

For developing this architecture, we mainly focused on highly maintainable, pluggable, loosely coupled and end user friendly technologies.

- Blockly : Blockly is a client-side JavaScript library for creating visual block programming languages and editors.
- Django : Django is a Python-based free and open-source web framework, which follows the model-view-template architectural pattern. It helps us to make pluggable architecture.
- Ajax : Ajax is a set of web development techniques using many web technologies on the client side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page.

Chapter 4

PROPOSED SOLUTION

4.1 END USER

End User is the actor of this system who does not have any programming knowledge but wants result according to his/her need .

4.2 SERVICES

Service (Imtiaz-Ud-Din, K. M., 2011) is a process that uses a set of instructions which are predefined to accomplish a specific goal . Services in this case is the perspective of user . Using services user can obtain a certain goal.

4.3 COMPOSITE COMPOSITION

Composite Composition (Imtiaz-Ud-Din, K. M., 2011) is a composition of various services and conditions . The service compositions must follow some criteria . Composite composition allows the full control of a device.

4.4 PLUGGABILITY

Pluggability is a process that automatically discovers new components and makes them functionable .

4.5 DISTRIBUTED ARCHITECTURE

In distributed architecture components are presented in different platforms or locations and communicate through network to achieve a specific goal .

Chapter 5

ARCHITECTURE

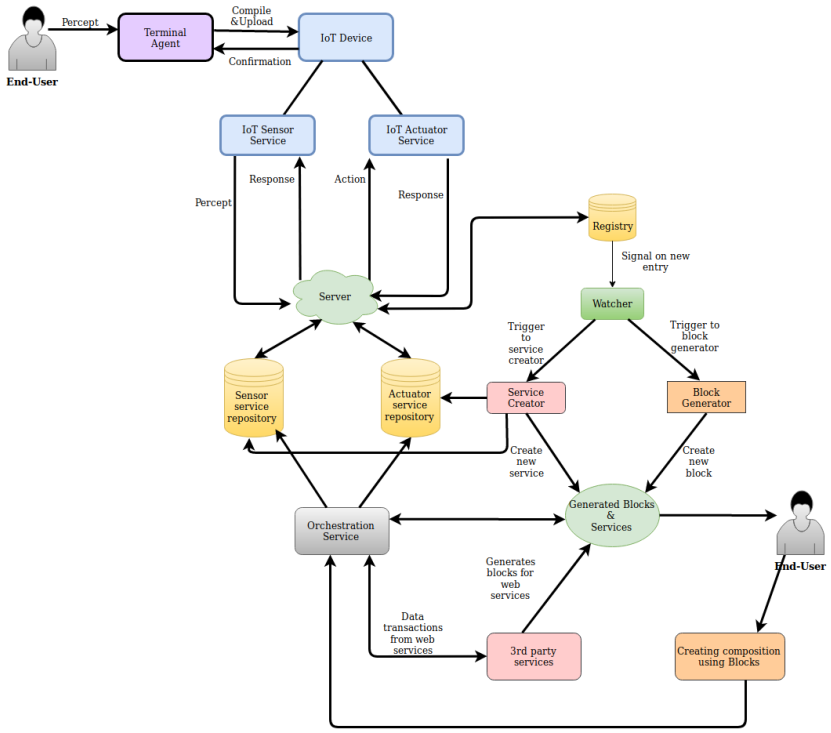


Figure 5.1: Proposed System Architecture

Our architecture consists of ten components which are distributively connected with others because all the components are loosely coupled. The above figure presents a higher

view of our proposed architecture and presents a view of communication between those component.

5.1 SERVICE REGISTRY

A service registry shown in appendix 5 is a storing mechanism. Whenever a new sensor or actuator will be added, service registry takes some sorts of information from those sensors and actuator,like sensor's/ actuators unique id, name, present state etc.

5.2 WATCHER

Watcher will distributively continuously watch the change of service registry. If a new change happens like, if a new sensor/actuator is added in service registry, then at that time watcher will send two signals for the last updated service on service registry.

5.2.1 Block generator

From watcher one signal will come into block generator phase shown in appendix 2. according to the type of service, new blocks will be generated.

5.2.2 Service creator

Another signal from the watcher will trigger the service creator section shown in appendix 3 and according to the service, service creator will create service for the latest update on the service registry.

5.3 3rd PARTY WEB SERVICES

End user can use 3rd party web services as per their choice.

5.4 GENERATED BLOCKS AND SERVICES

With the collaboration of block generator and service creator, in that phase services and blocks will be created.This phase is a collaboration of sensor iot services, actuator iot services and 3rd party web services.

5.5 COMPOSITE COMPOSITION

A composite (Imtiaz-Ud-Din, K. M., 2011) composition is created by combining 1 or more tasks , From Ubicom For All Service Composition Concepts and Notation we can say a composite combination can be made of triggers, services, conditions or/and queries .

5.6 ORCHESTRATION SERVICE

According to the re composite composition, data processing unit shown in appendix 4 will write the results from composite composition on its own section. Then this section will change the current values of sensor and actuator current state value according to the end user's need.

Chapter 6

IMPLEMENTATION

Our prototype runs with the help of django framework and the database part is independent as it is loosely coupled . The User interface is easy to use and users can combine blocks and make a dynamic commands .

