OULU UNIVERSITY OF APPLIED SCIENCES

Olid Hossain Khan

BLUETOOTH LE (LOW ENERGY) CONNECTION MANAGEMENT

BLUETOOTH LE (LOW ENERGY) CONNECTION MANAGEMENT

Olid Hossain Khan
Bachelor's Thesis
Spring 2015
Degree Programme in
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Olid Hossain Khan
Title: Bluetooth LE (Low Energy) Connection Management
Supervisor: Pertti Heikkilä
Term and year of completion: Spring, 2015 Number of pages: 56 pages

This Bachelor's thesis describes an Android Bluetooth Low Energy connection management collects sensor measurement data frequently and continuously from Bluetooth Low Energy devices to an Android phone. Currently there exists no such application that can connect with Bluetooth Low Energy devices continuously in a very short interval of time. Existing applications are also facing problems to establish a Bluetooth connection with Bluetooth Low Energy devices smoothly.

The target of this research was to develop an Android application which can connect to the *IoLiving* Bluetooth Low Energy devices to read different sensor data like temperature, movement, humidity and Co2.

This study was commissioned by *IoLiving* which is a manufacturer of Bluetooth Low Energy devices in Finland. *IoLiving* Bluetooth Low Energy devices and iteration method including testing in every phase and monitoring Bluetooth data packets were utilized throughout the research.

This report will help a reader to have an overview of the Bluetooth Low Energy connection overview, Android Bluetooth Low Energy application development and testing process and problems regarding a connection establishment in detail.

Keywords: Android, Bluetooth Low Energy, Internet of Things, Sensor, Connect, Database.

PREFACE

Ceruus Oy is a company developing various kinds of IOT (Internet of Things) devices. Matti Verkasalo is the CEO of the company. Ceruus Oy is known as *IoLiving* .The aim of the company is to become a major IOT (Internet of Things) device and Internet service provider. This thesis work was commissioned by *IoLiving*. *IoLiving* is also developing an Android application to communicate with IOT devices using Android phone's Bluetooth stack to retrieve data from IOT devices.

I have been working with *IoLiving* team since March 2014. In the beginning I learned to develop Android applications. I developed various kinds of Android applications. After that I have learned about Android Bluetooth and software design and finally I started my thesis work.

In this thesis Pertti Heikkilä was my instructor. And Kaija Posio was my Language instructor.

Finally I would like to thank Matti Verkasalo(CEO of *IoLiving*), Pertti Heikkilä my instructor , Kaija Posio my language instructor and IoLiving team for such a wonderful support throughout my thesis work.

CONTENTS

VOCABULARY

Abbreviations	Meaning
A2DP	Advanced Audio Distribution Profile
ATT	Attribute Profile
BLE	Bluetooth Low Energy
BR	Basic Rate
EDR	Enhanced Data Rate
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
HDP	Health Device Profile
ID	Identification
IDE	Integrated Development Environment
IoT	Internet of Things
LE	Low Energy
MAC	Media Access Control
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
SIG	Special Interest Group
UUID	Universally Unique Identifier

1 INTRODUCTION

Internet of thing is an object or device or entity which has a unique identification number or address and which has an ability to measure various kinds of data, like e.g. temperature, heart rate, and which can send data over a network without the help of a computer.

Bluetooth is a technology which can exchange data in short distance wirelessly. Using Bluetooth, phones, computers or IoT(Internet of Things) devices can be interconnected wirelessly to exchange data between them. Bluetooth LE (Low Energy) is a version of Bluetooth technology which can communicate or exchange data between devices consuming very little energy. (Huang, F 2013,21-23)

Ceruus Oy known as *IoLiving* is a company which is producing IoT devices which can measure temperature, movement, humidity, Co2, etc. Those IoT devices are saving those measurement values inside their own memory. Then an Android application, which is using Bluetooth LE, is connecting to IoT devices and retrieving data from IoT devices and storing that data to the local memory of the phone. After that data is sent to the cloud server to use the data in a *IoLiving* web service for further consumer uses.

This thesis will focus on how to develop an Android application which can detect IoT devices and how to connect with IoT devices and how to retrieve data from IoT devices and how to make that data usable for a further use. This thesis will describe problems to establish a Bluetooth LE connection with IoT devices and steps to avoid problems.

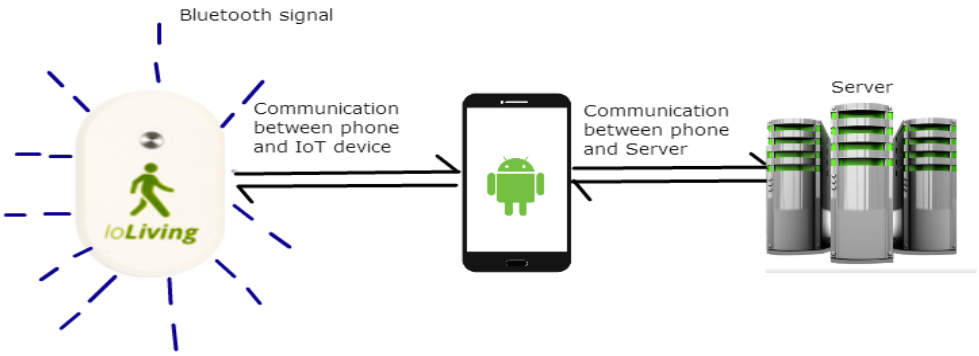

FIGURE 1.IoT device communication with Android phone in brief (Android Smart phone icon, Date of Retrieval 26.04.2015 , IoLiving Device, Date of Retrieval 26.04.2015 , cloud-server, Date of Retrieval 26.04.2015)

1.1 Internet of Things

Internet of Things consists of things which has physical existence and can connect to the internet, e.g. machine, vehicle, building, people, and almost everything is part of things. Internet of things must have a unique identity, so that each and every entity can be separated easily. Internet of Things must have the ability to communicate with other object. Internet of Things needs to have senses which will send information about the entity. Internet of Things can be controlled from anywhere. If any object has the above feature, then it can be called as Internet of Things. (Barrett, J., Internet of Things, Date of Retrieval 21.05.2015)

1.2 Bluetooth

In our daily life we are using many devices like computer, mobile phone, television. Devices are mainly connected by cables. So it becomes a demand of the time to use such a technology through which we can connect to different devices without any cables. Bluetooth wireless technology is a new Radio Frequency transmission standard which can be used to connect to different devices in a short range. Bluetooth technology opens the door to connect to billion devices. (Ali M Aljuaied 2001. 19-20)

1.3 Bluetooth SIG

Bluetooth SIG (Special Interest Group) is a non-profit organization founded in 1998. It has more than 20,000 member companies worldwide. (Bluetooth SIG, Date of Retrieval 21.05.2015)

In 1994 Ericsson Mobile communications started a study to find an alternative of cables which could link their mobile phones with accessories. This study preferred to use radio links because it had an advantage of complete directional transmission. Ericsson realized that this technology is powerful and has a big potential in it. They realized the necessity of an open and common specification. From this realization Special Interest Group was found. (Ali M Aljuaied 2001. 19-20)

There are seven promoter level member companies in Bluetooth SIG: Ericsson, Intel, Lenovo, Microsoft, Motorola Mobility, Nokia and Toshiba. (Bluetooth SIG, Date of Retrieval 21.05.2015)

1.4 Research and Development method

This thesis has followed an iterative method for its research and development process. The iterative method generally divides the requirements into smaller pieces and then implements those pieces one after another. So iterative processes divide the whole software into smaller pieces and develop one piece at a time and test it properly. And then it develops a new piece of software and adds it with the existing one. This process continues until it reaches its destination.

