DISTRIBUTED AGENTS ARCHITECTURE USING INTERNET OF THINGS

Submitted By

MD. Junaid Fahad
ID 2011-2-60-014

Mohammad Shafiullah
ID 2011-2-60-015

Supervised By

K.M. Imtiaz Ud-din
Senior Lecturer
Department of Computer Science and Engineering
East West University

A thesis Submitted in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science & Engineering

At the

EAST WEST UNIVERSITY
DHAKA, BANGLADESH

JANUARY 2016

Declaration

This is certified that this project is an orginal work and is done by us. Neither it nor part of it has been submitted elsewhere for the requirment of any degree or diploma or for any other purposes.

_____

Md. Junaid Fahad

ID: 2011-2-60-014

Department of

Computer Science and Engineering

East West University

_____

Mohammad Shafiullah

ID: 2011-2-60-015

Department of

Computer Science and Engineering

East West University

Letter of Acceptance

This project is entitled "**Distributed Agent Architecture Using Internet of Things**" submitted by Md. Junaid Fahad (2011-2-60-014) and Mohammad Shafiullah (2011-2-60-015) to the department of Computer Science & Engineering, East West University, Dhaka, Bangladesh is accepted by the department in partial fulfillment of requirments for the degree of Bachelor of Science in Computer Science & Engineering on January, 2016.

Board of Examiners

Supervisor

_____

K.M. Imtiaz Uddin

Senior Lecturer

Department of Computer Science and Engineering

East West University

Chairperson

_____

Dr. Shamim Hasnat Ripon

Associate Professor and Chairperson

Department of Computer Science and Engineering

East West University

Abstract

Many mobile, hardware and windows system is frequently using Internet of things at present. The number of user and the demand of automated system is also increasing. Many researcher working on IoT and they are trying to create a new era of technology. Some statistics shows that in 2020 the connected device over the internet will be Fifty Billion which is more than six times larger than the total population of world. So the demand of system automation will be largely depended in internet. The concept of Distributed agent using IoT is to communicate between people to people, machines to people, machine to machine. The basic concept of this research project is to isolate sensor, actuator, and logic although they are working together which is basically known as distributed smart agent. This research project also described that a sensor is not dependent on actuator as well as logic, it can also work independently and vice versa. Because of the increasing number of IoT device this paper will contribute to give basic idea of how a distributed smart agent can communicate with another smart agent.

Acknowledgement

First of all we would like to thank Almighty Allah for giving us the strength and patience.

This document presents my Bachelor of Science Thesis “Distributed Agent architecture using Internet of Things”. This project took place from September 2015 to January of 2016. This was one of the most difficult projects I have worked on during my four years of studying Computer Science, due to the ambiguity and high level of abstraction of the topics that play a role in this thesis. Thankfully, I received help from various people to whom I wish to express my gratitude towards in this section.

Our sincere gratitude goes to our parents for their all kindness and encouragement during our studies, without them this thesis would have been very hard to finish.

We would like to thank our supervisor K.M. Imtiaz Ud-din for provide his utmost direction, constant helpful support and feedbacks regarding the structure and motivation for this thesis.

We also thank the researchers for their works that help us to learn and implement Distributed Agents Architecture using Internet of Things.

Last but not the least, we express gratitude to our friends, seniors and juniors for providing effective suggestions and supporting us.

Md. Junaid Fahad & Mohammad Shafiullah

Contents

List of Tables

List of Figure

Abbreviation & Acronyms

- **DAI –** Distributed Artificial Intelligence
- **IoT-** Internet of Things
- **IDE-** integrated development environment
- **.APK-** Android Package Kit
- **MQTT-** MQ Telemetry Transport
- **HTTP-** Hypertext Transfer Protocol
- **API-** Application Programming Interface
- **IBM-** International Business Machines
- **TCP/IP-** Transmission Control Protocol/Internet Protocol
- **MANET-** Mobile Ad Hoc Network
- **RFID-** Radio Frequency Identification
- **VANET-** Vehicle Ad Hoc Network
- **GSM-** Global System for Mobile communication
- **WiMAX-** Worldwide Interoperability for Microwave Access
- **DARPA-** Defense Advanced Research Projects Agency
- **CIM-** City Information Model
- **LED-** Light Emitting Diode
- **NAVs-** Nano Air Vehicles
- **NFC-** Near Field Communication

Chapter 1

Introduction

The Internet of Things is a new concept of our everyday objects. We use this from industrial machines to wearable devices, using built-in sensors to gather data and take action on that data across a network. The sensor and actuator can be setup in different place but they are working together over an internet network. So it's a building that uses sensors to automatically adjust heating and lighting or production equipment alerting maintenance administration to detect failure and problem. Simply, the Internet of Things is the future of technology that can make our lives more efficient and reliable.

This chapter introduces the distributed artificial intelligence with internet of things. In, Section 1.1 describes the motivation behind this system. Section 1.2 objectifies the proposed system. Section 1.3 describes a short overview of our proposed system. Section 1.4 presents the contribution of proposed system in distributed Artificial Intelligence with Internet of Things. Section 1.5 gives a outline of the rest of the document.

1.1 Motivation

One of the general goals of Internet of things is to design a architecture that can easily transfer data over an internet network. In Most of the Artificial intelligence we see that they are tightly coupled between sensor, actuator, and the processing system. So if any of them become ruined or destroy than the whole system will become priceless. Because the system totally depended on each of it's parts for sensing data. If a sensor facing problem to collect a sample from the environment the whole system wouldn't work well & it will continuously generate wrong output.

So by thinking this we become motivated for working on a system which is not like tightly coupled system like before. We don't want to face problem when we need. So this problem can be solve by taken help of Distributed Agents Architecture. Use of distributed artificial intelligence is increasing day by day. Every network device Connecting our cars, homes, gadgets and machines to the Internet creates a network of connected objects called the Internet of Things. How can the data transfer to and from these objects be understood to benefit our organization, improve our daily routine or stores many social biggest challenges. There are various type application related with IoT is proposed and people will get help from these application. IoT can save people's life, help to control traffic system etc. In details,

Many people have wearable devices to help monitor exercise, sleep and other health habits and these items are help to how IoT impacts health care. Patient monitoring devices, electronic records and other smart accessories can help save lives.

While cars aren't at the point of driving themselves, they're undoubtedly more technologically advanced than ever. The IoT also impacts transportation on a larger scale: delivery companies can track their vehicle using GPS solutions. Also many road sensor can update the news of traffic system and communicate through GPS

Industries is one of the field that benefits from IoT the most. Data-collecting sensors embedded in factory machinery or warehouse shelves can communicate problems or track resources in real time, making it easy to work more efficiently and keep costs down.

Smart meters not only collect data automatically, they make it possible to apply analytics that can track and manage energy use. Like, sensors in devices such as windmills can track data and use predictive modeling to schedule downtime for more efficient energy use.

IoT is a combination of programming, wireless sensor networking, artificial intelligence, web application. So this is a good chance to implement all of them in a single application.

1.2 Objectives

The Internet of Things (IoT) is aimed at enabling the interconnection and integration of the physical world and the cyber space. It represents the trend of future networking, and leads the third wave of

the IT industry revolution. The objectives of this work is to achieve comprehensive performance to control multiple android device (e.g. Smart phone, Wearable watch) for machine to machine communication over the internet. Day by day the uses of internet is elaborating. Many new devices is connected daily with internet. But they are tightly depended on fixed chips. The device can only input data by their built in sensor. It doesn't matter for an agent to be tightly coupled is that sensor, actuator and logic don't need to be part of a single agent. So this work can easily handle other devices sensor and actuator for taking input and giving output.

1.3 Overview of the proposed System

Our system is IoT application with using android devices. The system can control another android device for change background, blink flash light of device, turn on the camera and capture an image than turn off using Distributed Agents Architecture. This is a very interesting system which is based on MQTT protocol. By using that protocol we can transfer data from server and server will again transfer that data to another android device. Both devices will be control by a android application. When one device is moving to different coordinate than the device send the X,Y,Z-axes through MQTT protocol to the server and server process the data with implemented logic. Logic can be stored in devices but in this system we stored that in server. When server also transfer that data to another android device than some interesting things will happened only for using IoT idea.

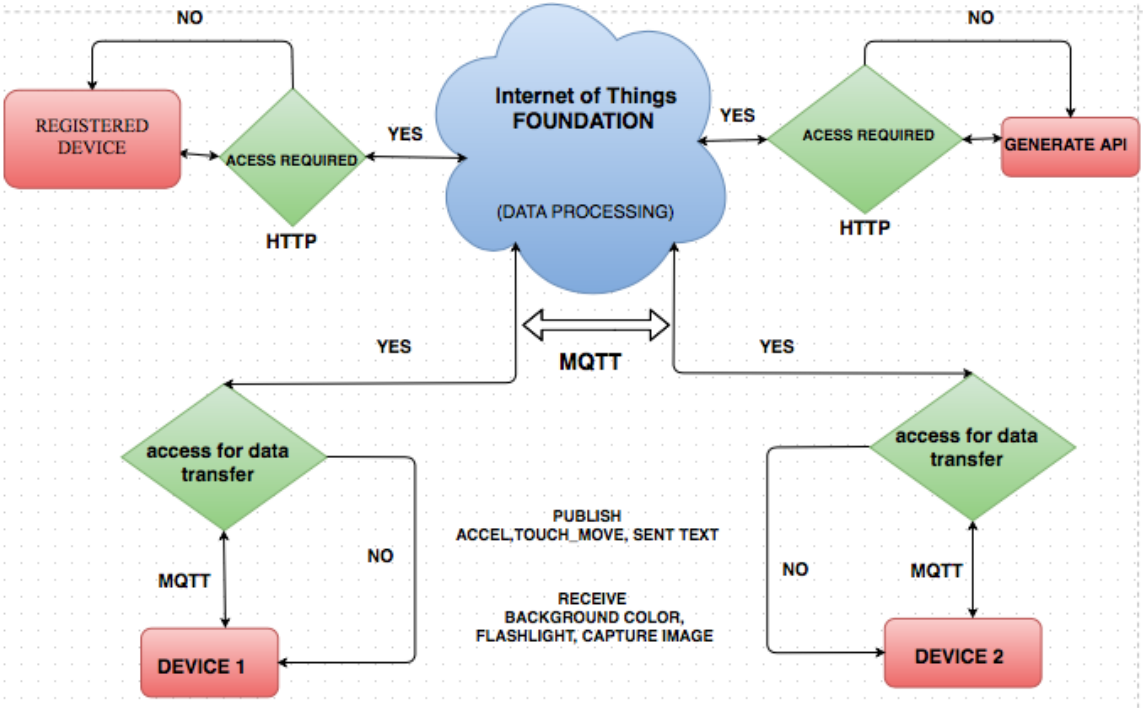

Figure 1.1: Overview of Distributed Agents Architecture using Block Diagram

1.4 Outline of the Thesis

Chapter 2 reviews related work in Internet of Things. Many technologies is implement about IoT. Apple, Cisco, Google, IBM many different major companies are working on machine to machine communication. It also describe about big data and wireless network classification in this proposed system.

Chapter 3 describes the proposed system mechanism for the proposed internet of things with DAI.

Chapter 4 discusses the experimental setup and implementations of the system. It also analyze the data of the proposed system.

Chapter 5 it concludes the thesis work by mentioning the few limitations and future related work of this system.

Chapter 2

Literature Review

The development of the Internet of Things [IoT] has been primarily driven by needs of large corporations that stand to benefit greatly from the foresight and predictability afforded by the ability to follow all objects through the commodity chains in which they are embedded [1]. The ability to code and track objects has allowed companies to become more efficient, speed up processes, reduce error, prevent theft, and incorporate complex and flexible organizational systems through IoT [2].

This chapter discuses previous related works on IoT in section 2.1, it also briefly describe the technology we need to build up for IoT in section 2.2.

2.1 Related Works

The Internet of Things (IoT) – a term need to know – is comprised of all these connected devices and continues to grow at an exponential pace. This year, the IT research firm Gartner estimates there are close to five billion physical objects – from laptop computers to tiny sensors attached to a pallet of goods – capable of gathering and sharing electronic information. By 2020, it predicts the tally will reach 25 billion. Some say it will be even higher.

"*The IoT is getting bigger and bigger every year but we are still in the very early stages of this technology,*" - Jason Aiginitis.

Google IoT based Android OS BRILLO

Brillo is an IoT platform with three elements: Android-based embedded OS, core platform services, and a developer kit.