6.1 USER INTERFACE

We used Blockly (Blockly, Google,2018) library , because from the perspective of a non technical end user this is easier than ever . They will just combine blocks and they can create logic that is fully functionable . What blockly does is , it generates python output of the combination depending on its predefined set of rules .

To grant full access we used ontologies from the paper(Imtiaz-Ud-Din, K. M., 2011) .Here we used various types of custom and default blocks .

- Trigger Block : Trigger is a is a set of if-else . The trigger sends a signal true or false based on some condition (Imtiaz-Ud-Din, K. M., 2011) . Upon trigger's signal the full code is executed.
- Condition Block :

  Condition (Imtiaz-Ud-Din, K. M., 2011) type blocks (Blockly,Google.,2018) compare things exactly as if-else . These Trigger and conditions are similar but their purpose is different. Condition Blocks can have nested if-else.

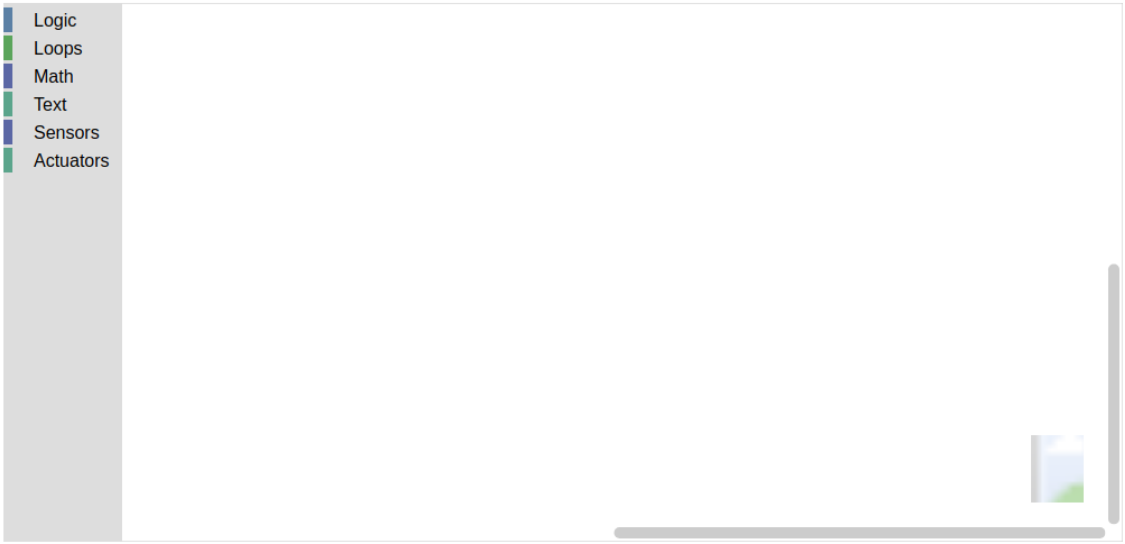


Figure 6.1: User interface

Figure 6.2: Trigger type block

- Service Block : Service(Imtiaz-Ud-Din, K. M., 2011) type blocks are the actions which are to be perform upon completion of the conditions . These are custom made blocks . It has a end which can take another block as input which we use as a perimeter of a predefined generic function . There is a socket type part at other end of these blocks which we use to attach with other blocks like print , show , etc . This part returns value from the function we mentioned earlier.

- Query Block : This is a custom made block . It can act two different ways . One is It can directly accept a value and another is it can receive return value returned from another function . Suppose you have a phone book block which is query(Imtiaz-Ud-Din, K. M., 2011) type . You can directly dial number and call that entry . And other way you can use this same block is that you can enter recipient name and it will find the number and return the number then you call him/her .

Figure 6.3: Condition type block

Figure 6.4: Service type block

- Web API Blocks : This type block is created using data we get from web api. Suppose we want to know the weather and use the data of that weather to accomplish a task . This type of block help us in this regard . User can use one or more of these Blocks at a time .

6.2 USER REQUIREMENT

User can use the blocks to express requirements . Google's Blockly made this an easy task . User can drag and drop blocks , combine them . The final combination express a full logic . To build this logic user does not have to know programming at all.

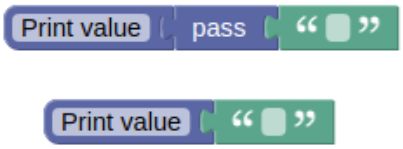


Figure 6.5: Service type block

Figure 6.6: Service type block

6.3 CODE GENERATION

A fully machine understandable code is generated from the block combinations which we get from user . The blocks use javascript to convert it into a meaningful code structure , then we can get a python output of that code using google's blockly . The code given by blockly is fully executable in python .

6.4 CODE EXECUTION

Code execution is done using python's default exec() method . After user creates the combination , the generated code is sent to the server-side using javascript . The Code then executed by python .

6.5 DISTRIBUTION OF COMPONENTS

We used Restful API to send and receive . with the help of api we can transform it into a distributed system where we do not need to keep all the components in one place. Now the Team working with the Sensor and Actuator can easily send and receive data from anywhere anytime . We designed the database for actuators and sensors and built an api that can filter the data and give the specific data to the devices .