The iteration process was planned according to the requirements. Some of those requirements are, to initialize the Bluetooth for an Android application, to scan for the available device, to separate *IoLiving* devices from all devices, to receive broadcast data, to break broadcast data for further use , to save broadcast data into a database, to make a connection priority list considering all available catchers, to send a connection request for the highest priority device, to connect to the Bluetooth GATT(Generic Attribute Profile) server, to discover services, to read measurements from the device, to save those measurements into a phone's database, to send a disconnect request, to disconnect from a Bluetooth GATT server(if it is still not disconnected, then forcefully disconnect from a device) , to display received data in the phone view or display that data in a web service.

The Design of the Iteration model:

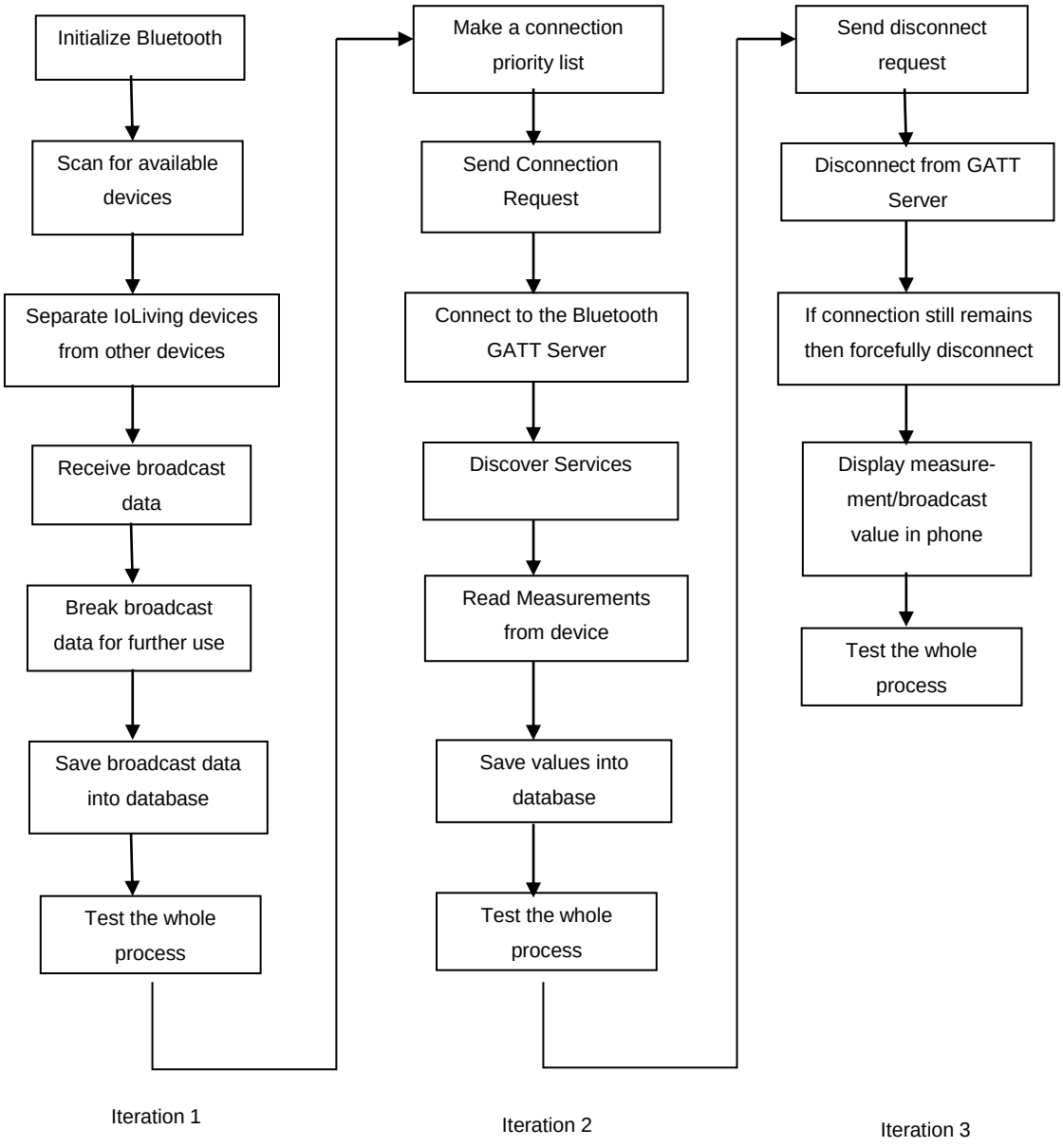

FIGURE 2. The iteration flow

2 KEY TERMS AND CONCEPTS

2.1 Eclipse

Eclipse is an IDE (Integrated Development Environment) which allows a developer to develop Java based applications like an Android application. Eclipse has an extendable plug-in system which allows a developer to customize the development environment. Eclipse was used in this thesis to develop the Android application.

2.2 Android SDK

Android SDK (Software Development Kit) is a software development kit which allows a developer to develop an Android application. Eclipse needs to install Android SDK to enable the developer to develop an Android application.

2.3 Android Manifest

Android Manifest is an xml file. Every Android application contains this file. This file describes the requirement of the application. This file also uses a declare permission in order to use Bluetooth in this application.

2.4 Bluetooth Profile

Bluetooth profile is a specification which specifies the general behaviour of a Bluetooth enabled device while communicating with other Bluetooth enabled device. A Bluetooth device must be able to interpret a certain Bluetooth profile in order to communicate with other Bluetooth enable device. Bluetooth profile specifies many different kinds of uses and applications which can be used by a developer to communicate with Bluetooth enable devices

A few Bluetooth profiles are mentioned below:

1. Advanced Audio Distribution Profile (A2DP)

2. Attribute Profile (ATT)

3. Generic Access Profile (GAP)

4. Generic Attribute Profile (GATT)

5. Health Device Profile (HDP)

(Bluetooth Profile, Date of Retrieval 21.05.2015)

2.5 GATT Profile

(Bluetooth GATT Profile, Date of Retrieval 21.05.2015)

Generic Attribute Profile which is known as a GATT profile is a Bluetooth Specification which uses Attribute Protocol (ATT) to define a way to communicate between Bluetooth LE devices. GATT defines two roles of Server and Client. GATT profile can be used both for BR (Basic Rate)/EDR (Enhanced Data Rate) and LE (Low Energy). GATT and ATT profiles are needed to discover services in the LE device communication. In GATT the Server and Client relationship is important. An IoT device plays the role of GATT Server and Android phone plays the role of GATT Client. The Android phone which acts as a Master starts the communication and the IoT device which acts as a slave gives a response when there is a request.

GATT Server stores all the data which has been transported using Attribute Protocol. Attribute Protocol is formatted as services and characteristics where services contain characteristics and characteristics contain any number of descriptors that describes the characteristic values.