The Android OS is an open source software that will get minor updates once in six weeks with the long-term support (LTS) build updated once in every six months. Brillo can be built from source code to target ARM, Intel, and MIPS architecture. The board support package (BSP) adapts to specific boards running one of the supported architectures. The OS can run on low-end devices with at least 128MB of storage and 32MB of RAM. Google is working with hardware partners to certify the boards that are compatible with Brillo. These boards are verified and tested to run the current and future versions of the OS. Each board comes with extensive documentation and samples. This approach ensures that Google delivers better developer experience while keeping the devices up-to-date.

Core services of the platform include Weave, which helps devices to securely connect to the network. It enables users to connect the devices to the mobile and desktops. Weave-enabled devices can seamlessly talk to each other. Metrics component is a part of core services to collect usage data from devices based on the user permission. The data can be viewed and analyzed in the console to understand the usage patterns of consumers. Crash reports can also be analyzed to debug remote devices deployed in the field. Administrators can push patches and the latest version of the software through over-the-air (OTA) updates.

2.2. Technologies

The Internet of Things [37] was initially inspired by members of the RFID community, who referred to the possibility of discovering information about a tagged object by browsing an internet address or database entry that corresponds to a particular RFID or Near Field Communication [38] technologies. Some useful technologies of IoT would be,

2.2.1 Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) is a system that transmits the identity of an object or person wirelessly using radio waves in the form of a serial number [39]. First use of RFID device was happened in 2nd world war in Britain and it is used for Identify of Friend or Foe in 1948. Later RFID technology is founded at Auto-ID center in MIT in the year 1999. RFID technology plays an important role in IoT for solving identification issues of objects around us in a cost effective manner [40]. The technology is classified into three categories based on the method of power supply provision in Tags: Active RFID, Passive RFID and Semi Passive RFID. The main components of

RFID are tag, reader, antenna, access controller, software and server. It is more reliable, efficient, secured, inexpensive and accurate. RFID has an extensive range of wireless applications such as distribution, tracing, patient monitoring, military apps etc. [41].

2.2.2 Internet Protocol (IP)

Internet Protocol (IP) is the primary network protocol used on the Internet, developed in 1970s. IP is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. The two versions of Internet Protocol (IP) are in use: IPv4 and IPv6. Each version defines an IP address differently. Because of its prevalence, the generic term IP address typically still refers to the addresses defined by IPv4. There are five classes of available IP ranges in IPv4: Class A, Class B, Class C, Class D and Class E, while only A, B, and C are commonly used. The actual protocol provides for 4.3 billion IPv4 addresses while the IPv6 will significantly augment the availability to 85,000 trillion addresses [42]. IPv6 is the 21st century Internet Protocol. This supports around for 2128 addresses.

2.2.3 Wireless Fidelity (Wi-Fi)

Wireless Fidelity (Wi-Fi) is a networking technology that allows computers and other devices to communicate over a wireless signal. Vic Hayes has been named as father of Wireless Fidelity. The precursor to Wi-Fi was in- vented in 1991 by NCR Corporation in Nieuwege in the Netherland. The first wireless products were brought on the market under the name WaveLAN with speeds of 1 Mbps to 2 Mbps. Today, there are nearly pervasive Wi-Fi that delivers the high speed Wireless Local Area Network (WLAN) connectivity to millions of offices, homes, and public locations such as hotels, cafes, and airports. The integration of Wi-Fi into notebooks, handhelds and Consumer Electronics (CE) devices has accelerated the adoption of Wi-Fi to the point where it is nearly a default in these devices [43]. Technology contains any type of WLAN product support any of the IEEE 802.11 together with dual-band, 802.11a, 802.11b, 802.11g and 802.11n. Nowadays entire cities are becoming Wi-Fi corridors through wireless APs.

2.2.4 ZigBee

ZigBee is one of the protocols developed for enhancing the features of wireless sensor networks.

ZigBee technology is created by the ZigBee Alliance which is founded in the year 2001. Characteristics of ZigBee are low cost, low data rate, relatively short transmission range, scalability, reliability, flexible protocol design. It is a low power wireless network protocol based on the IEEE 802.15.4 standard [44]. ZigBee has range of around 100 meters and a bandwidth of 250 kbps and the topologies that it works are star, cluster tree and mesh. It is widely used in home automation, digital agriculture, industrial controls, medical monitoring &power systems.

2.2.5 Near Filed Communication (NFC)

Near Field Communication (NFC) is a set of short-range wireless technology at 13.56 MHz, typically requiring a distance of 4 cm. NFC technology makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. Allows intuitive initialization of wireless networks and NFC is complementary to Bluetooth and 802.11 with their long distance capabilities at a distance circa up to 10 cm. It also works in dirty environment, does not re- quire line of sight, easy and simple connection method. It is first developed by Philips and Sony companies. Da- ta exchange rate now days approximately 424 kbps. Power consumption during data reading in NFC is under 15ma.

2.2.6 Wireless Sensor Networks (WSN)

A WSN is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations (Wikipedia). Formed by hundreds or thousands of nodes that communicate with each other and pass data along from one to another. A wireless sensor network is an important element in IoT paradigm. Sensor nodes may not have global ID because of the large amount of overhead and large number of sensors. WSN based on IoT has received remarkable attention in many areas, such as military, homeland security, healthcare, precision agriculture monitoring, manufacturing, habitat monitoring, forest fire and flood detection and so on [45]. Sensors mounted to a patient's body are monitoring the responses to the medication. So that doctors can measure the effects of the medicines using internet of things [46].

Chapter 3

This chapter describe the proposed Distributed Artificial Intelligence with Internet of Things. In section 3.1 we have presented the architecture of distributed AI. In section 3.2 we have described our mechanism of our proposed system.

3.1 Proposed System Architecture

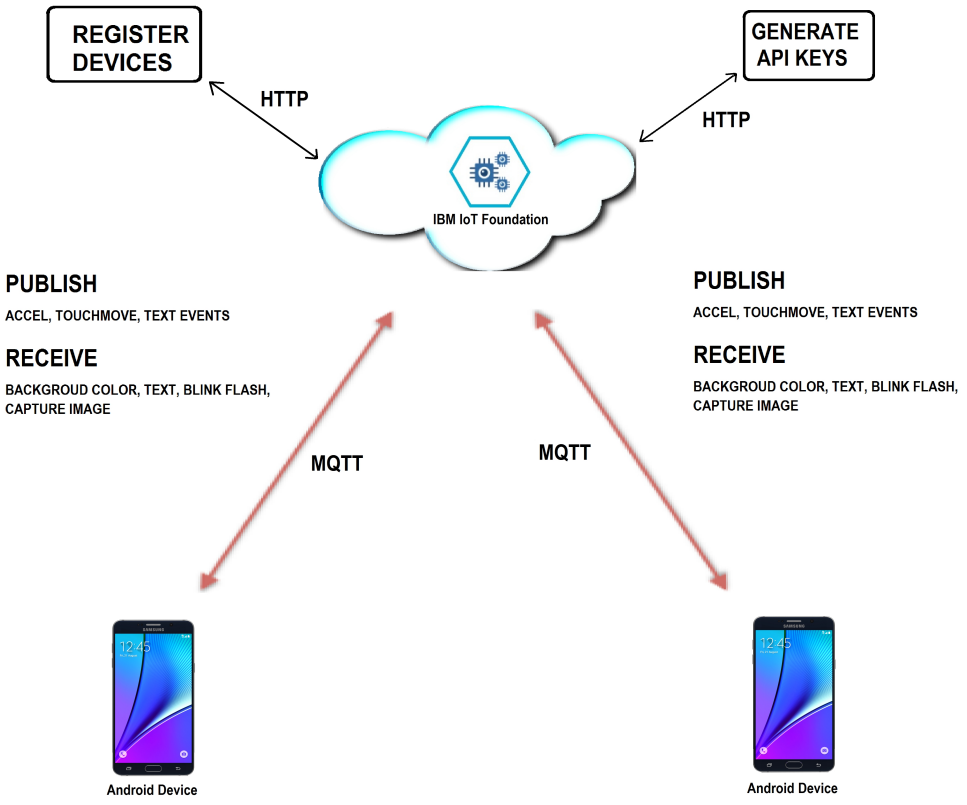

Figure 3.1 : Architecture of Proposed System

3.2 Mechanism

Internet of things is a system where we need to follow some rules and regulation. Where we have to isolate the sensor actuator and logic. We are working on distributed Artificial intelligence what means the sensor, actuator and logic can be stored in different place but they can work together simultaneously. This paper proposed a design where a sensor can act as sensor and same device can work as actuator. Not necessary the logic will be stored in different place is that different server. A logic can work from both sensor and actuator. We can control a mobile device with other device by using this distributed system. Like we can Change the background color, blink the flashlights, capture image.

3.2.1 Application Installation:

Our proposed application is able to work in android operating system enabled device. Any android OS device like mobile, tablet, wearable, Computer etc is able to perform the action. To install the application we used eclipse as our IDE.

And some required permission for the applications are

- Internet
- Access network state
- Write external storage
- Access WIFI state
- Wake lock
- Flashlight
- Camera

Without those permission in android manifest file the application will not deploy. After deploying the application an .APK file will be generated in bin folder. Copy these file into both your sensor and actuator device. Install the application in the device. After successful installation an input field window will appear on the device.

3.2.2 Device Registration In the server

We will use HTTP protocol for to register device and API key generate. To publish and receive our data which will produce our sensor and receive actuator a global server is mandatory. It is possible to hosting an own server for to publish and receive data but for simplicity we used IBM Blue-mix server where we are able to add different types of service, create application and add multiple device. At first create an application to the server. Add Internet of things service to the application and finally create multiple device inside of internet of things service with their own device id and authentication token.

3.2.3 Publishing Data Into Server

In this phase we will use MQTT protocol for to publish data in the IBM Bluemix server. Now from the android device login with device id and authentication token. The device will publish device accelerometer, touch sensor value and text to the IBM Bluemix server with lower than 1 second latency rate using MQTT protocol. It will provide real time data publishing facility.

3.3 Protocol Used

Protocol is actually a rules and regulation of a system. A system can be used to run by a single protocol or multiple combined protocol. Protocol will help to Generate keys, Registered device and publish or restore data, transfer data from one device to server than other device. The Key points of the protocol are:

- MQTT Protocol
- HTTTP Protocol

3.3.1 MQTT Protocol

MQTT[1] (formerly MQ Telemetry Transport) is a publish-subscribe based "light weight" messaging protocol for use on top of the . It is designed for connections with remote locations where a "small

code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message. Andy Stanford-Clark and Arlen Nipper of Cirrus Link Solutions authored the first version of the protocol in 1999.

MQTT methods:

MQTT defines methods (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified resource [3]. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

Connect

Waits for a connection to be established with the server.

Disconnect

Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect.

Subscribe

Waits for completion of the Subscribe or UnSubscribe method.

UnSubscribe

Requests the server unsubscribe the client from one or more topics.

Publish

Returns immediately to the application thread after passing the request to the MQTT client.

3.3.2 HTTP Protocol

The Hypertext Transfer Protocol (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems. **HTTP** is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.

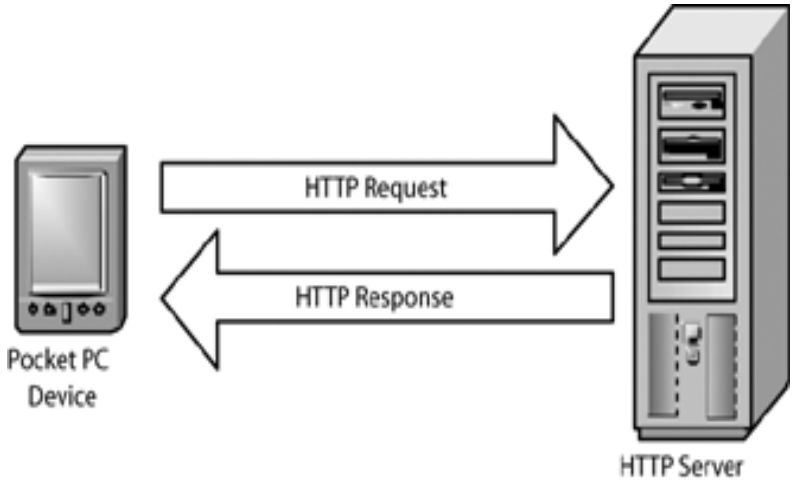

Figure 3.2 : HTTP Protocol for establish a connection.

In Our Application we use HTTP Protocol for send a request for registered the device with server and the server response with device name and authentication token. Also every device have to generate a API key through HTTP Protocol.