6.6 PLUGGABILITY

This is a major part of our work . We made the architecture pluggable . So prototype is ready to have new components and it make it full functional . Here whenever we refresh the page it looks for any new components added or not. If any new device is added to the system our system will automatically create controlling blocks , and server side functions to make it fully operational . We made a generic way of writing functions that will improve the control functionalities .

Chapter 7

PROOF OF CONCEPT

7.1 DESCRIPTION TEST CASE

Mr D wants to implement a smart system to his house by himself . He went to the market and bought some sensors and actuators and other parts to automate the lights , fans , and the door . He wants to control his smart home automation based on some web service, and sensor values. He is not a programmer or tech person . So if he uses this architecture described above, the sensors and actuators will follow the plug and play concept and all the control options will generate automatically .

In the figure we can see some blocks generated by blockly , these blocks are created after a new sensor or actuator connects with the system . The name and functionality is provided into the registry . The automatic Block Generator creates these blocks using those information.

A actuator block can attach with another block . The second block acts as parameter . Here 1 means On and if the info block was 0 then it means this block is giving the actuator information to turn of.

In the figure above we can see a composite composition where user is using the data from web service , and sensor as condition checkup . We can see the condition type blocks (If statement) . Here we can see that there is a web service named weather info which returns the temperature of the place provided with the info block . this is a query type block . Then it checks it with a number . Here it checks if it is greater than 200 kelvin . After that

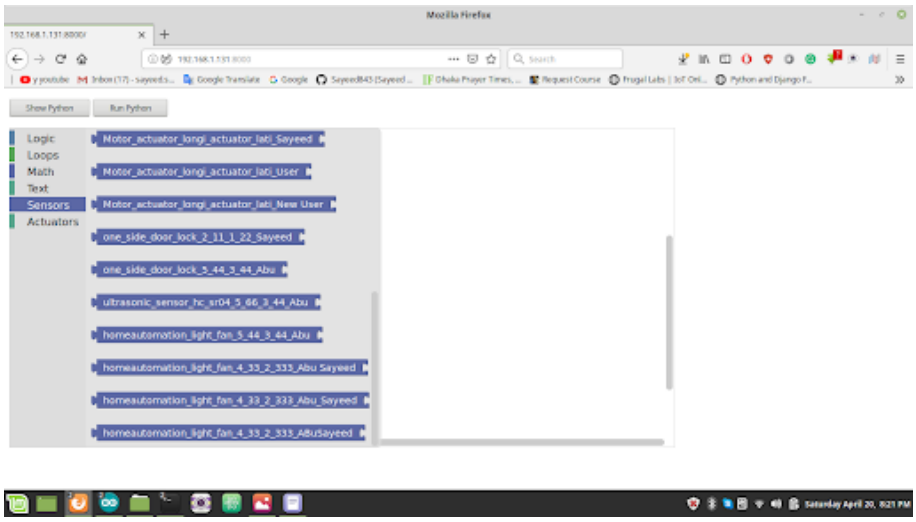


Figure 7.1: Automatically created blocks for controlling sensors/actuators

the block 'Ultrasonic_sensor_hc_sr04_5_66_3_44_abu' provides the info of sensor value and it compares with a number . if that is true then the certain block can be turn on or off.

The figure above shows the door's position before executing the combination and after executing it .

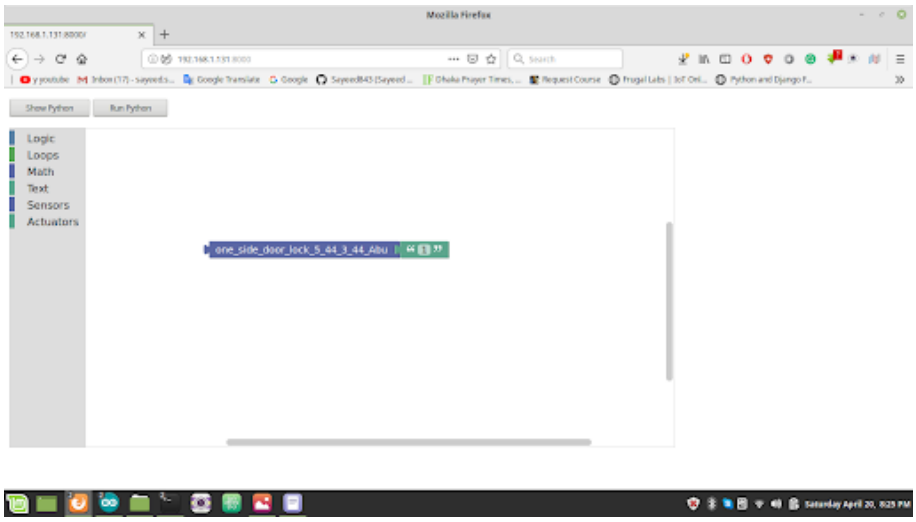


Figure 7.2: A actuator block with a information block attached

In the figure above we can see that There are two actuators . Before there was only one . This shows the dynamic and adaptability of the combinations .

The figure above shows that when the tow actuator is turned on the doors open and at the same time the lights are also on .