The following picture shows how GATT Client (master) and GATT Server (slave) exchange data between them.

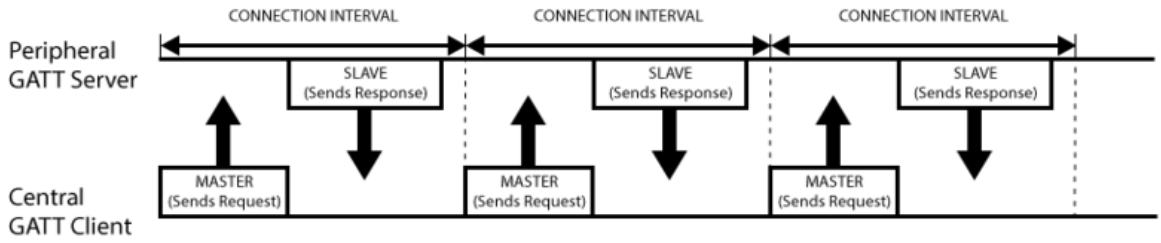

FIGURE 3. The data exchange between Master and Slave in GATT server (Bluetooth GATT Profile, Date of Retrieval 21.05.2015)

GATT Profile contains mainly two elements service and characteristics. The service contains characteristics and the characteristics contain its value, descriptor, additional information etc. In general, the profile contains services and services contain characteristics and each characteristic contains its values and additional description. The following picture will show the hierarchy of GATT Profile.

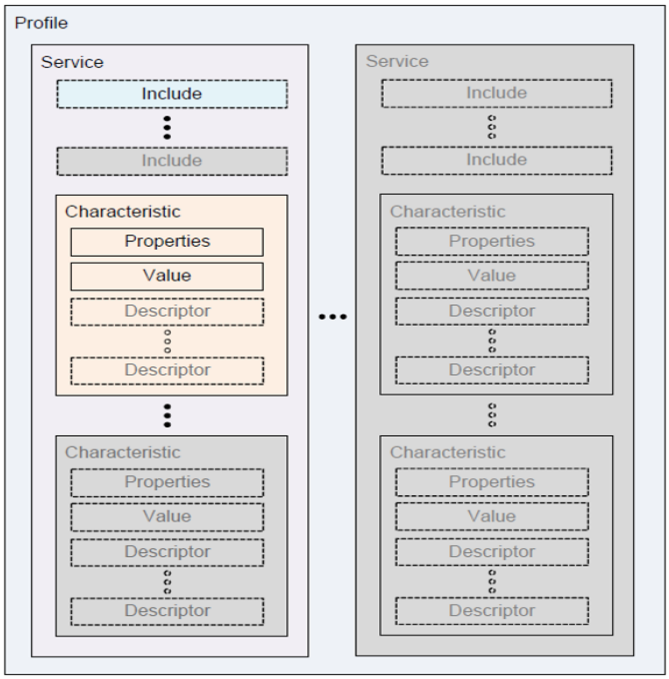

FIGURE 4. Hierarchy of GATT Profile (Bluetooth GATT Profile, Date of Retrieval 21.05.2015)

2.6 Services

A service is a set of data and related behaviour to perform a particular function for a device. Every service has its own id called UUID (Universally Unique Identifier). The UUID can be either 16 bit (official BLE service) or 128 bit (custom service). There are mainly two kinds of service available primary and secondary. The primary service provides a primary functionality of a device and the secondary service provides an additional service if any primary service refers to a secondary service. It could be possible to build a nested reference of services. The service also contains characteristics. (Bluetooth GATT Profile, Date of Retrieval 21.05.2015)

2.7 Characteristics

Characteristics actually hold the data. A declaration and a value are the main attributes of characteristics. A single characteristic is a bundle of declaration, value and any descriptor attribute. (Townsend, Devidson & Akiba, Chapter4, Date of Retrieval 21.05.2015)

2.8 UUID

UUID (Universally Unique Identifier) means a universally unique identifier which is a 128 bit (16 byte) number. Bluetooth and other protocols are using a UUID number. In Bluetooth specification a 16-bit or a32-bit UUID format are also available. These shortened UUIDs can only be used if Bluetooth Specification defines them. (Townsend, Devidson & Akiba, Chapter4, Date of Retrieval 21.05.2015)

2.9 HCI Log

Android provides a feature which can store all communication Log of Bluetooth in one file HCI (Host Controller Interface) log. To activate this feature Android Developer Option needs to activate first. To activate Android Developer Option, first you need to go to settings, you need to click about option of the phone, and then you need to press the Build number 7 times.

3 TOOLS USED IN DEVELOPMENT AND TESTING

3.1 Name of Tools

Several tools were used in this development phase. Eclipse was mainly used in this application development. Before starting the development, Eclipse needs to be setup properly. Android SDK needs to be installed in Eclipse in order to enable the Development environment. An Android phone, which has Bluetooth 4.2 stack in it, is required for the development. For testing purpose Bluetooth communication needs to be monitored. WireShark was used to read HCI Log which will provide details about Bluetooth Communication. Pixlr was used as a photo editing software. Command prompt was used to see Android Logs. StarUml and Dia were used to design the software.

A few tools, which were used in development of the application, are listed below:

1. Eclipse
2. Android phone
3. *Io**Living*** Device
4. Pixlr
5. WireShark
6. Notepad++
7. Es File Explorer
8. Command Prompt
9. StarUml
10. Dia

4 DESIGN OF THE SOFTWARE

4.1 Basic design of the software

The design of the software is described in the three pictures below. This design has been changed a little bit according to the demand of the situation. The whole application will run in the background and there will be mainly two cycles. In cycle one the application will run the Bluetooth communication loop and in cycle two the application will run the server sync loop.