3.4 Server side application:

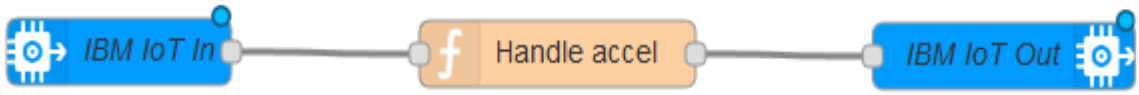

Figure 3.3: IBM IoT in (Left Side) is sensor, Handle accel(Middle) is the logic, IBM IoT Out(Right Side) is the actuator.

It is not mandatory to write logic into server, but in this project we will write our logic in server. The above figure shows the architecture of server side logic implementation. **IBM IoT in** is the device from where the sensor value or data will come. **Handle Accel** process the received data from sensor and it sends the processed data to IBM IoT out. **IBM IoT Out** will finally connect to the actuator. From the actuator we can show the output depend on the logic. In this application Handle accel function wrote using Node JS. IBM IoT sends JSON formatted data to Handle accel. Handle accel process the JSON formatted data. IBM IoT Out also sends JSON formatted data to actuator.

Chapter 4

4.1 Experimental Setup

4.1.1 IBM Bluemix

IBM Bluemix a cloud platform as a service (Paas) developed by IBM. It supports several programming languages like Java, Node.js, Go, PHP, Python, Ruby Sinatra, Ruby on Rails and services. Bluemix is based on Cloud Foundry open technology and runs on SoftLayer infrastructure. Bluemix provides access to a wide variety of services that can be incorporated into an application. Some of these services are delivered through Cloud Foundry. Others are delivered from IBM and third party vendors. New and enhanced services are added to the catalog often. Because Bluemix is based on Cloud Foundry, you can tap into a growing ecosystem of runtime frameworks and services. In addition to providing additional frameworks and services, Bluemix provides a dashboard for you to create, view, and manage your applications and services as well as monitor your application's resource usage. The Bluemix dashboard also provides the ability to manage organizations, spaces, and user access.

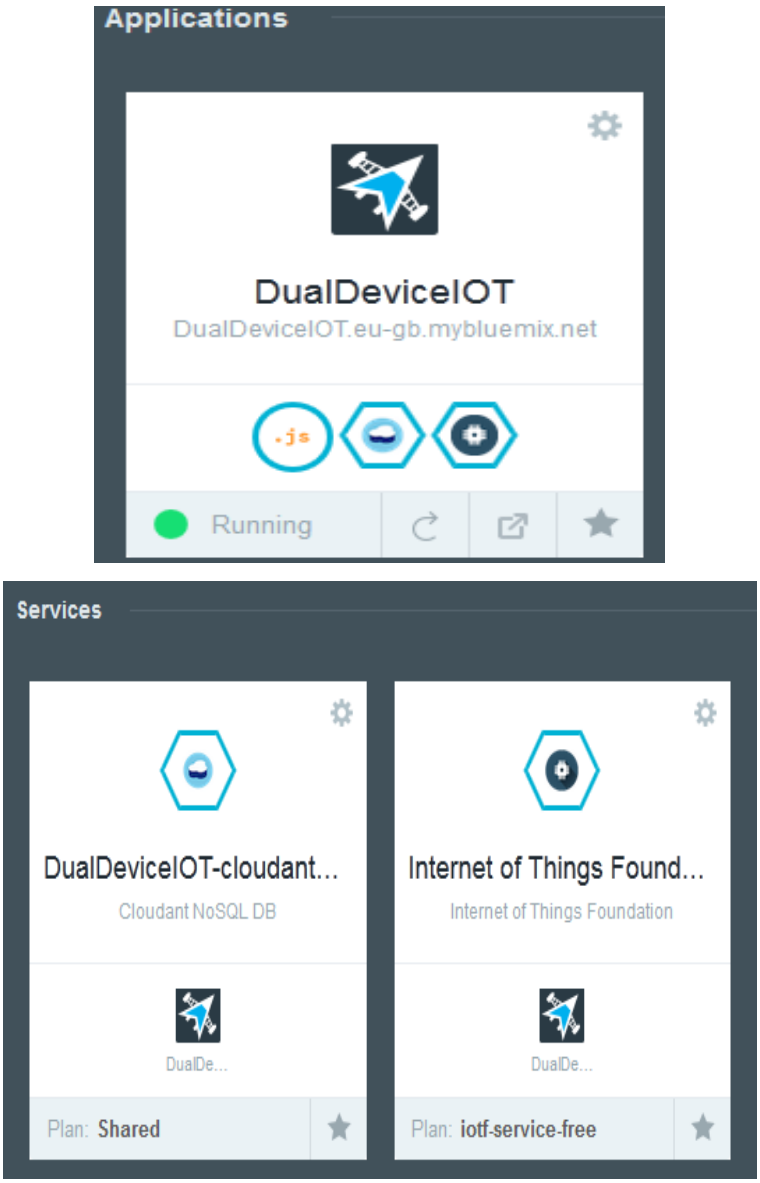

Figure 4.1: A single application with two services

4.1.1.1 Setup

For simplicity we will use IBM Bluemix [4] as our server side. In this site some one should must be registered for to run a single application. It is possible to run multiple application in the IBM Bluemix site. After registration create an app from the catalog menu. Add IoT service in the application. Register at least two android devices in the IoT service with their authentication token and device ID [5].

4.1.2 Core Programming Language and functions for device site Application

Java ME is used for the application implementation and the IDE used for to run the application is Eclipse. We created some basic functionality class using java languages. Some of this basic classes and their works are given below:

4.1.2.1 Main activity class:

Main activity class is used for to run the complete program with essential graphical user interface in mobile devices. This class is work as a launcher of the application. If we want to functioning the application effectively main activity class must be launched. This main activity class calls another important functional java class.

4.1.2.2 Login activity class:

Main activity class at first calls Login activity class. From the login activity class user have to input organization id, device id, and authorization token. The data of those input field will be sent to server for checking the validation of the user and the device.

4.1.2.3Device Sensor class:

Device sensor class is used for to sending the sensor data to the IBM Bluemix server. This class sensed the device accelerometer, screen touch data and text data. It will send data continuously to the server until user disconnect the device with server.

4.1.2.4 Message factory class:

This class is used for processing the data which sent from device sensor class. Because of IBM server will not allowed all formatted data this class basically process the data as JSON format and make it readable by the server.

4.1.2.5 MQTT Handler class

This is very important for the application because this class establish a connection between the device and the server. This class used for to sending the processed JSON formatted data to server and vice versa using MQTT protocol.

4.1.2.6 Message Conductor class

Message conductor class is used for to receive the data from the server. This class is very important because what will our actuator do completely depend on this class. Based on this class the actuator take the decision what will be next step of it.

4.1.2.7 IoT Starter Application class

Message conductor activity class calls this class because message conductor class only define what the actuator will do but how it will do defines on IoT Starter application class. So, this class is very important because for this class user is able to show the actual output by the actuator.

4.1.3 Logic setup in the server with node JS

For to actually do something with our device application we must have to build an application in the server side. We used node JS programming as server side logic application because node JS can easily work with JSON formatted data. There is a built in editor called Node Red in the IBM Bluemix server. For the simplicity we will use this editor. But it is possible to write node JS program in another editor and outside the Bluemix server. In the node red editor we will do the logic implementation for our sensor and actuator. The main logic is to play with z axis value received from sensor accelerometer data. If the z is positive we will send a message to actuator and if the value is negative send another message to the actuator.

4.1.4 Operating system (android)

For our device site application we used java language but this application is the core functionality. For to see the input and output we must have to define some layout which will be created using XML. Because of simultaneously working with each other this operating system is called android OS which is very user friendly and open source OS. Our application will be run reliably with this OS in any mobile device supports Java. So, the front end view and the backend logic all will work together with this OS.

4.1.5 Device Setup (Mobile/Smart watch)

Any device supports android OS is able to run the application and make it able to fully functioning. We will use two mobile device for the see the functionality of our application. Our purpose is to show that it is possible to work together all although they all are isolated from each other. So, two mobile device is good enough for to show the full functioning. The device side application need to deploy in the mobile device and the device must be connected to the internet for to functioning.

4.1.6 Sensor and actuator:

Sensor and actuator is very basic and important concept in AI. Without sensor or without actuator an agent function is incomplete. What we usually see is an agent architecture is like the image given below.

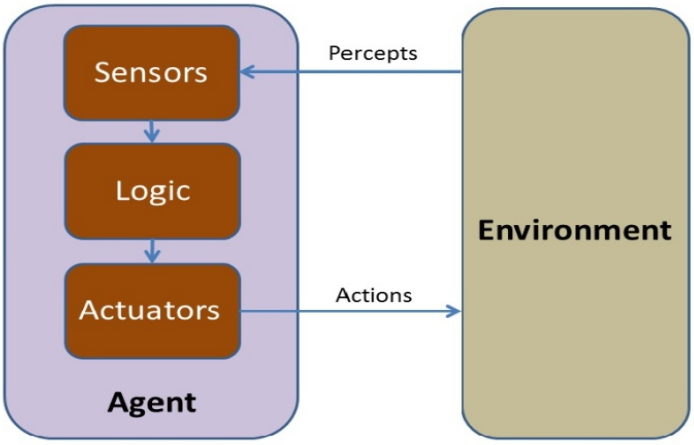

Figure 4.2: Simple Agent Architecture

From the architecture we can describe that an agent can sensed from the environment by the percept. After receiving the sensed data the logic have to be processed .Finally actuator act with processed data to the environment. The problem with that agent is that a single agent is receiving sensor data and acting by actuator from the corresponding sensed data. The problem is that a single agent must be have three things sensor, logic and actuators but with our proposed system it is possible to keep sensor in one place, logic in another place, and actuator is something else where .The architecture is given below.

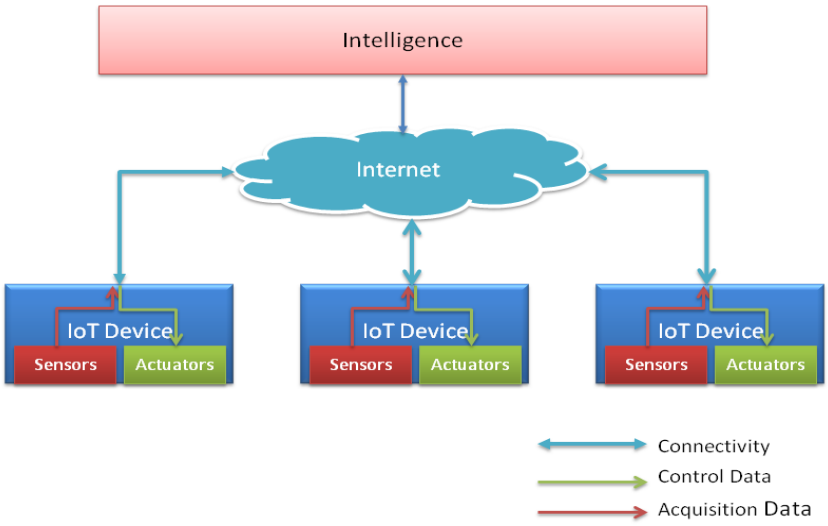

Figure 4.3: Smart IOT Architecture

As a sensor and as an actuator we used two mobile device both can work as a sensor and as an actuator and vice versa. We can assume that, one device is a sensor and another device is an actuator and vice versa. Here, with the architecture the good is that they will all work although they are isolated is that sensor is in one place and actuator is in another place through the internet.

4.2 Result and Discussion

In this section we described the performance of our application and input transfer to the server by sensor and the output received by actuator by some snapshot. First of all we are showing the accelerometer data what is generated by the sensor and publishing the data to the IBM Bluemix server by device side application.