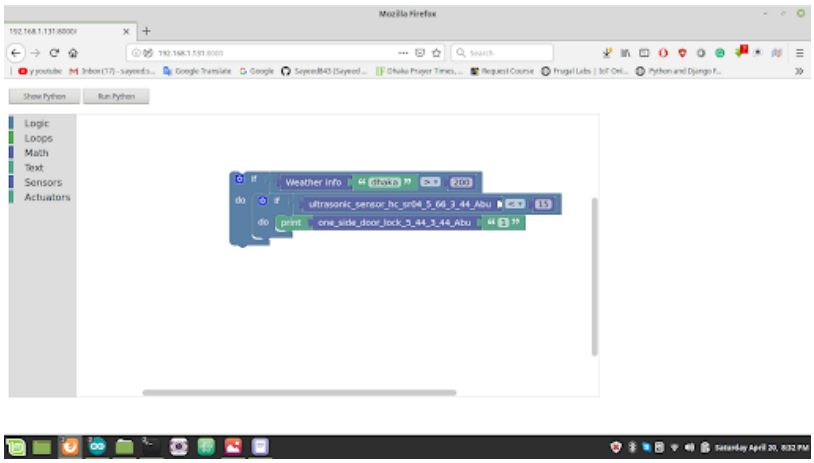


Figure 7.3: A composite Composition with sensor , actuator , web service

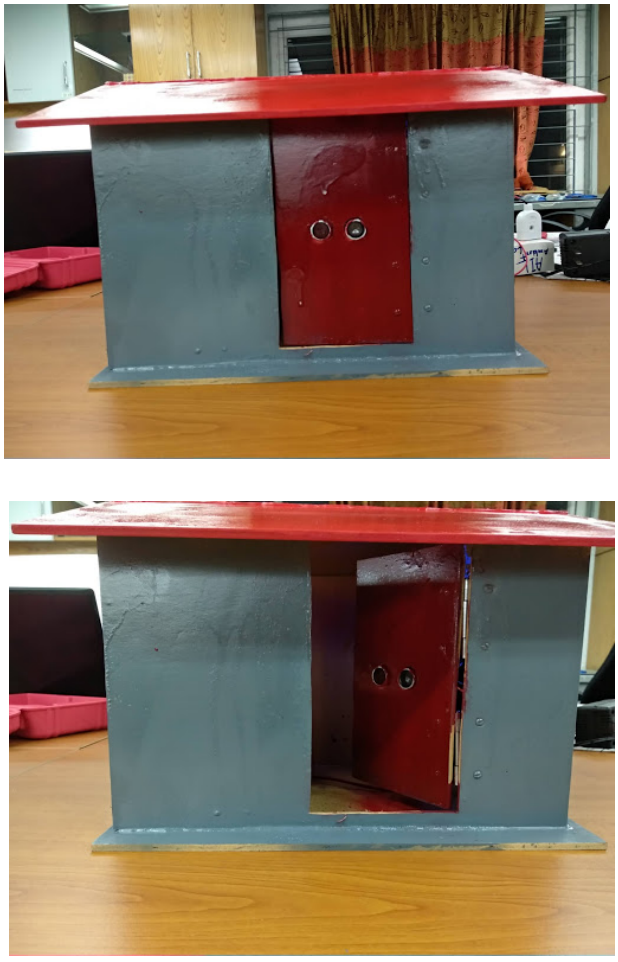


Figure 7.4: After Running The Composite Combination the door opens

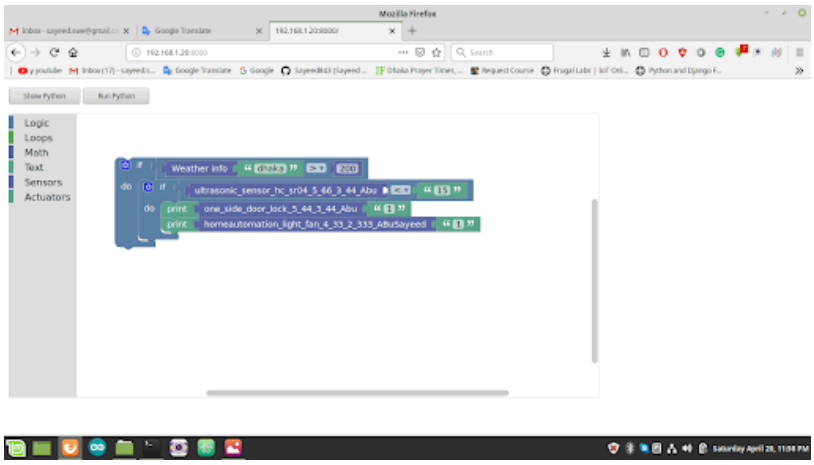


Figure 7.5: The composite combination with multiple actuator at a time

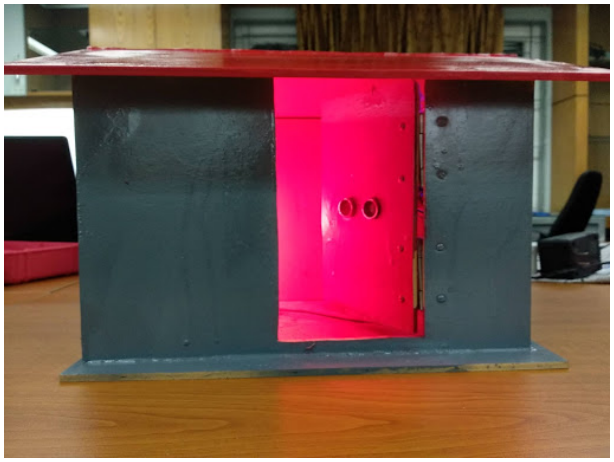


Figure 7.6: Two actuator function in real time

Chapter 8

CONCLUSION

8.1 CONTRIBUTION

Our system gets user inputs using Google's Blockly , those blocks generate a executable code which can be run by python , when it is run it can get data from different distributed database. Though blockly provided us some basic blocks like if , else etc but most of the blocks are custom made. We also made a generic way of writing services so that it can support plug and play functionality . Used the concept of API to make the system distributed and independent . Making a adaptive architecture and pluggability with flexible controlling options are the main contribution of our work .