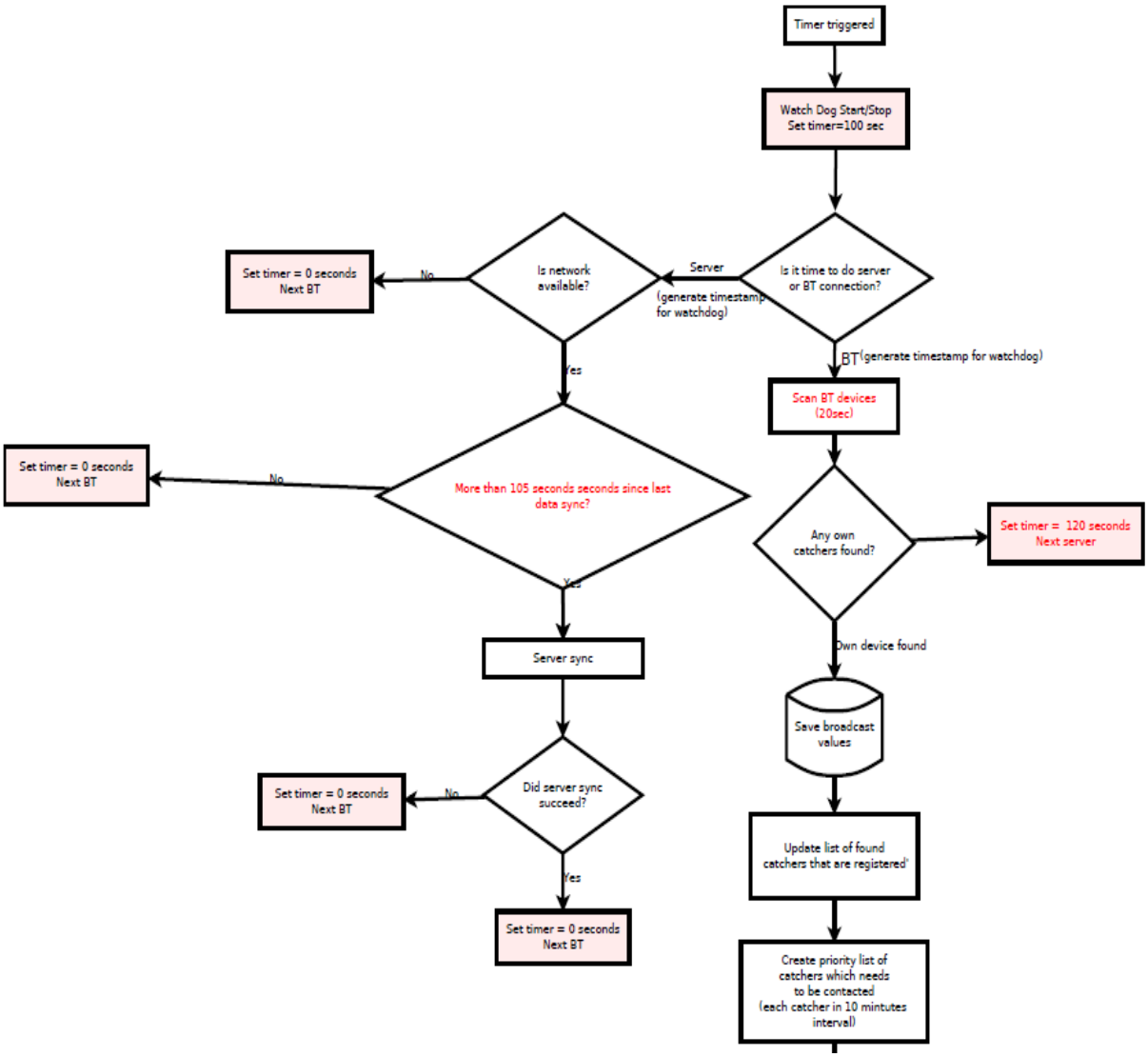

FIGURE 5. Basic Design of Software part 1 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

There will be one timer called Watch Dog which will check if the application stucks in any process. If the application stucks in any process, it will stop the process and start a new cycle.

In the picture one it can be seen that Timer triggered is the starting point of the application. When the application starts, Watch Dog will also start. The first thing the application will decide is whether it will run a Bluetooth connection loop or a server communication loop. Bluetooth needs to initialize before it will be ready to use, therefore, the server communication task will be the first loop to start the process. When it will go to the server communication loop it will first check whether there is any internet available. If there is no Internet that it will exit and call the next process to the Bluetooth connection loop. If there is an available Internet, then it will check whether the last server communication happened in 105 (currently 60 seconds) seconds. If the last server communication happened in 105 (currently 60 seconds) seconds, then it will call the next task to the BT connection task. Otherwise, it will do the server communication task. If the server responses status 200(OK status), it will call the next task to the BT connection task. Even if the server does not response status 200(OK) it will call the next task to the BT connection task.

The next loop is the Bluetooth connection loop. In this loop the application will first scan for available devices. If there is an available device, then it will check whether it is a *IoLiving* device or not. If there is no device available or the device is not a *IoLiving* device then the process will exit and call the server communication loop. If there is a *IoLiving* device available, then it will save the broadcast value and it will update the list of catcher in every scan. Then, it will make a priority list where it decides which device will connect first based on the previous connection time. If one device connected 10 minutes (currently 6 minutes) earlier, then that device will get the highest priority to make the next connection. If there is no device in the priority list the application will exit the process.

In the second picture the next part of the design is described. If there is any device found which has a priority, it will start the connection process, otherwise it

will exit the loop and it will go to the server communication loop. Next, it will check whether the connection process is successful or not. If the connection is not successful, then it will exit the loop. But if the connection is successful, it will start reading or writing from the device.

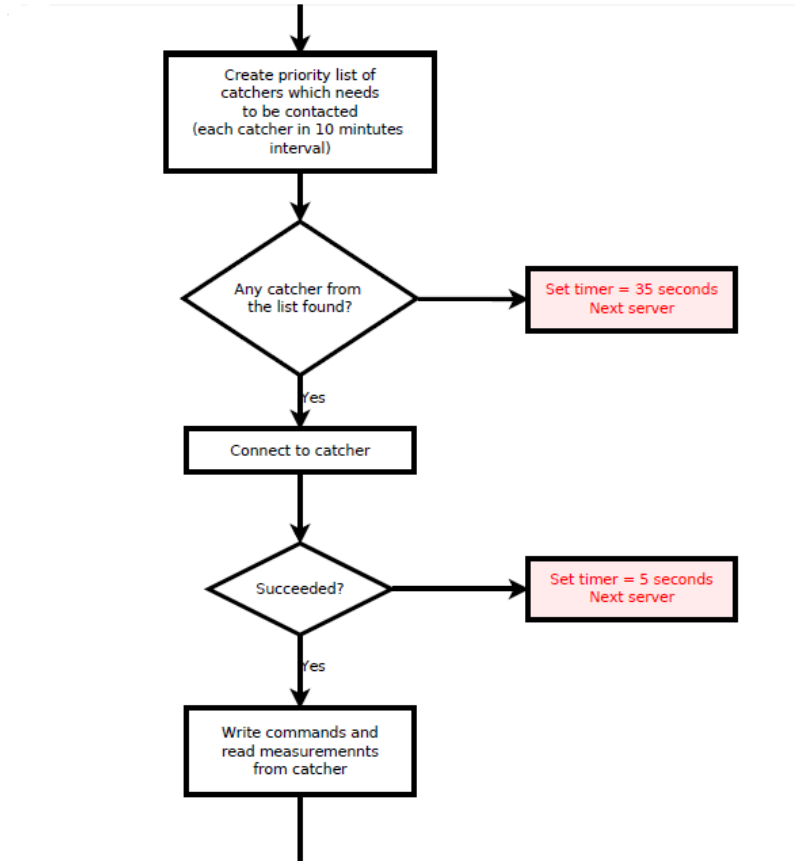

FIGURE 6. Basic Design of Software part 2 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

The rest of the design is described in the third picture. In this stage it will try to read or write the measurement from the device. If it stucks in the measurement reading or writing process, then Watch Dog will activate after around 100 seconds and it will forcefully disconnect from the device. If the application reads the measurements successfully, then it will save the measurement data to the local database. Then, it will exit the process and it will call the server communication task. In every exit point the application will wait 5 to10 seconds before starting a

new loop. This delay will give the application some time to settle down. The whole process will repeat again and again.