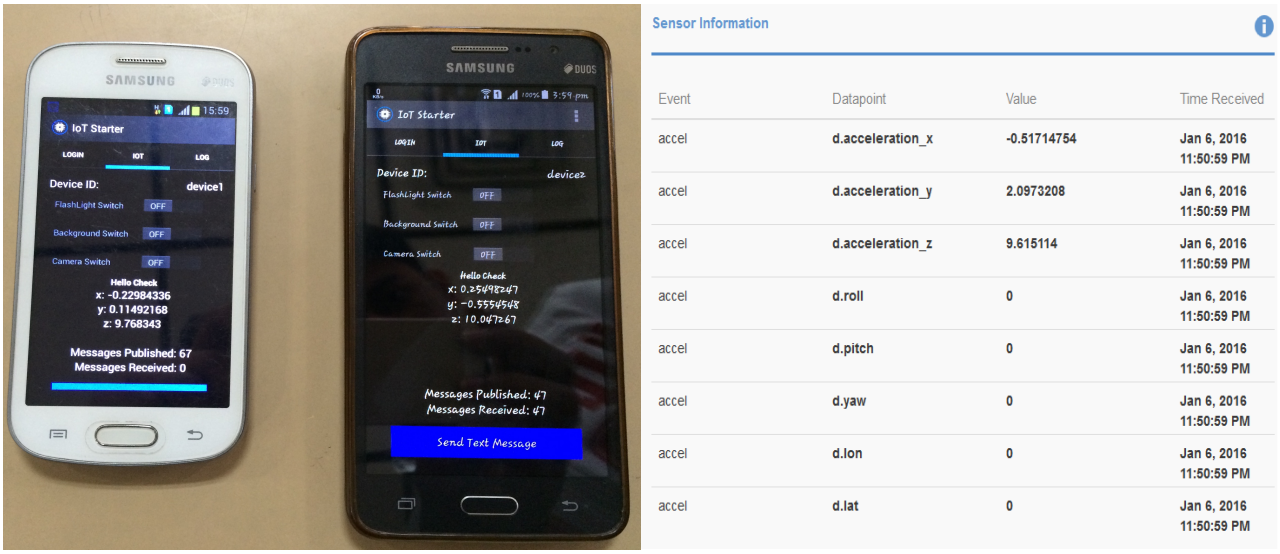

Figure 4.4 : Device publishing data to the server (Left Side), Server receiving data from device (Right Side)

Now device 1 is acting as sensor and device 2 is acting as actuator. For to change the background color device 1 & device 2 both have to permit switch on state. After switching on if the Z-axis is positive than the background color is green. If Z- Axis is negative the background color turn into red.

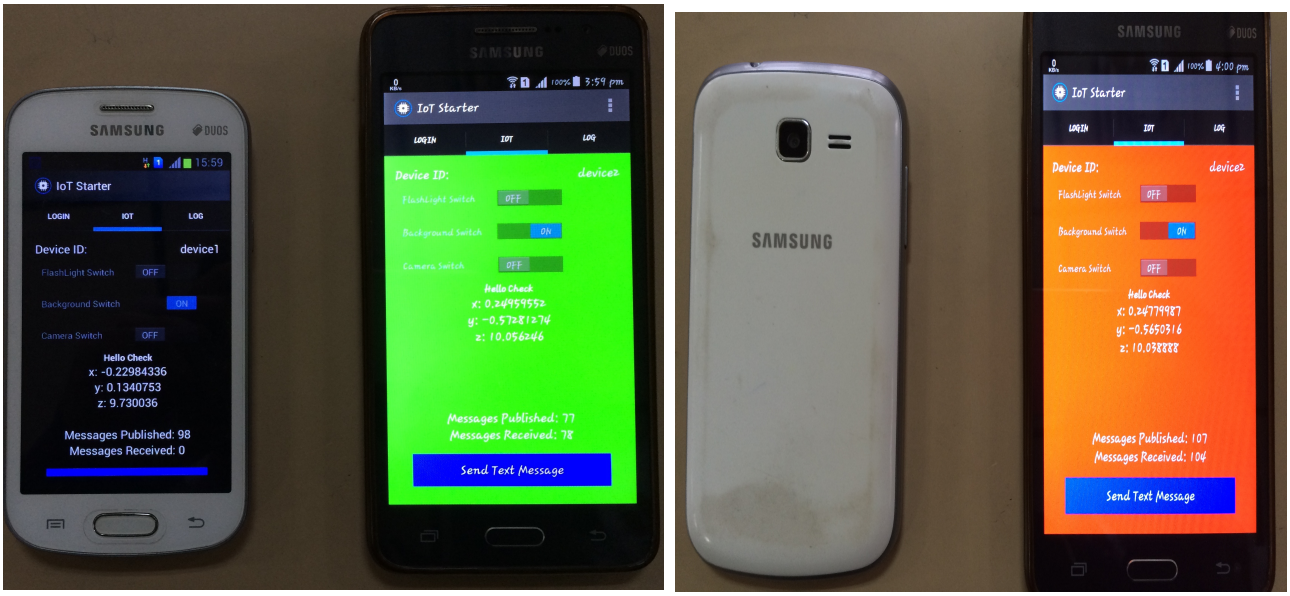

Figure 4.5: Z-axis positive (left image), Z-axis negative (Right image)

Again we are going to turn on the flashlight of actuator device following the same procedure and logic described in previous instruction.

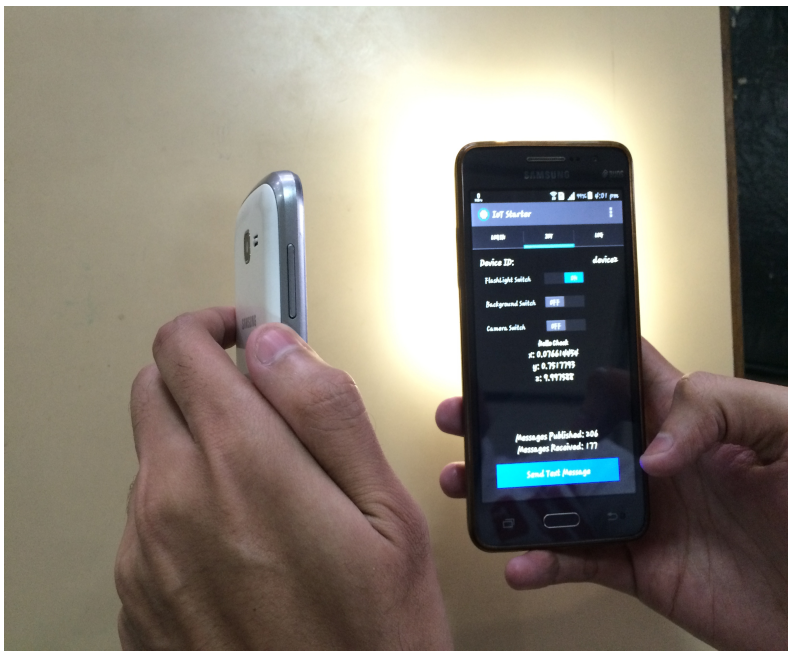

Figure 4.6: Flash light turned on in actuator device for negative Z-axis value

Again following the same procedure we can also auto capture the image from actuator device. Here the device 1 (white) is working as actuator and device 2 (Black) is working as sensor.

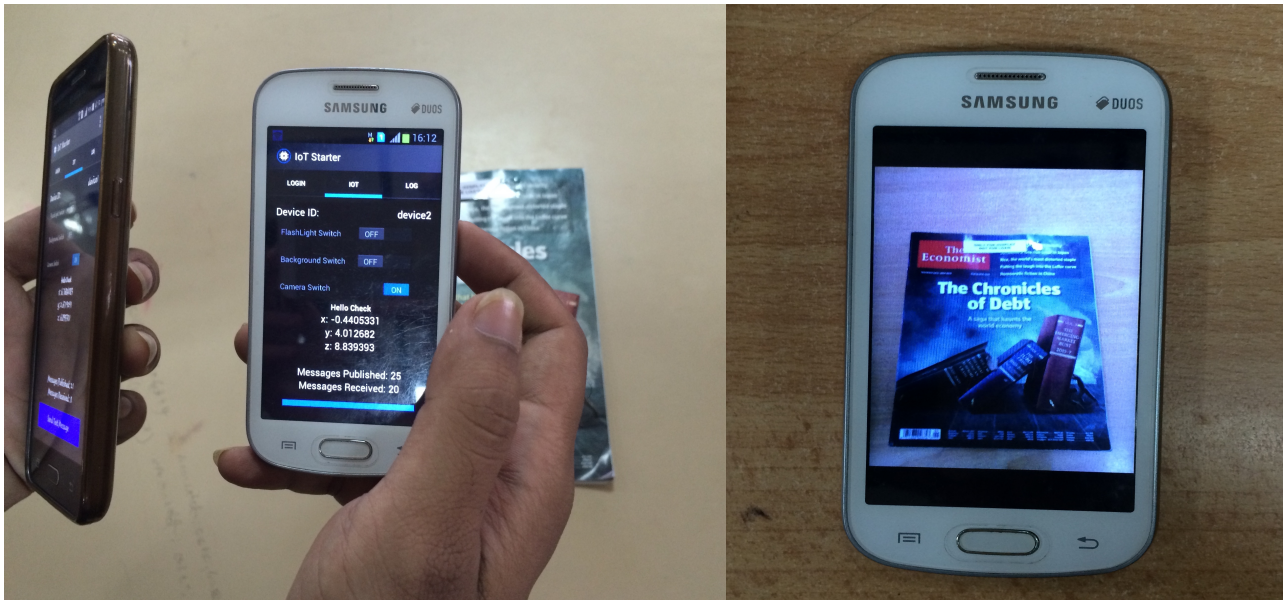

Figure 4.7: actuator device taking picture (left), taken picture in the directory (right)

The server is receiving the accelerometer data, touch event data, text data form sensor side using JSON format. The pictures are given below:

Figure 4.8: Accelerometer JSON Formatted data in server

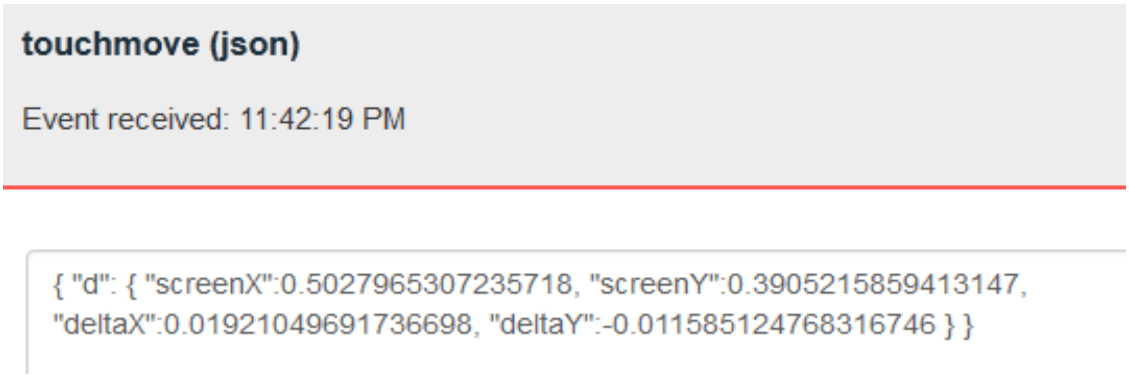

Figure 4.9: Touch event JSON Formatted data in server

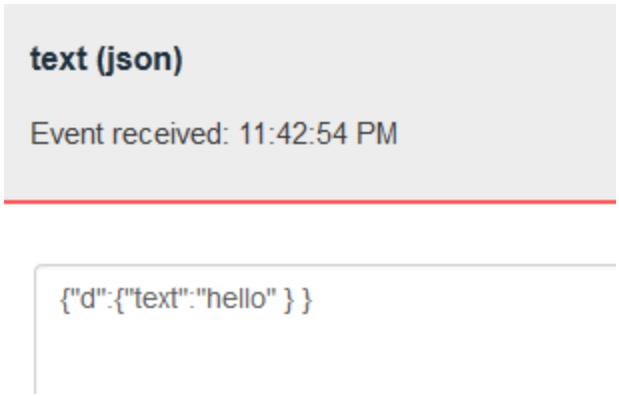

Figure 4.10: Text Message JSON Formatted data in server

4.3 Summary

In this chapter we have presented experimental setup and results with appropriate discussion. In the next chapter we will discuss about the real life future IoT application. How they actually work in real life we are describe that in next chapter.

Chapter 5

Application of IoT

In the last few years the evolution of markets and applications, and therefore their economic potential and their impact in addressing societal trends and challenges for the next decades has changed dramatically. Technological infrastructure provided by the IoT allows for creating and deploying many good applications that will improve the quality of our life.