8.2 APPLICATION

This research will help to provide non technical users with full control of a system . It will open new ways to provide End users the ability to control their services without depending on technical expertise . Using our concept remote use of services will be easier than ever . And the best thing is now End users can extend and grow the main system without the help of any professional . Non technical people like a farmer can easily set up a irrigation system . and they do not have to worry about all the technical things . This architecture will handle the server side works .

From kids to old people , everyone can setup and use various IoT devices . Till now non-technical end users were controlled by the service provider . But our research shows new way of providing the full functionality to end user .

8.3 LIMITATION OF OUR WORK

Though our system works with composite composition of services it still has a fare amount of limitations . We do not have stand alone watcher for new plugged devices , the web services are parsed from json , as the json response can be different from different websites it's not fully adaptable in this regard .

8.4 FUTURE WORK

Work on making the web services fully pluggable , Use Ai to help the system controlling more efficient . Develop the idea of a stand alone watcher to watch over the changes in system .

Reference

- Shaha,A& Sharnaker, S.P, 2018: Towards a middleware supporting end-user driven collaborative IoT and web service composition.
- Imtiaz-Ud-Din, K. M., 2011: Collaboration-based intelligent service composition at runtime by end users
- kovatsch,M., & Lanter,M., & Duquennoy,A.,2012 : Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications
- Hachem,S.,& Teixeira.,2011: Ontologies for the Internet of Things