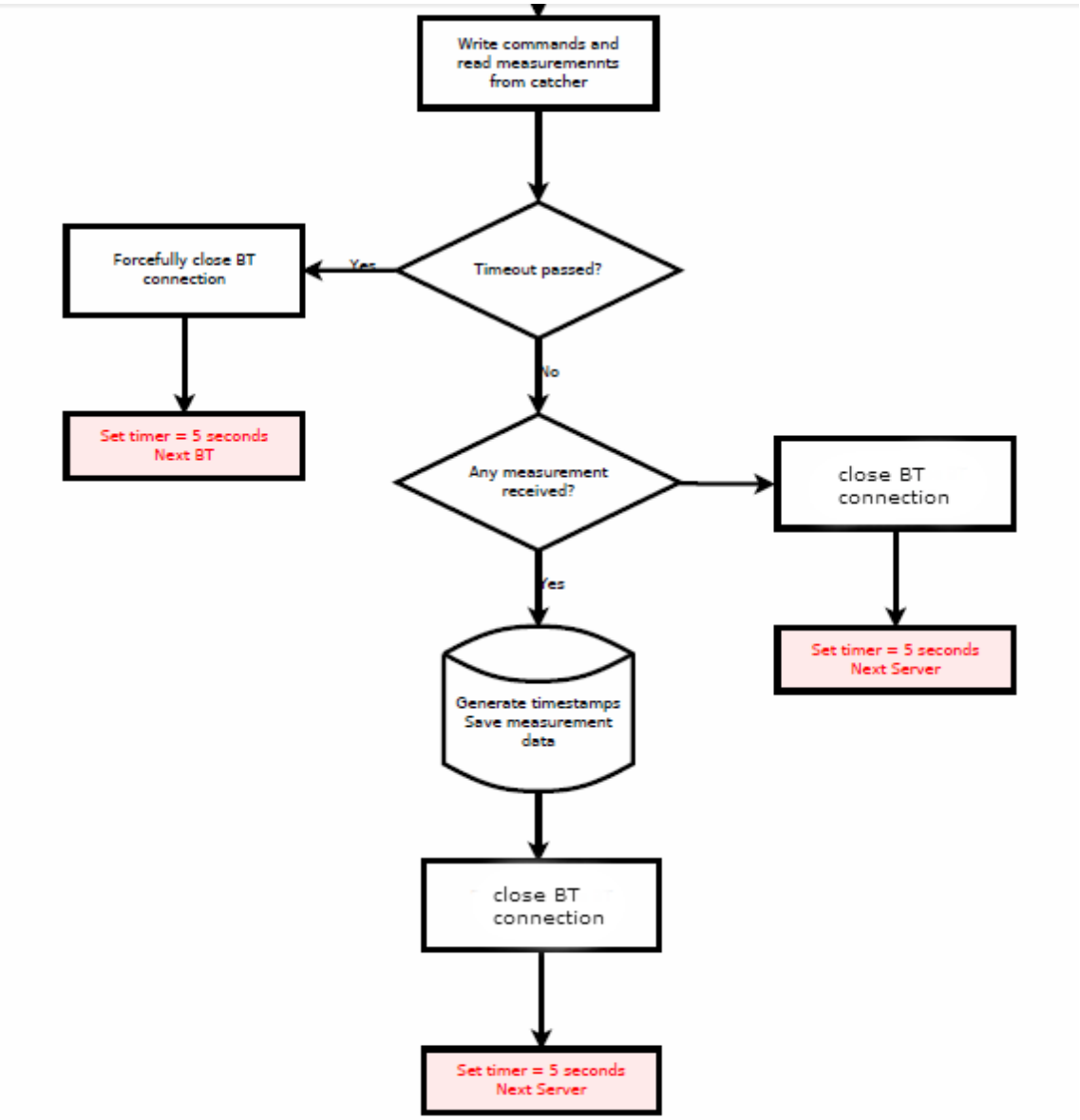

FIGURE 7. Basic Design of Software part 3 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

4.2 Life Cycle of the application

The below diagram will show the lifecycle of the application. The application will take approximately 1 to 3 seconds to start its process. To complete the server communication, it will take 1to 5 seconds. The application will scan for 20 seconds, then it will connect to a device it will take around 20 seconds to compete the communication process. After the successful communication is done, it will disconnect the Bluetooth connection. It will take around 5 to 25 seconds to complete the disconnection process. If the application stucks in any process around for 100 to160 seconds, Watch Dog will activate to kill the process and it will call the next process.