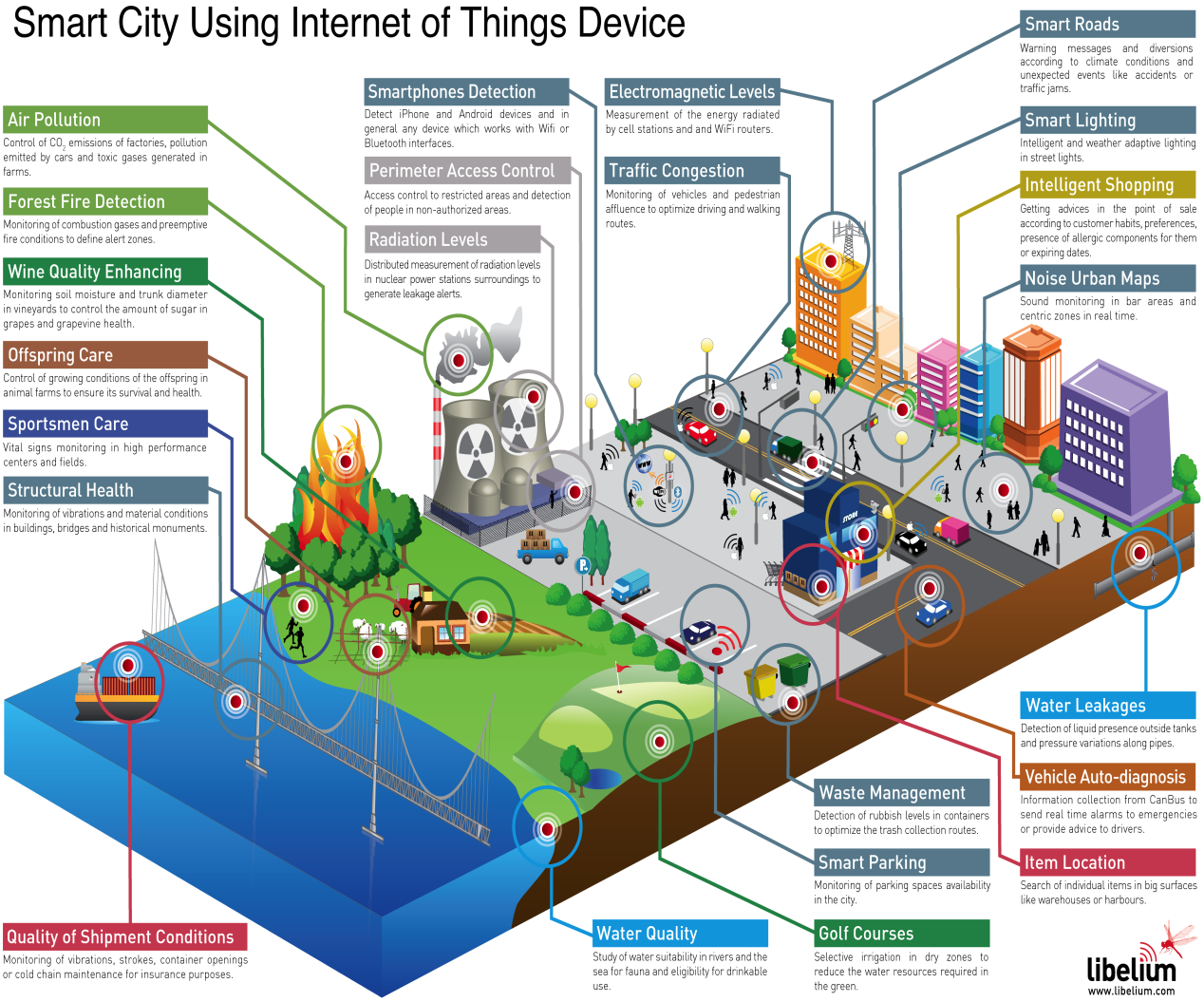

Figure 5.1: Build A smart city using IoT

By implement these application in real life a smart city can be created. Here, we just discuss the most mention application which is working to deploy in real life so that we can get some important idea about how the IoT help us.

5.1 Comfort Living

As envisioned by famous visionaries, we will live in smart environments where our living space is filled with sensing, computing, communication, and actuating devices that collaborate with each other to support our everyday activities. The IoT will be a key to a comfortable life.

5.1.1 Home and Office

RFID tags, sensors, and actuators embedded in homes and offices, together with the Internet, can make our life more comfortable in many ways: garage and house doors can automatically open up based on RFID communication; rooms' air condition is automatically regulated to our presence, preferences, and to the current weather forecast; home/office can be remotely observed and controlled via smart phone; electronic furnitures (e.g., TV, stereo player) learn our preferences to proactively search the Internet for content; room lighting changes according to the time of the day; and many more. A survey on smart home/office applications can be found in [6], [7], and [8]. There are research projects that already aim at realizing some of these applications. For example, the MavHome [9] and iDorm [10] systems learn and adapt to inhabitant behaviors based on sensor observation and power line control. The Gator Tech Smart House [11] strives to be a so-called "programmable pervasive space" that exposes sensors' and actuators' functionalities as an API for developers to build and deploy powerful applications to users. And in [12], an experiment on real deployment of wireless sensor networks for smart home/office environment is conducted and evaluated.

Table 1- "Settings" for IoT Applications (Source: McKinsey Global Institute)

Setting	Description	Examples
Human	Devices attached or inside the human body	Devices (wearables and ingestible) to monitor and maintain human health and wellness; disease management, increased fitness, higher productivity
Home	Buildings where people live	Home controllers and security systems
Retail Environments	Spaces where consumers engage in commerce	Stores, banks, restaurants, arenas – anywhere consumers consider and buy; self-checkout, in-store offers, inventory optimization
Offices	Spaces where knowledge workers work	Energy management and security in office buildings; improved productivity, including for mobile employees
Factories	Standardized production environments	Places with repetitive work routines, including hospitals and farms; operating efficiencies, optimizing equipment use and inventory
Worksites	Custom production environments	Mining, oil and gas, construction; operating efficiencies, predictive maintenance, health and safety
Vehicles	Systems inside moving vehicles	Vehicles including cars, trucks, ships, aircraft, and trains; condition-based maintenance, usage-based design, pre-sales analytics
Cities	Urban environments	Public spaces and infrastructure in urban settings; adaptive traffic control, smart meters, environmental monitoring, resource management
Outside	Between urban environments (and outside other settings)	Outside uses include railroad tracks, autonomous vehicles (outside urban locations), and flight navigation; real-time routing, connected navigation, shipment tracking

5.1.2 Traveling

The IoT technologies will also deliver a comfortable experience for people on traveling. Information about transportation services (e.g., costs, schedules) can be encoded in **NFC** tags that are attached to markers, posters, and panels. The users can retrieve these information by simply using their smartphone over, e.g., a marker, and decide to buy tickets with the smartphone [13]. Touristic maps can also be equipped with NFC tags so that NFC-enabled smartphones can browse them to display touristic information such as hotels, restaurants, points of interest, coming events,

etc, for the users [14].

5.1.3 Shopping

Future applications will be based on sensory data, actuators (e.g., domestic robots), and advanced artificial intelligence software to further improve the quality of our life. In future home or offices, smart refrigerators will automatically do inventory of their content, monitor and detect expired or recalled food items, and create shopping lists based on RFID and sensory information [15]. A house or an office will be able to monitor and adapt to its inhabitants' emotions and habits to assist them accordingly, e.g., play emotional musics or cook black every morning at 8:00 AM. Movies, television, and meetings will be more interactive than ever before as we will be meeting with our 3-D virtual friends in our living room or office [16]. Domestic robots will collaborate with wireless smart devices (e.g., smart refrigerator) to electively perform routine tasks such as cleaning or maintenance in an autonomous fashion (e.g., clean and refill the refrigerator).

5.2 Healthcare

There will be many IoT applications in the healthcare, according to [17] and [18], sensor- and actuator-based assistance systems will monitor elderly people's condition at their own home, raise alarms if necessary (e.g., likely to fall down, heart rate exceeds certain threshold), and provide medical care if required (e.g., via robot doctor), thus saving them from being moved into old people's homes.

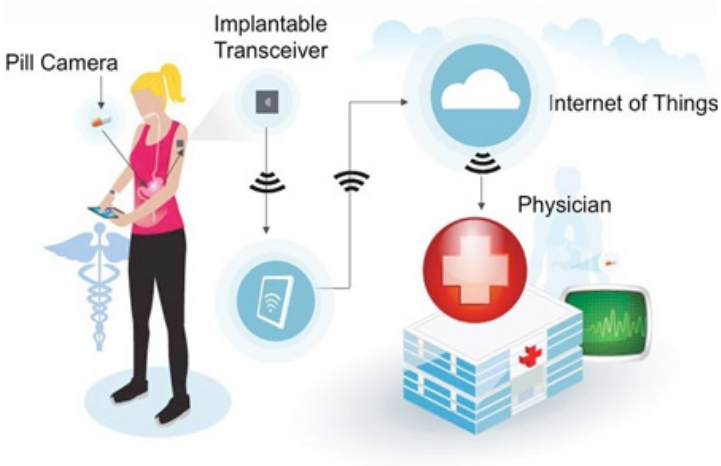

Figure 5.2: Internet of things in Healthcare

Microchips will be developed to be injected into human body, and use body fluid (e.g., blood) to

continuously perform many diagnosis (e.g., inflammation, early tumor cells).

5.3 Automotive

The automotive industry has long been using embedded systems to improve consumer experience during the move. The fast advancements of IoT technologies such as sensors and actuators, micro-controllers, and wireless communications will further deepen this usage. Wireless sensor nodes will be placed along roads and rails, and inside moving vehicles (e.g., cars, trains, buses). These embedded systems will exchange data via ad hoc (e.g., MANET, VANET), overlaying (e.g., GSM, WiMAX) networks, and the Internet, to provide real-time traffic information to drivers and passengers for better navigation and safety. Examples of such infrastructures can be found in [19] and several research projects on the newly emerging Car-to-X paradigm such as the PATH [20] and simTD [21]. Furthermore, Apple is working with General Motors, Honda, Toyota, and other car-makers to integrate its Eyes Free technology onboard [22]. With this integration, a driver can talk to his car to e.g., send an SMS, enter an address for navigation, as well as the car can talk back to the driver, e.g., reads out loud a received SMS or a reminder. The Volvo's V40 car [23] can drive itself in busy traffic without human intervention by automatically maintaining a safe distance to the surrounding vehicles, keeping in lane, and braking when it senses an imminent collision. The tele-service [24] feature developed by BMW allows a car to automatically monitor certain functions that require maintenance such as oil and brakes, and contacts the dealer to schedule an appointment when required, so that the dealer will know the full health status of the car before the consumer arrives. All these examples give an indication that vehicles will become a part of the IoT, having their own intelligence, functioning automatically and even autonomously. In [25], a prototype of a completely autonomous vehicle system is given, that, in addition to sensory information provided by embedded sensors, is based on real-time data taken from Internet clouds to plan efficient paths and to avoid obstacles.

5.4 Security

Future systems for the security purpose will be inspired from more sophisticated hardware and intelligent software and their combination in a fast-evolving IoT. A research project funded by DARPA and Carnegie Mellon University aims at developing an artificial intelligence system that is close to human visual intelligence and can watch and predict what a person will "likely" do in the future [26]. Scientists at the University of Illinois introduced a prototype of a tattoo-like patch to be

laminated onto human skin, that is able to monitor human nerves and muscles via embedded EEG and EMG sensors [27]. Despite of its thinness, the patch contains a wide range of hardware components including LEDs, transistors, wireless antennas, sensors, and conductive coils and solar cells for power. A number of DARPA-funded projects such as HIMEM are developing bug-like machines that are equipped with sophisticated sensors (e.g., camera, acoustic) for "hidden" surveillance systems. In the long term, surveillance systems will even be able to read and analyze human brain waves to proactively react to harmful deeds [28]. There may also be parabolic microphones that can pick up conversations a mile away, cameras that learn what and who to photograph, radars that "see through walls"[29], and nano air vehicles (NAVs) resembling drones to fly in swarms by 2030 [30].

5.5 Energy Saving

In an IoT enhanced future, the status and performance of each urban structure (e.g., side- walk, bridge, building, railway, bus corridor) of the entire city will be continuously monitored and selectively accessible via a series of APIs, thus enabling a live modeling of the entire city, which changes in real time according the live sensory information streams, and is known as City Information Model (CIM) [31]. Based on CIM, urban facility managers and service providers can communicate and collaborate with each other to trade surplus energy, and to calculate prices that match energy supply and demand in real time [32] [33]. Furthermore, with the realization of future smart grids for efficient energy distribution, energy usage optimization across countries will be possible [34].

5.6 Supply Chain

In an IoT-enhanced future, smart containers/pallets will automatically and continuously monitor the condition of the goods (e.g., to ensure right temperature, to avoid toxic chemical substances) during transportation, track the transportation route and their actual position to ensure that they are arriving at their destination [35]. Supply chains will be capable of learning and making decisions by themselves, without human involvement. For example, they could reconfigure supply chain networks when disruptions occur, or could acquire rights to use physical assets like production capacity, distribution facilities and transportation fleets on demand through virtual exchanges [36].