Appendix

Index.html

```
1
2
3 {% load static %}
4 <!DOCTYPE html>
5 <html>
6 <head>
7   <meta charset="utf-8">
8   <title></title>
9   <script src="{% static 'blockly/blockly_compressed.js' %}"></script>
10  <script src="{% static 'blockly/blocks_compressed.js' %}"></script>
11  <script src="{% static 'blockly/python_compressed.js' %}"></script>
12  <script src="{% static 'blockly/msg/js/en.js' %} "></script>
13  <script src="{% static 'js/generators/codegenerator-weather.js' %}"></script
      >
14  <script src="{% static 'js/generators/codegenerator-motor-on.js' %}"></
      script>
15  <script src="{% static 'js/generators/codegenerator-motor-off.js' %}"></
      script>
16  <script src="{% static 'js/generators/codegenerator-uppersensor.js' %}"></
      script>
17  <script src="{% static 'js/generators/codegenerator-lowersensor.js' %}"></
      script>
```

```
18  <script src="{% static 'js/generators/codegenerator-print_content.js' %}"></
      script>
19  <script src="{% static 'js/generators/codegenerator-value_to_print.js' %}"
      ></script>
20  <script src="{% static 'js/generators/codegenerator-do_it.js' %}"></script>
21  <script src="{% static 'js/jquery.min.js' %}"></script>
22  {% include "blocks_js_declaration.js" %}
23  <script src="{% static 'js/jquery.min.js' %}"></script>
24  <style>
25    body {
26      background-color: #fff;
27      font-family: sans-serif;
28    }
29    h1 {
30      font-weight: normal;
31      font-size: 140%;
32    }
33  </style>
34 </head>
35 <body>
36  <!-- added three buttons in the very first -->
37  <p>
38    <button onclick="showCode()">Show Python</button>
39
40    <button onclick="runCode()" id="runcode">Run Python</button>
41     <button onclick="save()" id="save">Save Combination</button>
42      <button onclick="restore()" id="restore">Restore</button>
43    <!--not used right now
44    <button onclick="showXml()">Generate XML tree</button>
45  -->
46  {% csrf_token %}
47  </p>
48
49    <!-- work space -->
```

```
<div id="blocklyDiv" style="height: 480px; width: 1000px;"></div>
  <!-- end work space -->


  <!-- blockly blocks(default from documentation) -->
<xml id="toolbox" style="display: none">
  <category name="Logic" colour="%{BKY_LOGIC_HUE}">
    <block type="controls_if"></block>
    <block type="logic_compare"></block>
    <block type="logic_operation"></block>
    <block type="logic_negate"></block>
    <block type="logic_boolean"></block>
    <block type="do_it"></block>
  </category>
  <category name="Loops" colour="%{BKY_LOOPS_HUE}">
    <block type="controls_repeat_ext">
      <value name="TIMES">
        <block type="math_number">
          <field name="NUM">10</field>
        </block>
      </value>
    </block>
    <block type="controls_whileUntil"></block>
  </category>
  <category name="Math" colour="%{BKY_MATH_HUE}">
    <block type="math_number">
      <field name="NUM">123</field>
    </block>
    <block type="math_arithmetic"></block>
    <block type="math_single"></block>
  </category>
  <category name="Text" colour="%{BKY_TEXTS_HUE}">
    <block type="text"></block>
    <block type="text_length"></block>
```

```
84      <block type="text_print"></block>
85    </category>
86    <!-- end blockly blocks  -->
87
88
89    <!-- custom blockly bocks(made by us) -->
90        <category name="Sensors" colour="%{BKY_MATH_HUE}">
91
92      <!-- This is here to declare all blocks -->
93      {% include "blocks_declaration.html" %}
94
95    </category>
96
97    <category name="Actuators" colour="%{BKY_TEXTS_HUE}">
98
99
100
101   </category>
102   <!-- end custom blockly bocks -->
103 </xml>
104
105
106
107 <script>
108   //initialize workspace
109   //get the logic statements from blockly/media
110   var demoWorkspace = Blockly.inject('blocklyDiv',
111       {media: 'blockly/media/',toolbox: document.getElementById('toolbox')})
    ;
112   // Generate JavaScript code and display it.
113   function showCode() {
114     Blockly.Python.INFINITE_LOOP_TRAP = null;
115     //convert blockly logics into python code
116     var code = Blockly.Python.workspaceToCode(demoWorkspace);
```

```
117       alert(code);
118     }
119
120
121     function save(){
122       console.log("abcd");
123       if (typeof(Storage)!=="undefined")
124       {
125         var xml = Blockly.Python.workspaceToDom(Blockly.demoWorkspace);
126         localStorage.setItem(document.getElementById("blocklyDiv").value,
    Blockly.Python.domToText( xml ));
127         Blockly.demoWorkspace.clear();
128       }
129       else{
130         alart("error in save");
131       }
132     }
133
134
135     function restore(){
136       Blockly.demoWorkspace.clear()
137       var nameOfTheProject = document.getElementById("blocklyDiv").value;
138       if (typeof(Storage)!=="undefined"){
139         if (localStorage.data!=null){
140           var xml = Blockly.Python.textToCode(localStorage.getItem(
    nameOfTheProject));
141             Blockly.Python.domToWorkspace(Blockly.demoWorkspace,xml);
142         }
143         else{
144           alart("nothing to restore");
145         }
146       }
147     }
148
```

```
149
150
151
152
153
154     //run code and give a post httprequest to the server
155     //then server will get the value and make decissions
156     function runCode(){
157           Blockly.Python.INFINITE_LOOP_TRAP = null;
158           //convert blockly logics into python code
159           var code = Blockly.Python.workspaceToCode(demoWorkspace);
160           //var code = String(code);
161           //var fields = code.split(':');
162           //var logic_section = fields[0];
163           //var function_call = fields[1];
164           $.ajax({
165             type: 'POST',
166             url: "/getMotorStatus/",
167             headers:{
168               "X-CSRFToken": $("input[name=csrfmiddlewaretoken]").val()
169             },
170             data: {'code': code},
171             success: function(response) {
172                 console.log(response.code);
173             }
174           });
175     }
176 //weater block start
177 // Blockly.Blocks['weather'] = {
178 //   init: function() {
179 //     this.appendValueInput("to_input")
180 //         .setCheck(null)
181 //         .appendField("Weather info");
182 //     this.setOutput(true, null);
```

```
//      this.setColour(230);
//   this.setTooltip("");
//   this.setHelpUrl("");
//    }
// };
//weather block end

//motor block start


Blockly.Blocks['do_it'] = {
  init: function() {
    this.appendValueInput("NAME")
        .setCheck(null)
        .appendField("Do");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};

    Blockly.Blocks['motor-on'] = {
      init: function() {
        this.appendDummyInput()
        .appendField(new Blockly.FieldLabel('motor on'));//label
        //this.setOutput(true, 'String');
        this.setPreviousStatement(true, 'Action');
        this.setColour(60);
      }
    }
    Blockly.Blocks['motor-off'] = {
      init: function() {
```

```
217         this.appendDummyInput()
218         .appendField(new Blockly.FieldLabel('motor off'));
219         //this.setOutput(true, 'String');
220         this.setPreviousStatement(true, 'Action');
221         this.setColour(20);
222       }
223     }
224 //motor block end
225 //lower sensor block start
226 Blockly.Blocks['l-sensor'] = {
227       init: function() {
228         this.appendDummyInput()
229         .appendField(new Blockly.FieldLabel('lower sensor'));
230         this.setOutput(true, 'Number');
231         //this.setPreviousStatement(true, 'Action');
232         this.setColour(20);
233       }
234     }
235 //lower sensor block end
236
237
238
239 //start print_content
240
241 Blockly.Blocks['print_content'] = {
242   init: function() {
243     this.appendValueInput("to_print")
244         .setCheck(null)
245         .appendField(new Blockly.FieldTextInput("Print value"), "NAME");
246     this.setColour(230);
247  this.setTooltip("");
248  this.setHelpUrl("");
249   }
250 };
```

```
251
252
253
254
255 Blockly.Blocks['value_to_print'] = {
256   init: function() {
257     this.appendValueInput("to_input")
258         .setCheck(null)
259         .appendField("pass");
260     this.setOutput(true, null);
261     this.setColour(230);
262  this.setTooltip("");
263  this.setHelpUrl("");
264   }
265 };
266
267
268
269
270 //This is here to include all blocks Ui
271     {% include "blocks.js" %}
272
273
274   </script>
275
276 </body>
277 </html>
```