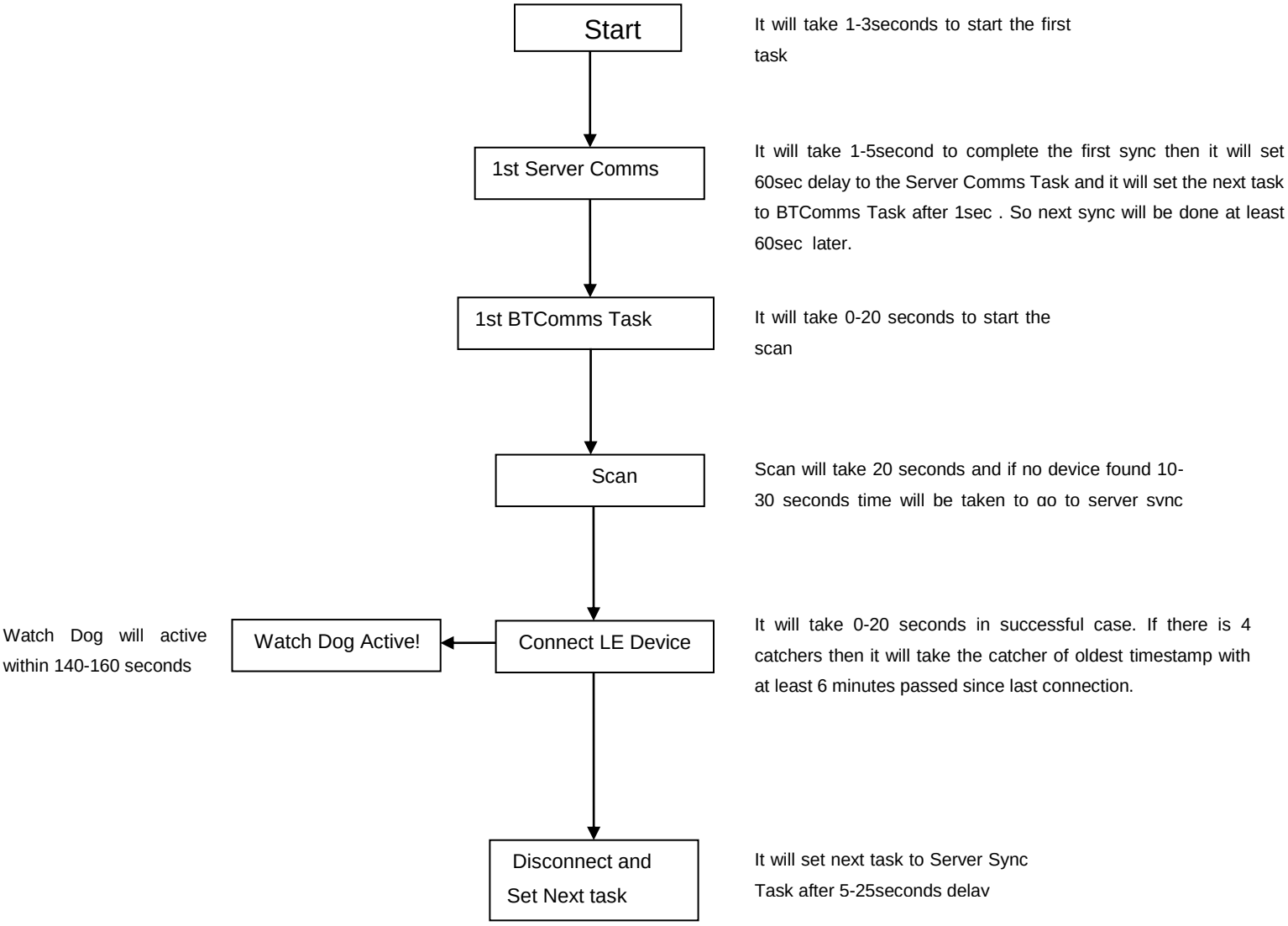

Figure 4.4: Application Life Cycle

4.3 Class diagram

This part describes the classes that were used to make a Bluetooth connection. Many different classes were used to complete the Bluetooth connection task. In every class several methods were used to complete the task. Below in the diagram it can be seen how these classes are working.

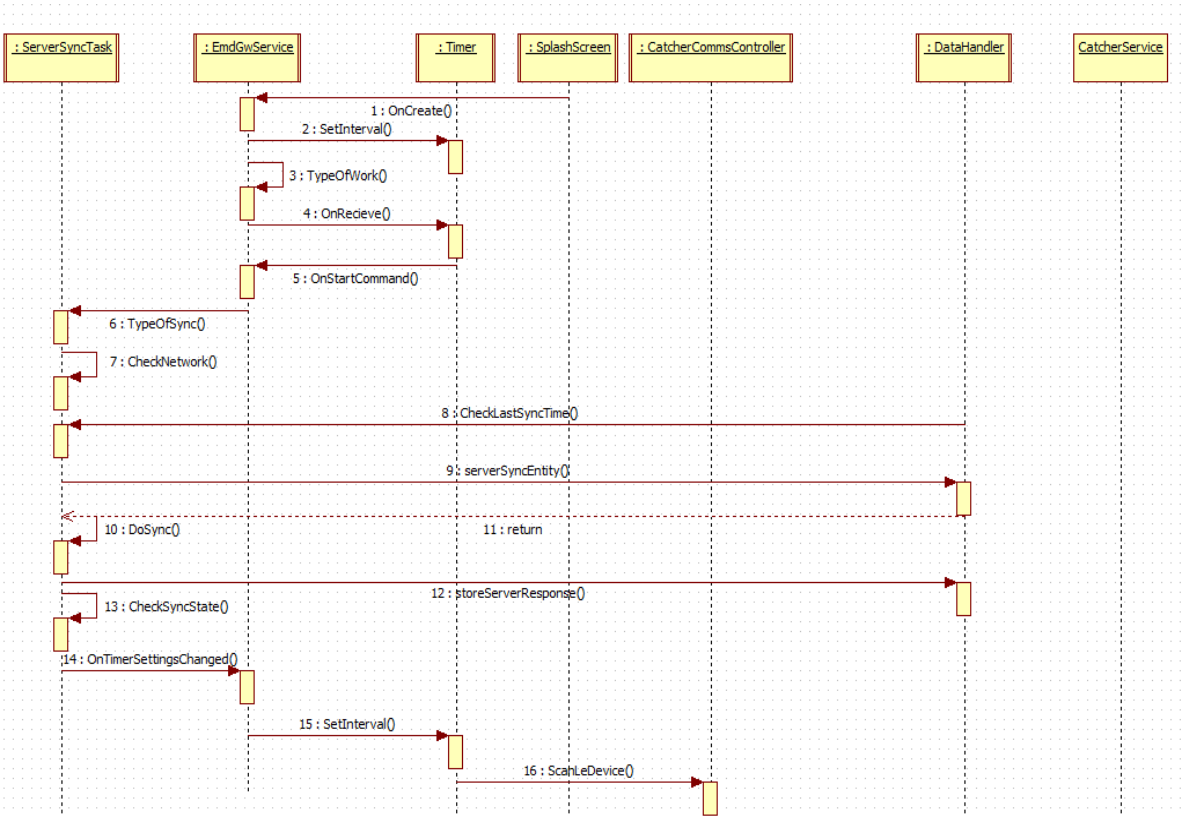

FIGURE 9.Class diagram part 1 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

The EmdGwService class will start the proceedings by calling the ServerSyncTask class to do a server sync. When the server sync will be done, it will call the next task as a Bluetooth connection task. This task will be handled by the CatcherCommsController class. The CatchreService is helping the CatcherCommsController class to complete the Bluetooth connection. The DataHandler class is saving all kinds of data as it is working as a local database.

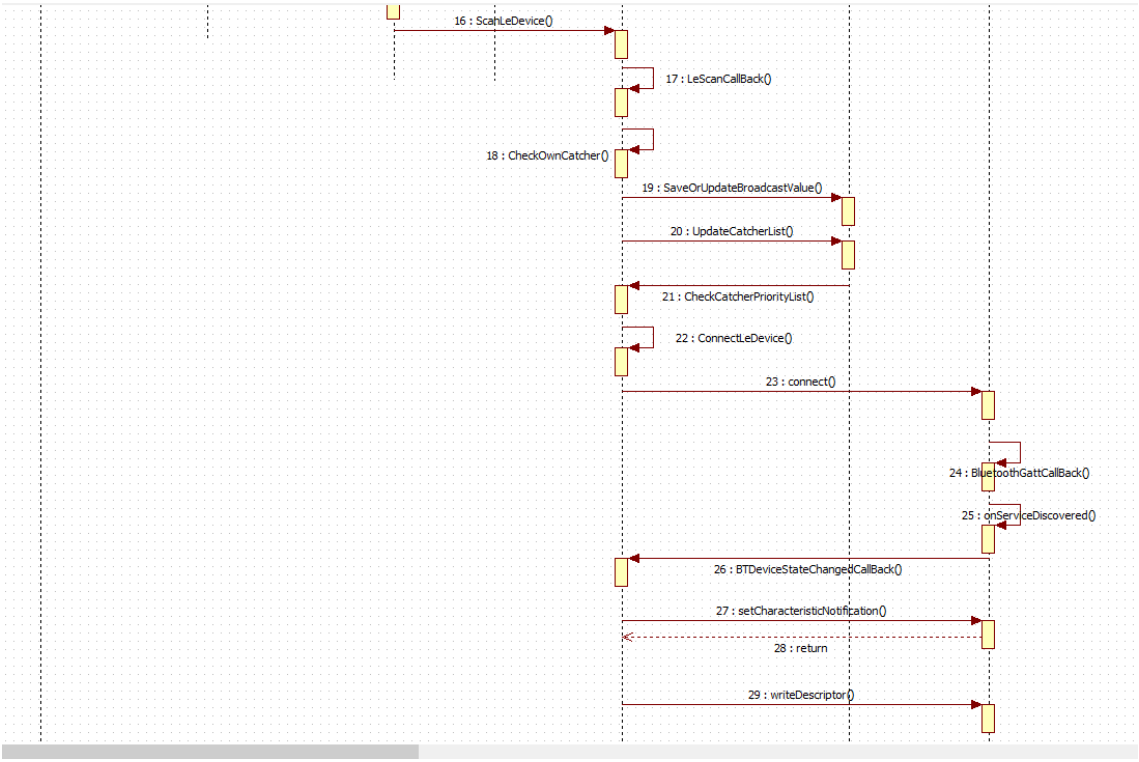

FIGURE 10.Class diagram part 2 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

Other classes like CatcherBroadcastDataBreaker, CatcherAdvertisingData, CatcherMeasurementStream, CatcherGattAttribute are helper classes to complete a Bluetooth connection process. The CatcherBroadcastDataBreaker class will break broadcast data from a byte to a string to display a temperature value in the application's user interface. The CatcherAdvertisingData class handes information that a *IoLiving* device or other BT device sends as broadcast. The CatcherMeasurementStream class was used to parse a byte stream sent from the *IoLiving* device and this class will also help to persist the measurement for later use. The CatcherGattAttribute class was used to define a BT-related constant which will later be needed to complete a Bluetooth connection.