5.7 Summary

In this section we are discussing about some real life application in our life style. By discussing these applications now we can understand how IoT will impact in future Technology. In the next chapter we will conclude this thesis mentioning the contribution of this research, limitations of the proposal system, future works related with this research.

Chapter 6

Conclusion

Chapter summarize the current work. Section 6.1 presents the contributions of the proposed systems. Section 6.2 describes the limitations of the proposed system regarding to distributed agents architecture using IoT. Section 6.3 focuses on future works in the domain proposed system.

6.1 Contributions

The design system of our application is basically show how data generate from a sensor device and send the data to the server and than server send that receive data, process it and send another device which is an actuator. We motivated by IBM corporation to do this research work and used IBM open source project to launch our application and we also did some extra modification using open source template which work successfully.

6.2 Limitations of these Systems

our project has some limitation as like if we use auto image capture module for the application than the camera service can capture only a single picture. For again use the auto capture image module we need to restart the camera service. We need a strong strength internet connection for operate this project. Here, we are using only IBM Bluemix server for transfer data from one device to another device but we can do this by building our own web server application which is more secured for data. By using our application we can connect at most twenty android device for connecting with IoT. But in real life it is possible to work with many devices.

6.3 Future Works

As discussed in chapter 4, we implement this research by store the logic in server, we can also work on put the logic in device. In future we will try all combination setup to check which is more faster. All combination means many possibility for transfer data using internet of things.

Reference

[1] Lianos, M. and Douglas, M. (2000) Dangerization and the End of Deviance: The Institutional Environment. *British Journal of Criminology*, 40, 261-278. http://dx.doi.org/10.1093/bjc/40.2.261

[2] Ferguson, T. (2002) Have Your Objects Call My Object. *Harvard Business Review*, June, 1-7.

[3] https://en.wikipedia.org/wiki/MQTT

[4] http://www.ibm.com/developerworks/cloud/library/cl-bluemixfoundry/

[5] https://en.wikipedia.org/wiki/Bluemix

[6] Jin Cheng and Thomas Kunz. A Survey on Smart Home Networking. Carleton Uni- versity, Systems and Computer Engineering, Technical Report SCE-09-10, September 2009.

[7] Abowd G and Mynatt E. Designing for the human experience in smart environments. In: Cook D, Das S, editors. Smart Environments: Technology, Protocols, and Applica- tions, 2004.

[8] Rashvand Habib F. and Alcaraz Calero Jose M. Distributed Sensor Systems: Practice and Applications. John Wiley & Sons., Ltd, 2012.

[9] G. Michael Youngblood and Diane J. Cook. Data Mining for Hierarchical Model Cre- ation. IEEE Transactions on Systems, Man, and Cybernetics, 37:561–572, July 2007.

[10] F. Doctor, H. Hagras, and V. Callaghan. A Fuzzy Embedded Agent-based Approach for Realizing Ambient Intelligence in Intelligent Inhabited Environments. IEEE Trans- actions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 35:55–65, January 2005.

[11] Sumi Helal, William Mann, Hicham El-Zabadani, Jefi rey King, Youssef Kaddoura, and Erwin Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. Computer, 38(3):50–60, 2005.

[12] Dipak Surie, Olivier Laguionie, and Thomas Pederson. Wireless Sensor Networking of Everyday Objects in a Smart Home Environment. ISSNIP'08, 2008.

[13] G. Broll, E. Rukzio, M. Paolucci, M. Wagner, A. Schmidt, and H. Hussmann. PERCI: Pervasive service integration with the Internet of Things. IEEE Internet Computing, 13:74–81, November 2009.

[14] D. Reilly, M. Welsman-Dinelle, C. Bate, and K. Inkpen. Just Point and Click? Using Handhelds to Interact with Paper Maps. In Proc. of the 7th Intl. Conf. on Human Com- puter

Interaction with Mobile Devices & Services (MobileHCI'05), September 2005.

[15] RSA Labotaries. RFID, a Vision of the Future. URL http://www.rsa.com/rsalabs/ node.asp? id=2117.

[16] http://www.futuretechnology500.com/index.php/future-homes/

[17] Universitat Polit`ecnica de Catalunya. Advances In Medical Technology: What Does

The Future Hold? ScienceDaily, 16 Jun. 2009. Web. 30 Mar. 2013.

[18] Future Trends in Medical Technology. URL www.medica.de.

[19] Ben-Jye Chang, Bo-Jhang Huang, and Ying-Hsin Liang. Wireless Sensor Network- Based Adaptive Vehicle Navigation in Multihop-Relay WiMAX Networks. Advanced Information Networking and Applications, 2008.

[20] http://www.path.berkeley.edu/

[21]http://www.psychologie.uniwuerzburg.de/izvw/forschung/projekte/fahrerinformation/simtd.php .en

[22]http://www.wired.com/autopia/2012/07/siri-eyes-free-competing/

[23]http://www.economist.com/node/21548992

[24] http://www.bmw.com/com/en/owners/service/bmw teleservices.html

[25] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A Cloud-Assisted Design for Autonomous Driving. MCC'12, August 17, 2012, Helsinki, Finland.

[26] Alessandro Oltramari and Christian Lebiere. Using Ontologies in a Cognitive-Grounded System: Automatic Action Recognition in Video Surveillance. In Proc. of Semantic Technology for Intelligence, Defense, and Security, 2012.

[27] Dae-Hyeong Kim et al. Epidermal Electronics. Science, 333:838–843, August 2011.

[28] University of Pennsylvania. Mind Reading from Brain Recordings? "Neural Finger- prints" of Memory Associations Decoded. ScienceDaily, 26 Jun. 2012. Web. 28 Mar. 2013.

[29] http://topdocumentaryfilms.com/surveillance-technology/

[30] http://www.activistpost.com/2011/02/how-close-are-we-to-nano-based.html

[31] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. Computer Networks, 2010.

[32] Jorge Gil, Ju´lio Almeida, and Jos´e Pinto Duarte. The backbone of a City Information Model (CIM): Implementing a spatial data model for urban design. City Modelling - eCAADe, 2011.

[33] Bentley Systems. City Information Modeling for Sustaining Cities: Lessons Learned from Advanced Users. Case Study Showcase, August 2011.

[34] Hamid Gharavi and Reza Ghafurian. Smart Grid: The Electric Energy System of the Future. Proceedings of the IEEE, 99(6):917–921, June 2011.

[35] SENSEI FP7 Project. Scenario Portfolio: User and Context Requirements. URL http: //www.sensei-project.eu.

[36] IBM. The smarter supply chain of the future: Insights from the Global Chief Supply Chain Officer Study.

[37] (2005) ITU Internet Reports, International Telecommunication Union. The Internet of Things: 7th Edition. www.itu.int/internetofthings/on

[38] Want, R. (2006) An Introduction to RFID Technology. *IEEE Pervasive Computing*, **5**, 25-33.

[39] Sun, C. (2012) Application of RFID Technology for Logistics on Internet of Things.

[40] Aggarwal, R. and Lal Das, M. (2012) RFID Security in the Context of "Internet of Things". *First International Conference on Security of Internet of Things*, Kerala, 17-19 August 2012, 51-56. http://dx.doi.org/10.1145/2490428.2490435

[41] Moeinfar, D., Shamsi, H. and Nafar, F. (2012) Design and Implementation of a Low-Power Active RFID for Container Tracking @ 2.4 GHz Frequency: Scientific Research.

[42] Bicknell, IPv6 Internet Broken, Verizon Route Prefix Length Policy, 2009.

[43] Pahlavan, K., Krishnamurthy, P., Hatami, A., Ylianttila, M., Makela, J.P., Pichna, R. and Vallstron, J. (2007) Handoff in Hybrid Mobile Data Networks. *Mobile and Wireless Communication Summit*, **7**, 43-47.

[44] Chen, X.-Y. and Jin, Z.-G. (2012) Research on Key Technology and Applications for the Internet of Things. *Physics Procedia*, **33**,561-566. http://dx.doi.org/10.1016/j.phpro.2012.05.104

[45] Arampatzis, T., *et al*. (2005) A Survey of Security Issues in Wireless Sensors Networks, in Intelligent Control. *Proceeding of the IEEE International Symposium on, Mediterrean Conference on Control and Automation*, 719-724.

[46] Chorost, M. (2008) The Networked Pill, MIT Technology Review, March.