Block Generator.py

```
1
2
3 #This file include All Block Generator Functions
4 from .models import *
```

```
import string
import random
from django.http import HttpResponse
from django.shortcuts import render, redirect

#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)
    f.close()

#return Type of Block Ui you want
def GetBlockUI(BlockName, BlockType):
    # if BlockType == 'sensor':
    #     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"
    #     return BlockUi
    if BlockType=='sensor' or 'actuator':
        BlockUi = """
            Blockly.Blocks['"""+BlockName+""" '] = {
            init: function() {
            this.appendValueInput("to_input")
            .setCheck(null)
            .appendField('"""+BlockName+""" ');
            this.setOutput(true, null);
            this.setColour(230);
            this.setTooltip("");
            this.setHelpUrl("");
      }
    };
        """

    return BlockUi

```

```
#Greturn type of block js you want
def GetBlockJs(BlockName, BlockType):
    # if BlockType == 'sensor':
    #     BlockJs = "Blockly.Python["+"'"+BlockName+"'"+"] = function(Block)
    {"+'\n'+'\t'+"var code = "+"'"+BlockName+'()'"+";"+'\n'+'\t'+"return [code
    , Blockly.Python.ORDER_ATOMIC];"+'\n'+"}"
    #     return BlockJs
    if BlockType=='sensor' or 'actuator':
        BlockJs = """
            Blockly.Python['"""+BlockName+""" '] = function(block) {
              var value_to_input = Blockly.Python.valueToCode(block, 'to_input
    ', Blockly.Python.ORDER_ATOMIC);
              // TODO: Assemble JavaScript into code variable.
              var code = '"""+BlockName+"""('+value_to_input+');
              // TODO: Change ORDER_NONE to the correct strength.
              return [code, Blockly.Python.ORDER_NONE];
            };
        """
    return BlockJs


def BlockJsDeclaration(BlockName):
    f= open("templetes/blocks_js_declaration.js","a+")
    f.write('\n\n<script src="static/js/generators/codegenerator-'+BlockName+'
    .js"></script>')
    f.close()
```

Automatic writer.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://127.0.0.1:8000/api/actuators/'

    data = {

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

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

    #Print Response
    print(response.text)

    ""
    file.write(code)

    file.close()


def sensor_function_writter(function_name,tag):
    file = open("test_file.py","a+")

    code = """
def """+function_name+"""():
    empty_list=[]
    sensor_tag = """+tag+"""
    data = requests.get("http://127.0.0.1:8000/api/lowersensors/").json()
    for i in data:
        if i['topic']==sensor_tag:
```

```
            empty_list.append(i)

    data = empty_list[-1]
    print(data)

    data = int(data['value'])
    return(data)
"""

    file.write(code)

    file.close()
```

Data processing.py

```

import requests
import json

def weather_info(city):
    api_key = "bd27419d66c8678613e978ca561ad3f7"
    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


```

```
19
20
21
22
23  def action_motor(motor_status):
24      url = 'http://127.0.0.1:8000/api/actuators/9/'
25
26      data = {
27
28          "topic": "mc101",
29          "value": motor_status,
30          "time": "2019-01-24T13:35:24.246226Z",
31          "name": "1"
32              }
33
34
35
36      #Call REST API
37      response = requests.put(url, data=data)
38
39      #Print Response
40      print(response.text)
41
42
43  def get_lowersensor():
44      empty_list=[]
45      sensor_tag = 'soil_moisture_sensor::2.33::21.22::sayeed'
46      data = requests.get("http://127.0.0.1:8000/api/lowersensors/").json()
47      for i in data:
48          if i['topic']==sensor_tag:
49              empty_list.append(i)
50
51      data = empty_list[-1]
52      print(data)
```

```
53
54      data = int(data['value'])
55      return(data)
56
57
58
59
60  def value_to_print(text):
61      print(text+" pass function")
62      return text
63
64
65  def print_content(what_to_print):
66      what_to_print=what_to_print+" print_content"
67      print(what_to_print)
68
69  # important ends
70
71
72  def Labtest_2_33_1_22_LAB(motor_status):
73      url = 'http://127.0.0.1:8000/api/actuators/'
74
75      data = {
76
77          "topic": 'Labtest_2.33_1.22_LAB',
78          "value": motor_status,
79          "time": "2019-01-24T13:35:24.246226Z",
80          "name": "1"
81          }
82
83
84
85      #Call REST API
86      response = requests.post(url, data=data)
```

```
87
88      #Print Response
89      return response
90
91
92  def soil_moisture_sensor_3_22_2_111_User():
93      empty_list=[]
94      sensor_tag = "soil_moisture_sensor_3.22_2.111_User"
95      data = requests.get("http://127.0.0.1:8000/api/lowersensors/").json()
96      for i in data:
97          if i['topic']==sensor_tag:
98              empty_list.append(i)
99
100     data = empty_list[-1]
101     print(data)
102
103     data = int(data['value'])
104     return(data)
105
106
107 def Motor_actuator_longi_actuator_lati_Sayeed(motor_status):
108     url = 'http://192.168.1.110:8000/api/actuators/'
109
110     data = {
111
112         "topic": 'Motor_actuator_longi_actuator_lati_Sayeed',
113         "value": motor_status,
114         "time": "2019-01-24T13:35:24.246226Z",
115         "name": "1"
116         }
117
118
119
120     #Call REST API
```