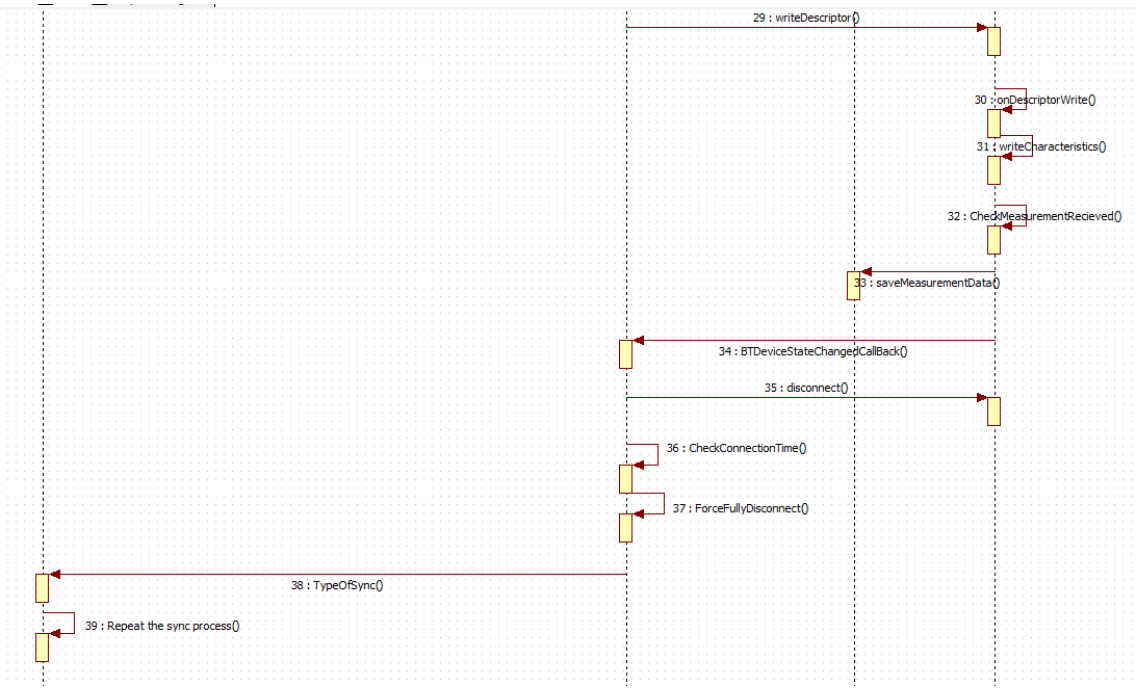

FIGURE 11.Class diagram part 3 (IoLiving documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015)

This diagram is showing the main functionality of the application. There are many other methods that were also used in this application but they are not mentioned in the diagram. Only the important methods are mentioned here.

The Bluetooth connection process was started by doing the Bluetooth scan. scanLeDevice is the method used in this application to call scan process. The LeScanCallBack method is returning the scan result. The CheckOwnCatcher method is checking whether the device is a *IoLiving* device or not. The SaveOrUpdateBroadcastValue method is saving the broadcast value to the database. The UpdateCatcherList method is updating the list of available devices around the Android phone. The CheckCatcherPriorityList method is checking which device should connect first. This method is deciding the high priority device. The ConnectLeDevice method was used to prepare the connection pro-

cess. The connect method was used to establish a connection. The BluetoothGattCallBack method decides whether the device is able to establish a connection with the GATT server. If the GATT server connects successfully, onServiceDiscovered is called when services, characteristics and descriptor for the device have been updated. BTDeviceStateChangedCallBack is a function which carries the information of the current state to the Bluetooth connection process and this method helps to decide what to do next according to the current state situation. The setCharacteristicsNotification method helps enabling or disabling the notification for a given characteristic. The WriteDescriptor method was used to initiate the descriptor writing to Bluetooth. After the writing operation is complete a separate callback method will be called by the Bluetooth stack. The WriteCharacteristic method was used to initiate the characteristic writing into a Bluetooth device. The CheckMeasurementReceived method was used to check whether there is enough notification in the measurement stream. The saveMeasurementData method was used to save the measurement reading from the device. The BTDeviceStateChangedCallBack method was used to notify the application that measurement reading has been ended. The method disconnect was used to disconnect the device from the application. The CheckConnectionTime method was used to check whether the application stucks in the connection state. If the application stucks in the connection state, then the forcefullyDisconnect method will destroy the CatcherService class to disconnect the device forcefully.

5 FIRST DEVELOPMENT PHASE

The whole development was divided into three phases. In the first phase an Android application needs to initialize Bluetooth, to scan for available devices, to separate *IoLiving* devices from other devices, to receive broadcast data, to break broadcast data for further use, to save broadcast data into a database and to test the whole process. The mentioned steps need to be implement one after another.

5.1 Initialize Bluetooth

In order to perform any Bluetooth communication first it needs to put permission to the application. The permission needs to declare in the Android Mainfest file of the application as below:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

The target of this application is to use BLE only. Another permission needs to add in the Android Mainfest in order to make it BLE capable only.

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

The next thing is that the application should check whether the device supports BLE or not. If BLE is not supported by the device then this application will not work. If BLE is supported by the device, then the application needs to check whether the Bluetooth is enabled or not. If Bluetooth is not enabled in the phone, this application should turn the Bluetooth on.

To enable Bluetooth for the application, first a Bluetooth adapter is required. Without a Bluetooth adapter no Bluetooth related activity is possible. The following code is necessary to initialize the Bluetooth adapter.

```
/ Initializes Bluetooth adapter.
final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
```

The next important task is to make it sure that Bluetooth is enabled in the phone. There is a method called isEnabled() to check whether it is enabled in the phone or not. If this method returns true, it means that Bluetooth is enabled in the phone but if it returns false it means that Bluetooth is not enabled in the phone. If Bluetooth is not enabled then, a dialog needs to show to the user requesting to enable Bluetooth. The following code will do this work.

```
private BluetoothAdapter mBluetoothAdapter;
...
// Ensures Bluetooth is available on the device and it is enabled. If not,
// displays a dialog requesting user permission to enable Bluetooth.
if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

5.2 Scan for a Bluetooth Low Energy device

Bluetooth Initialization has been done now. The next task is to scan for an available Bluetooth Low Energy device. To make a scan happen, it is necessary to call a method startLeScan(). This method takes a parameter with it called BluetoothAdapter.LeScanCallBack. To get a Bluetooth scan result, this callback is necessary. In the Bluetooth scan it is possible to set the scan period. This application has now a 20 seconds scan period. This scan period time is variable so it can be changed at any time according to the need of the application. After the scan is complete, the stopLeScan() method needs to call to stop the scan process. The following code was implemented to start and stop the scanning process.