Appendix

Essential Source Code of This Project

```
/********************** IoTFragment  **********************/

public class IoTFragment extends IoTStarterFragment
{
   private final static String TAG = IoTFragment.class.getName();
   private Switch light,camera,background;
   private TextView textTimeLeft; // time left field


   /*************************************************************************
    * Fragment functions for establishing the fragment
    *************************************************************************/

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
   {
      return inflater.inflate(R.layout.iot, container, false);
   }

   /**
    * Called when the fragment is resumed.
    */
   @Override
   public void onResume() {
      Log.d(TAG, ".onResume() entered");

      super.onResume();
      app = (IoTStarterApplication) getActivity().getApplication();
      app.setCurrentRunningActivity(TAG);

      if (broadcastReceiver == null) {
         Log.d(TAG, ".onResume() - Registering iotBroadcastReceiver");
         broadcastReceiver = new BroadcastReceiver() {

            @Override
            public void onReceive(Context context, Intent intent) {
               Log.d(TAG, ".onReceive() - Received intent for iotBroadcastReceiver");
               processIntent(intent);
            }
         };
      }

      getActivity().getApplicationContext().registerReceiver(broadcastReceiver,
            new IntentFilter(Constants.APP_ID + Constants.INTENT_IOT));

      // initialise
      initializeIoTActivity();
   }

   /**
    * Called when the fragment is destroyed.
    */
   @Override
   public void onDestroy() {
      Log.d(TAG, ".onDestroy() entered");
```

```
    try {
        getActivity().getApplicationContext().unregisterReceiver(broadcastReceiver);
    } catch (IllegalArgumentException iae) {
        // Do nothing
    }
    super.onDestroy();
}

/**
 * Initializing onscreen elements and shared properties
 */
private void initializeIoTActivity() {
    Log.d(TAG, ".initializeIoTFragment() entered");

    context = getActivity().getApplicationContext();

    updateViewStrings();


    light = (Switch) getActivity().findViewById(R.id.lightSwitch);
    camera = (Switch) getActivity().findViewById(R.id.cameraSwitch);
    background = (Switch) getActivity().findViewById(R.id.backgroundSwitch);
    textTimeLeft = (TextView) getActivity().findViewById(R.id.textTimeLeft);

    textTimeLeft.setText(Constants.TAKING_PICTURE);


    light.setChecked(false);
    camera.setChecked(false);
    background.setChecked(false);

    light.setOnCheckedChangeListener(new OnCheckedChangeListener()
    {
                @Override
                public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
                {
                        // TODO Auto-generated method stub
                        if(isChecked) Constants.LIGHT_SWITCH = true;
                        else Constants.LIGHT_SWITCH = false;

                }
        });


    camera.setOnCheckedChangeListener(new OnCheckedChangeListener()
    {
                @Override
                public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
                {
                        // TODO Auto-generated method stub
                        if(isChecked) Constants.CAMERA_SWITCH = true;
                        else Constants.CAMERA_SWITCH = false;

                }
        });

    background.setOnCheckedChangeListener(new OnCheckedChangeListener()
    {
                @Override
                public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
                {
                        // TODO Auto-generated method stub
                        if(isChecked) Constants.BACKGROUND_SWITCH = true;
                        else Constants.BACKGROUND_SWITCH = false;

                }
        });
```

```
    // setup button listeners
    Button button = (Button) getActivity().findViewById(R.id.sendText);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            handleSendText();
        }
    });
}

/**
 * Update strings in the fragment based on IoTStarterApplication values.
 */
@Override
protected void updateViewStrings() {
    Log.d(TAG, ".updateViewStrings() entered");
    // DeviceId should never be null at this point.
    if (app.getDeviceId() != null) {
        ((TextView) getActivity().findViewById(R.id.deviceIDIoT)).setText(app.getDeviceId());
    } else {
        ((TextView) getActivity().findViewById(R.id.deviceIDIoT)).setText("-");
    }

    // Update publish count view.
    processPublishIntent();

    // Update receive count view.
    processReceiveIntent();

    int unreadCount = app.getUnreadCount();
    ((MainActivity) getActivity()).updateBadge(getActivity().getActionBar().getTabAt(2), unreadCount);
}

/*****************************************************************************
 * Functions to handle button presses
 *****************************************************************************/

/**
 * Handle pressing of the send text button. Prompt the user to enter text
 * to send.
 */
private void handleSendText() {
    Log.d(TAG, ".handleSendText() entered");
    if (app.getConnectionType() != Constants.ConnectionType.QUICKSTART) {
        final EditText input = new EditText(context);
        new AlertDialog.Builder(getActivity())
                .setTitle(getResources().getString(R.string.send_text_title))
                .setMessage(getResources().getString(R.string.send_text_text))
                .setView(input)
                .setPositiveButton(getResources().getString(R.string.ok), new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        Editable value = input.getText();
                        String messageData = MessageFactory.getTextMessage(value.toString());
                        MqttHandler mqtt = MqttHandler.getInstance(context);
                        mqtt.publish(TopicFactory.getEventTopic(Constants.TEXT_EVENT), messageData, false, 0);
                    }
                }).setNegativeButton(getResources().getString(R.string.cancel), new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                // Do nothing.
            }
        }).show();
    } else {
        new AlertDialog.Builder(getActivity())
                .setTitle(getResources().getString(R.string.send_text_title))
                .setMessage(getResources().getString(R.string.send_text_invalid))
                .setPositiveButton(getResources().getString(R.string.ok), new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                    }
                }).show();
```

```
    }
}

/**************************************************************************
 * Functions to process intent broadcasts from other classes
 **************************************************************************/

/**
 * Process the incoming intent broadcast.
 * @param intent The intent which was received by the fragment.
 */
private void processIntent(Intent intent) {
    Log.d(TAG, ".processIntent() entered");

    // No matter the intent, update log button based on app.unreadCount.
    updateViewStrings();

    String data = intent.getStringExtra(Constants.INTENT_DATA);
    assert data != null;
    if (data.equals(Constants.INTENT_DATA_PUBLISHED)) {
        processPublishIntent();
    } else if (data.equals(Constants.INTENT_DATA_RECEIVED)) {
        processReceiveIntent();
    } else if (data.equals(Constants.ACCEL_EVENT)) {
        processAccelEvent();
    } else if (data.equals(Constants.COLOR_EVENT)) {
        Log.d(TAG, "Updating background color");
        getView().setBackgroundColor(app.getColor());
    } else if (data.equals(Constants.ALERT_EVENT)) {
        String message = intent.getStringExtra(Constants.INTENT_DATA_MESSAGE);
        new AlertDialog.Builder(getActivity())
                .setTitle(getResources().getString(R.string.alert_dialog_title))
                .setMessage(message)
                .setPositiveButton(getResources().getString(R.string.ok), new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                    }
                }).show();
    }
}

/**
 * Intent data contained INTENT_DATA_PUBLISH
 * Update the published messages view based on app.getPublishCount()
 */
private void processPublishIntent() {
    Log.v(TAG, ".processPublishIntent() entered");
    String publishedString = this.getString(R.string.messages_published);
    publishedString = publishedString.replace("0",Integer.toString(app.getPublishCount()));
    ((TextView) getActivity().findViewById(R.id.messagesPublishedView)).setText(publishedString);
}

/**
 * Intent data contained INTENT_DATA_RECEIVE
 * Update the received messages view based on app.getReceiveCount();
 */
private void processReceiveIntent() {
    Log.v(TAG, ".processReceiveIntent() entered");
    String receivedString = this.getString(R.string.messages_received);
    receivedString = receivedString.replace("0",Integer.toString(app.getReceiveCount()));
    ((TextView) getActivity().findViewById(R.id.messagesReceivedView)).setText(receivedString);
}

/**
 * Update acceleration view strings
 */
private void processAccelEvent() {
    Log.v(TAG, ".processAccelEvent()");
    float[] accelData = app.getAccelData();
    ((TextView) getActivity().findViewById(R.id.accelX)).setText("x: " + accelData[0]);
```

```
        ((TextView) getActivity().findViewById(R.id.accelY)).setText("y: " + accelData[1]);
        ((TextView) getActivity().findViewById(R.id.accelZ)).setText("z: " + accelData[2]);
    }
}
```

```
/****************** IoT Starter Application ********************/


    /**
     * Enables or disables the publishing of accelerometer data
     */

    public void toggleAccel()
    {
        this.setAccelEnabled(!this.isAccelEnabled());
        if (connected && accelEnabled)
        {
            // Device Sensor was previously disabled, and the device is connected, so enable the sensor
            if (deviceSensor == null)
            {
                deviceSensor = DeviceSensor.getInstance(this);
            }
            deviceSensor.enableSensor();
        }
        else if (connected && !accelEnabled)
        {
            // Device Sensor was previously enabled, and the device is connected, so disable the sensor
            if (deviceSensor != null)
            {
                deviceSensor.disableSensor();
            }
        }
    }



/**
     * Turn flashlight on or off when a light command message is received.
     */
    public void handleLightMessage()
    {
        Log.d(TAG, ".handleLightMessage() entered");
        if (this.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH))
        {
            if (!isCameraOn)
            {
                Log.d(TAG, "FEATURE_CAMERA_FLASH true");
                camera = Camera.open();
                Camera.Parameters p = camera.getParameters();
                p.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
                camera.setParameters(p);
                camera.startPreview();
                isCameraOn = true;
            }

            else
            {
                camera.stopPreview();
                camera.release();
                isCameraOn = false;
            }
        }
        else
        {
            Log.d(TAG, "FEATURE_CAMERA_FLASH false");
```

```
    }
}

public void handleLightOffMessage()
{
        if(isCameraOn == true)
        {
      camera.stopPreview();
      camera.release();
      isCameraOn = false;
        }
}


//////////////////////Image Auto Capturing Functionality

public void CaptureImageAuto()
{


                camera = Camera.open();
                SurfaceView view = new SurfaceView(this);

                try
                {
                        camera.setPreviewDisplay(view.getHolder());

                        new CountDownTimer(2000,1000)
                        {

                                @Override
                                public void onFinish()
                                {
                                        // count finished
                                        //textTimeLeft.setText("Picture Taken");
                                        Constants.TAKING_PICTURE = "Picture Taken";
                                        camera.takePicture(null, null, null, jpegCallBack);
                                }

                                @Override
                                public void onTick(long millisUntilFinished) {
                                        // every time 1 second passes
                                        //textTimeLeft.setText("Seconds Left: "+millisUntilFinished/1000);
                                        Constants.TAKING_PICTURE = "Seconds Left: "+millisUntilFinished/1000;
                                }

                        }.start();
                }
                catch (IOException e)
                {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                camera.startPreview();

}


        Camera.PictureCallback jpegCallBack=new Camera.PictureCallback()
        {
                public void onPictureTaken(byte[] data, Camera camera)
                {
                        // set file destination and file name
                        //File destination=new File(Environment.getExternalStorageDirectory(),"myPicture.jpg");
                        File destination = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES),
Long.toString(System.currentTimeMillis()) + ".jpg");
                        try
                        {
```

```
                                        Bitmap userImage = BitmapFactory.decodeByteArray(data, 0, data.length);
                                        // set file out stream
                                        FileOutputStream out = new FileOutputStream(destination);
                                        // set compress format quality and stream
                                        userImage.compress(Bitmap.CompressFormat.JPEG, 90, out);
                                }
                                catch (FileNotFoundException e)
                                {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                        }
                };


/********************** MessageConductorActivity **********************/
public class MessageConductor
{
   private final static String TAG = MessageConductor.class.getName();
   private static MessageConductor instance;
   private Context context;
   private IoTStarterApplication app;

   private MessageConductor(Context context)
   {
      this.context = context;
      app = (IoTStarterApplication) context.getApplicationContext();
   }

   public static MessageConductor getInstance(Context context)
   {
      if (instance == null)
      {
         instance = new MessageConductor(context);
      }
      return instance;
   }
   /**
    * Steer incoming MQTT messages to the proper activities based on their content.
    *
    * @param payload The log of the MQTT message.
    * @param topic The topic the MQTT message was received on.
    * @throws JSONException If the message contains invalid JSON.
    */
   public void steerMessage(String payload, String topic) throws JSONException
   {
      Log.d(TAG, ".steerMessage() entered");
      JSONObject top = new JSONObject(payload);
      JSONObject d = top.getJSONObject("d");

      if (topic.contains(Constants.COLOR_EVENT))
      {
         Log.d(TAG, "Color Event");
         int r = d.getInt("r");
         int g = d.getInt("g");
         int b = d.getInt("b");

         // alpha value received is 0.0 < a < 1.0 but Color.agrb expects 0 < a < 255
         int alpha = (int)(d.getDouble("alpha")*255.0);
         if ((r > 255 || r < 0) || (g > 255 || g < 0) || (b > 255 || b < 0) || (alpha > 255 || alpha < 0))
         {
            return;
         }

         if(Constants.LIGHT_SWITCH == true)
         {
            if(g==255 && r==0 && b==0)
            {
```

```
                app.handleLightOffMessage(); ////////////////////////////////////Light off if red background
            }
            else if(r==255 && g==0 && b==0)
            {
                app.handleLightMessage(); ////////////////////////////////////Light on if green background
            }
        }
        else if(Constants.CAMERA_SWITCH == true)
        {
            if(r==255 && g==0 && b==0)
            {
                app.CaptureImageAuto();
            }
        }
        else if(Constants.BACKGROUND_SWITCH == true)
        {
            app.setColor(Color.argb(alpha, r, g, b));
        }
        else if(Constants.LIGHT_SWITCH == false && Constants.CAMERA_SWITCH == false &&
Constants.BACKGROUND_SWITCH == false)
        {
            app.setColor(Color.argb(alpha, 0, 0, 0));
            app.handleLightOffMessage();
        }


        String runningActivity = app.getCurrentRunningActivity();
        if (runningActivity != null && runningActivity.equals(IoTFragment.class.getName()))
        {
            Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_IOT);
            actionIntent.putExtra(Constants.INTENT_DATA, Constants.COLOR_EVENT);
            context.sendBroadcast(actionIntent);
        }
    }
    else if (topic.contains(Constants.LIGHT_EVENT))
    {