```
    response = requests.post(url, data=data)

    #Print Response
    print(response.text)


def device_name_actuator_longi_actuator_lati_owner_name(motor_status):
    url = 'http://127.0.0.1:8000/api/actuators/'

    data = {

        "topic": 'device_name_actuator_longi_actuator_lati_owner_name',
        "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)
```

Models.py

```


from django.db import models
from datetime import datetime
from django.utils import timezone
from django.db.models import signals
import requests
```

```
8
9
10
11
12 # Create your models here.
13 class Actuator(models.Model):
14     topic = models.CharField(max_length=100, null = True)
15     value = models.CharField(max_length=100, null = True)
16     time = models.DateTimeField(auto_now_add=True,null = True)
17     name = models.CharField(max_length=60, null = True)
18
19     def __str__(self):
20         return self.name
21
22     def save(self, *args, **kwargs):
23
24         if not self.id:
25             self.time = timezone.now()
26         return super(Actuator, self).save(*args, **kwargs)
27
28
29 class LowerSensor(models.Model):
30     topic = models.CharField(max_length=100, null = True)
31     value = models.CharField(max_length=100, null = True)
32     time = models.DateTimeField(default=datetime.now, null=True)
33     name = models.CharField(max_length=60, null = True)
34
35     def __str__(self):
36         return self.name
37     def save(self, *args, **kwargs):
38
39         if not self.id:
40             self.time = timezone.now()
41         return super(LowerSensor, self).save(*args, **kwargs)
```

```
42
43
44 class Service_registry(models.Model):
45     name = models.CharField(max_length=100,unique=True)
46     #name_id = models.CharField(max_length=100)
47     #api_link = models.CharField(max_length=200)
48     service_type = models.CharField(max_length=50)
49
50     def __str__(self):
51         return self.name
```

Serilizers.py

```
1 Serilizers.py
2
3
4 from rest_framework import serializers
5 from .models import *
6
7 class ActuatorSerializer(serializers.ModelSerializer):
8     class Meta:
9         model = Actuator
10        fields = ('topic','value', 'time', 'name')
11
12 class LowerSensorSerializer(serializers.ModelSerializer):
13     class Meta:
14         model = LowerSensor
15         fields = ('topic','value', 'time', 'name')
16
17
18 #service registry serializer
19 class ServiceRegistrySerializer(serializers.ModelSerializer):
20     class Meta:
21         model = Service_registry
```

```
22          fields = ('id','name','service_type')
```

Views.py

```
1
2  from django.shortcuts import render
3  from rest_framework import viewsets, generics
4  from rest_framework.views import APIView
5  from rest_framework.filters import (
6      SearchFilter,
7      OrderingFilter)
8  from .serializers import *
9  from .models import *
10 from .data_processing_library1 import *
11
12 from django.db.models import Q
13 from django.http import HttpResponse, JsonResponse
14 from django.views.decorators.csrf import csrf_protect
15 import requests
16 from . import automatic_writter
17 from . import BlockGenerator
18
19 from rest_framework.response import Response
20
21
22 # Create your views here.
23 def home(request):
24     action_motor("0")
25     get_lowersensor()
26     all_service = requests.get("http://127.0.0.1:8000/api/serviceregistry/").
       json()
27     #print(all_service)
28     last_service = all_service[-1]
29
```

```
30      print(last_service['id'])
31      read_file = open('last-id.txt','r')
32      a = read_file.read()
33      read_file.close()
34      name_id = last_service['name']
35      sensor_tag = name_id
36      name_id = name_id.replace(".","_")
37
38      # print(a)
39
40      # checking for new entry in Service_registry
41      if int(a) < last_service['id']:
42          print('a')
43          write_file = open('last-id.txt','w+')
44          write_file.write(str(last_service['id']))
45          write_file.close()
46
47          if last_service['service_type'] == 'sensor':
48              automatic_writter.sensor_function_writter(name_id,sensor_tag)
49              BlockGenerator.BlockGenerator(name_id,last_service['service_type'
    ])
50
51          elif last_service['service_type'] == 'actuator':
52              automatic_writter.actuator_function_writter(name_id,sensor_tag)
53              BlockGenerator.BlockGenerator(name_id,last_service['service_type'
    ])
54
55  # posting to actuator model
56              url = 'http://127.0.0.1:8000/api/actuators/'
57
58              data = {
59
60                  "topic": name_id,
61                  "value": "null",
```

```
                "time": "null",
                "name": "1"
                }


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


    return render(request, 'index.html')


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

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




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


class ActuatorListApiView(generics.ListAPIView):
    serializer_class= ActuatorSerializer

```

```
96      def get_queryset(self):
97          qs=Actuator.objects.all()
98          query = self.request.GET.get("q")
99
100         if query is not None:
101             qs= qs.filter(
102                 Q(topic__icontains=query)
103                 ).distinct().order_by('-time')[:1]
104
105         return qs
106
107
108 def getMotorStatus(request):
109     #csrfContext = RequestContext(code)
110     if request.is_ajax():
111         code = request.POST['code']
112         #function_call = request.POST['function_call']
113         #motor_status = get_action_motor_status(function_call)
114         print(code)
115         exec(code)
116     else:
117         return HttpResponse('Use ajax format!')
118
119     return JsonResponse({'code': code })
```