        app.handleLightMessage();
    }
    else if (topic.contains(Constants.TEXT_EVENT))
    {
        int unreadCount = app.getUnreadCount();
        app.setUnreadCount(++unreadCount);

        // save payload in an arrayList
        List messageRecvd = new ArrayList<String>();
        messageRecvd.add(payload);

        app.getMessageLog().add(d.getString("text"));

        String runningActivity = app.getCurrentRunningActivity();
        if (runningActivity != null && runningActivity.equals(LogFragment.class.getName())) {
            Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOG);
            actionIntent.putExtra(Constants.INTENT_DATA, Constants.TEXT_EVENT);
            context.sendBroadcast(actionIntent);
        }

        Intent unreadIntent;
        if (runningActivity.equals(LogFragment.class.getName()))
        {
            unreadIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOG);
        }
        else if (runningActivity.equals(LoginFragment.class.getName()))
        {
            unreadIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOGIN);
        }
        else if (runningActivity.equals(IoTFragment.class.getName()))
        {
            unreadIntent = new Intent(Constants.APP_ID + Constants.INTENT_IOT);
        }
```

```
            else if (runningActivity.equals(ProfilesActivity.class.getName()))
            {
               unreadIntent = new Intent(Constants.APP_ID + Constants.INTENT_PROFILES);
            }
            else
            {
               return;
            }

            String messageText = d.getString("text");
            if (messageText != null)
            {
               unreadIntent.putExtra(Constants.INTENT_DATA, Constants.UNREAD_EVENT);
               context.sendBroadcast(unreadIntent);
            }
         }
         else if (topic.contains(Constants.ALERT_EVENT))
         {
            // save payload in an arrayList
            int unreadCount = app.getUnreadCount();
            app.setUnreadCount(++unreadCount);

            List messageRecvd = new ArrayList<String>();
            messageRecvd.add(payload);

            app.getMessageLog().add(d.getString("text"));

            String runningActivity = app.getCurrentRunningActivity();
            if (runningActivity != null) {
               if (runningActivity.equals(LogFragment.class.getName()))
               {
                  Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOG);
                  actionIntent.putExtra(Constants.INTENT_DATA, Constants.TEXT_EVENT);
                  context.sendBroadcast(actionIntent);
               }

               Intent alertIntent;
               if (runningActivity.equals(LogFragment.class.getName()))
               {
                  alertIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOG);
               }
               else if (runningActivity.equals(LoginFragment.class.getName()))
               {
                  alertIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOGIN);
               }
               else if (runningActivity.equals(IoTFragment.class.getName()))
               {
                  alertIntent = new Intent(Constants.APP_ID + Constants.INTENT_IOT);
               }
               else if (runningActivity.equals(ProfilesActivity.class.getName()))
               {
                  alertIntent = new Intent(Constants.APP_ID + Constants.INTENT_PROFILES);
               }
               else
               {
                  return;
               }

               String messageText = d.getString("text");
               if (messageText != null)
               {
                  alertIntent.putExtra(Constants.INTENT_DATA, Constants.ALERT_EVENT);
                  alertIntent.putExtra(Constants.INTENT_DATA_MESSAGE, d.getString("text"));
                  context.sendBroadcast(alertIntent);
               }
            }
         }
      }
   }
}
```

```
/********************MqttHandler*********************/


public class MqttHandler implements MqttCallback
{

   private final static String TAG = MqttHandler.class.getName();
   private static MqttHandler instance;
   private MqttAndroidClient client;
   private Context context;
   private IoTStarterApplication app;

   private MqttHandler(Context context)
   {
      this.context = context;
      this.app = (IoTStarterApplication) context.getApplicationContext();
      this.client = null;
   }

   /**
    * @param context The application context for the object.
    * @return The MqttHandler object for the application.
    */
   public static MqttHandler getInstance(Context context)
   {
      Log.d(TAG, ".getInstance() entered");
      if (instance == null)
      {
         instance = new MqttHandler(context);
      }
      return instance;
   }

   /**
    * Connect MqttAndroidClient to the MQTT server
    */
   public void connect()
   {
      Log.d(TAG, ".connect() entered");

      // check if client is already connected
      if (!isMqttConnected())
      {
         String serverHost;
         String serverPort = Constants.SETTINGS_MQTT_PORT;
         String clientId;
         if (app.getConnectionType() == Constants.ConnectionType.M2M)
         {
            serverHost = Constants.M2M_DEMO_SERVER;
            clientId = Constants.M2M_CLIENTID + app.getDeviceId();
         }
         else if (app.getConnectionType() == Constants.ConnectionType.QUICKSTART)
         {
            serverHost = Constants.QUICKSTART_SERVER;
            clientId = "d:" + app.getOrganization() + ":" + Constants.DEVICE_TYPE + ":" + app.getDeviceId();
         }
         else
         {
            serverHost = app.getOrganization() + "." + Constants.SETTINGS_MQTT_SERVER;
            clientId = "d:" + app.getOrganization() + ":" + Constants.DEVICE_TYPE + ":" + app.getDeviceId();
         }

         Log.d(TAG, ".initMqttConnection() - Host name: " + serverHost + ", Port: " + serverPort
               + ", client id: " + clientId);

         String connectionUri = "tcp://" + serverHost + ":" + serverPort;
```

```
        if (client != null)
        {
            client.unregisterResources();
            client = null;
        }
        client = new MqttAndroidClient(context, connectionUri, clientId);
        client.setCallback(this);

        // create ActionListener to handle connection results
        ActionListener listener = new ActionListener(context, Constants.ActionStateStatus.CONNECTING);
        // create MqttConnectOptions and set the clean session flag
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(true);

        if (app.getConnectionType() == Constants.ConnectionType.IOTF)
        {
            options.setUserName(Constants.SETTINGS_USERNAME);
            options.setPassword(app.getAuthToken().toCharArray());
        }

        try
        {
            // connect
            client.connect(options, context, listener);
        }
        catch (MqttException e)
        {
            Log.e(TAG, "Exception caught while attempting to connect to server", e.getCause());
            if (e.getReasonCode() == (Constants.ERROR_BROKER_UNAVAILABLE))
            {
                // error while connecting to the broker - send an intent to inform the user
                Intent actionIntent = new Intent(Constants.ACTION_INTENT_CONNECTIVITY_MESSAGE_RECEIVED);
                actionIntent.putExtra(Constants.CONNECTIVITY_MESSAGE, Constants.ERROR_BROKER_UNAVAILABLE);
                context.sendBroadcast(actionIntent);
            }
        }
    }
}

/**
 * Disconnect MqttAndroidClient from the MQTT server
 */
public void disconnect()
{
    Log.d(TAG, ".disconnect() entered");

    // check if client is actually connected
    if (isMqttConnected())
    {
        ActionListener listener = new ActionListener(context, Constants.ActionStateStatus.DISCONNECTING);
        try
        {
            // disconnect
            client.disconnect(context, listener);
        }
        catch (MqttException e)
        {
            Log.e(TAG, "Exception caught while attempting to disconnect from server", e.getCause());
        }
    }
}

/**
 * Subscribe MqttAndroidClient to a topic
 *
 * @param topic to subscribe to
 * @param qos   to subscribe with
 */
public void subscribe(String topic, int qos)
```

```
{
   Log.d(TAG, ".subscribe() entered");

   // check if client is connected
   if (isMqttConnected())
   {
      try
      {
         // create ActionListener to handle subscription results
         ActionListener listener = new ActionListener(context, Constants.ActionStateStatus.SUBSCRIBE);
         Log.d(TAG, ".subscribe() - Subscribing to: " + topic + ", with QoS: " + qos);
         client.subscribe(topic, qos, context, listener);
      }
      catch (MqttException e)
      {
         Log.e(TAG, "Exception caught while attempting to subscribe to topic " + topic, e.getCause());
      }
   }
   else
   {
      connectionLost(null);
   }
}

/**
 * Unsubscribe MqttAndroidClient from a topic
 *
 * @param topic to unsubscribe from
 */
public void unsubscribe(String topic)
{
   Log.d(TAG, ".unsubscribe() entered");

   // check if client is connected
   if (isMqttConnected())
   {
      try
      {
         // create ActionListener to handle unsubscription results
         ActionListener listener = new ActionListener(context, Constants.ActionStateStatus.UNSUBSCRIBE);
         client.unsubscribe(topic, context, listener);
      }
      catch (MqttException e)
      {
         Log.e(TAG, "Exception caught while attempting to unsubscribe from topic " + topic, e.getCause());
      }
   }
   else
   {
      connectionLost(null);
   }
}

/**
 * Publish message to a topic
 *
 * @param topic    to publish the message to
 * @param message  JSON object representation as a string
 * @param retained true if retained flag is requred
 * @param qos      quality of service (0, 1, 2)
 */
public void publish(String topic, String message, boolean retained, int qos)
{
   Log.d(TAG, ".publish() entered");

   // check if client is connected
   if (isMqttConnected())
   {
      // create a new MqttMessage from the message string
```

```
            MqttMessage mqttMsg = new MqttMessage(message.getBytes());
            // set retained flag
            mqttMsg.setRetained(retained);
            // set quality of service
            mqttMsg.setQos(qos);
            try
            {
               // create ActionListener to handle message published results
               ActionListener listener = new ActionListener(context, Constants.ActionStateStatus.PUBLISH);
               Log.d(TAG, ".publish() - Publishing " + message + " to: " + topic + ", with QoS: " + qos + " with retained flag set to " +
retained);
               client.publish(topic, mqttMsg, context, listener);

               int count = app.getPublishCount();
               app.setPublishCount(++count);

               String runningActivity = app.getCurrentRunningActivity();
               if (runningActivity != null && runningActivity.equals(IoTFragment.class.getName()))
               {
                  Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_IOT);
                  actionIntent.putExtra(Constants.INTENT_DATA, Constants.INTENT_DATA_PUBLISHED);
                  context.sendBroadcast(actionIntent);
               }
            }
            catch (MqttPersistenceException e)
            {
               Log.e(TAG, "MqttPersistenceException caught while attempting to publish a message", e.getCause());
            }
            catch (MqttException e)
            {
               Log.e(TAG, "MqttException caught while attempting to publish a message", e.getCause());
            }
         }
         else
         {
            connectionLost(null);
         }
      }

      /**
       * Handle loss of connection from the MQTT server.
       * @param throwable
       */
      @Override
      public void connectionLost(Throwable throwable)
      {
         Log.e(TAG, ".connectionLost() entered");

         if (throwable != null)
         {
            throwable.printStackTrace();
         }

         app.setConnected(false);

         String runningActivity = app.getCurrentRunningActivity();
         if (runningActivity != null && runningActivity.equals(LoginFragment.class.getName()))
         {
            Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_LOGIN);
            actionIntent.putExtra(Constants.INTENT_DATA, Constants.INTENT_DATA_DISCONNECT);
            context.sendBroadcast(actionIntent);
         }
      }
      /**
       * Process incoming messages to the MQTT client.
       *
       * @param topic       The topic the message was received on.
       * @param mqttMessage The message that was received
       * @throws Exception  Exception that is thrown if the message is to be rejected.
```

```
     */
    @Override
    public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception
    {
        Log.d(TAG, ".messageArrived() entered");

        int receiveCount = app.getReceiveCount();
        app.setReceiveCount(++receiveCount);
        String runningActivity = app.getCurrentRunningActivity();
        if (runningActivity != null && runningActivity.equals(IoTFragment.class.getName()))
        {
            Intent actionIntent = new Intent(Constants.APP_ID + Constants.INTENT_IOT);
            actionIntent.putExtra(Constants.INTENT_DATA, Constants.INTENT_DATA_RECEIVED);
            context.sendBroadcast(actionIntent);
        }

        String payload = new String(mqttMessage.getPayload());
        Log.d(TAG, ".messageArrived - Message received on topic " + topic
                + ": message is " + payload);
        // TODO: Process message
        try
        {
            // send the message through the application logic
            MessageConductor.getInstance(context).steerMessage(payload, topic);
        }
        catch (JSONException e)
        {
            Log.e(TAG, ".messageArrived() - Exception caught while steering a message", e.getCause());
            e.printStackTrace();
        }
    }

    /**
     * Handle notification that message delivery completed successfully.
     *
     * @param iMqttDeliveryToken The token corresponding to the message which was delivered.
     */
    @Override
    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken)
    {
        Log.d(TAG, ".deliveryComplete() entered");
    }

    /**
     * Checks if the MQTT client has an active connection
     *
     * @return True if client is connected, false if not.
     */
    private boolean isMqttConnected()
    {
        Log.d(TAG, ".isMqttConnected() entered");
        boolean connected = false;
        try
        {
            if ((client != null) && (client.isConnected()))
            {
                connected = true;
            }
        }
        catch (Exception e)
        {
            // swallowing the exception as it means the client is not connected
        }
        Log.d(TAG, ".isMqttConnected() - returning " + connected);
        return connected;
    }
}

/******************MessageFactory*****************/
```

```
/**
 * Build messages to be published by the application.
 * This class is currently unused.
 */
public class MessageFactory {
   private final static String TAG = MessageFactory.class.getName();
   public static String getAccelMessage(float G[], float O[], float yaw, double lon, double lat) {
      String messageData = "{ \"d\": {" +
            "\"acceleration_x\":" + G[0] + ", " +
            "\"acceleration_y\":" + G[1] + ", " +
            "\"acceleration_z\":" + G[2] + ", " +
            "\"roll\":" + O[2] + ", " +
            "\"pitch\":" + O[1] + ", " +
            "\"yaw\":" + yaw + ", " +
            "\"lon\":" + lon + ", " +
            "\"lat\":" + lat + " " +
            "} }";
      return messageData;
   }

   /**
    * Construct a JSON formatted string text event message
    * @param text String of text message to send
    * @return String containing JSON formatted message
    */
   public static String getTextMessage(String text) {
      String messageData = "{\"d\":{" +
            "\"text\":\"" + text.toString() + "\"" +
            " } }";
      return messageData;
   }
   public static String getTouchMessage(double x, double y, double dX, double dY, boolean ended) {
      String endString;
      if (ended) {
         endString = ", \"ended\":1 } }";
      } else {
         endString = " } }";
      }

      String messageData = "{ \"d\": { " +
            "\"screenX\":" + x + ", " +
            "\"screenY\":" + y + ", " +
            "\"deltaX\":" + dX + ", " +
            "\"deltaY\":" + dY +
            endString;
      return messageData;
   }

}

/*******************NODE RED logic code using Node JS in IBM Server*****************/

msg.deviceId = msg.payload.d.deviceId;
msg.deviceType = msg.payload.d.deviceType;
msg.format = "json";

var accelZ = msg.payload.d.acceleration_z;
var r = 0.0;
var b = 0.0;
var g = 0.0;
if (accelZ > 0) {
            g = 255.0;
} else if (accelZ < 0) {
            r = 255.0;
} else {
            r = 104;
            g = 109;
            b = 115;
```

```
}
a = 1.0;
msg.eventOrCommandType = "color";
msg.payload = JSON.stringify({"d":{"deviceId":msg.deviceId, "r":r,"b":b,"g":g,"alpha":a}});

return msg;
```