```
private void scanLeDevice(final boolean enable) {
                                Log.v("BtGw", "scanLeDevice in");
                                if (enable) {
                                                // Stops scanning after a pre-defined scan period.
                                                mHandler.postDelayed(new Runnable() {
                                                                @Override
                                                                public void run() {

                                                                                mBluetoothAdapt-
er.stopLeScan(mLeScanCallback);
                                                                                conditionallyCon-
nectOldestBTDevice();
                                                                }
                                                }, 20*1000 );

                                                mBluetoothAdapt-
er.startLeScan(mLeScanCallback);

                                } else {
                                                mBluetoothAdapt-
er.stopLeScan(mLeScanCallback);

                                }
                                Log.v("BtGw", "scanLeDevice out");
                }
```

To get the scan result a BluetoothAdapter.LeScanCallback method is required. This call back method will return the scan result which will include a device Mac address, an RSSI (Radio signal strength) and a scan record. The scan record is a byte array of broadcast value. This broadcast value can contain lots of information e.g. company ID, temperature. The following piece of code is needed to implement the callback of scan.

```
private BluetoothAdapter.LeScanCallback mLeScanCallback = new BluetoothAdapt-
er.LeScanCallback() {

                                /**
                                 * Method that is called when a device is found during BT scan.
                                 */
                                @Override
                                public void onLeScan(final BluetoothDevice device, final int rssi,
final byte[] scanRecord) {
                                                Log.v(TAG, "onLeScan in");

                                                synchronized (DataHandler.class) {
                                                                mData-
Handler.updateCatcherRSSI(device.getAddress(), rssi);
                                                                mData-
Handler.updateBroadcastData(device.getAddress(),System.currentTimeMillis(), scanRecord);
```

```
                                        }
                                        synchronized (DataHandler.class) {
                                                for (Pair<BluetoothDevice, Integer>
d : mBTDevicesSeen)
                                                        if
(d.first.getAddress().equals(device.getAddress())) {

                mBTDevicesSeen.remove(d);
                                                        }
                                                mBTDevic-
esSeen.add(Pair.create(device, rssi));
                                        }
                                }
                };
```

5.3 Separate a IoLiving device from other device

The next task is to separate a *IoLiving* device from other devices. The broadcast value of *IoLiving* devices contains 4-digits company ID (Identification) which will help to identify if the device is *IoLiving* device or not. The first broadcast value needs to break down to get the company ID. The CatcherBroadcstDataBreaker class was used to break the broadcast data. After comparing the value broadcasted by the device with the company ID, it is easy to determine whether the device is a *IoLiving* device or not. The following sample code shows how to implement this functionality.

```
String compnayID = "";
if (scanRecord.length > 00) {
compnayID = String.format("%02x ", scanRecord[00])
                                + String.format("%02x ", scanRecord[00]);
// Log.e("test", String.valueOf(compnayID));

}

if (deviceAdd != null
```

```
&& deviceAdd.length() > 0
&& (compnayID.startsWith("00 00")) {

.....

}
```

5.4 Break broadcast data

The next task is to break broadcast data to be used for various purpose like a display temperature in the user interface. The CatcherBroadcastDataBreaker class was used to break the broadcast data. Broadcast data comes as a byte array. In this byte array a different position holds a different value like a temperature value, a broadcast frame ID, a company ID etc. These values were used in different places and for different purposes throughout the application. The following code contains an example of how to get a temperature value from the broadcast data using one method of the CatcherBroadcstDataBreaker class.

```
public double getCurrentTemperature() {

double value = (this.mManufacturerData[4] << 8 ) |
(this.mManufacturerData[3] & 0xff);
return value/10.0;
}
```

5.5 Save Broadcast data

The last thing to do in iteration 1 is to save the broadcast value. The Android SQLite (Database name) database was used in this application. A database declaration, a table structure and all methods that help to store or retrieve data are in the DataHandler class. The saveOrUpdateBroadcast value is the method used to save the broadcast data. Three parameters were used in this method. Parameters are a device ID, a scan record or broadcast value and time of sav-

ing the broadcast value. Inside this method a device registration was also checked. The code below will show how to use a method to save a broadcast value in the SQLite database.

```
public boolean saveOrUpdateBroadcastValue(String deviceId, long broadcastTimestamp,
byte[] broadcastValue) {
                                boolean retval = false;
                                SQLiteDatabase db = getWritableDatabase();
                                ContentValues values = new ContentValues();
                                values.put("broadcast_timestamp", broadcastTimestamp);
                                values.put("broadcast_data ", broadcastValue);
                                if(db.update("catcher", values, "catcher_id = ?", new String[] {
deviceId })==1){
                                                retval=true;
                                }
                                else{
                                                Log.d(TAG,"updateBroadcastData failed for un-
regiseterd catcher");
                                }
                                return retval;
                }
```

5.6 Test 1st Iteration processes

To start the second iteration process, it needs to test the first iteration process. A detail of all tests is described in a separate chapter. A long term and short term tests were run to make sure all processes are working fine in a different condition. An Android application crashes if there is any fatal exception. Therefore a long term test was needed to run to see whether the application crashes in any condition. The initial test showed that the application crashes for “null point exception”. After getting the exception report from the eclipse, all kind of exception has been handled. So the application is now ready for the second iteration development.

6 SECOND DEVELOPMENT PHASE

After completing the first development phase, it is now possible to list all available Bluetooth Low Energy devices. The target of the second development phase is to establish a Bluetooth connection successfully. Initial targets of this iteration are to make a priority list, to send a connection request, to connect to a GATT server, to discover services, to read measurements from the device, to save a measurement value and to test the whole process. After this iteration, the application will be able to read measurements from the device.

6.1 Make a priority list for connection

After scaning all devices are available now. So it is now time to decide which device will connect first. This priority list is made during the connection time. If any device did not connect during the last 6 minutes (10 minutes previously) ,then the device has a priority to make a connection. If more than one device did not connect during the last 6 minutes (10 minutes previously), then the device which has the oldest timestamp in milliseconds will get the highest priority. If the device is connecting for the first time, then the priority will be selected randomly. The following code will show how to sort one device from all devices.

```
BluetoothDevice deviceWithOldestMeasurement = null;
long oldestMeasurement = Long.MAX_VALUE;
synchronized (DataHandler.class) {

for (Pair<BluetoothDevice, Integer> d : mBTDevicesSeen) {

if (mDataHandler.checkOwnCatcher(d.first.getAddress())) {

long measurementReadTime = mDataHandler.getMeasurementReadTime(d.first.getAddress());
if (oldestMeasurement > measurementReadTime) {

oldestMeasurement = measurementReadTime;

if (measurementReadTime + Config.minConnectionInterval < System.currentTimeMillis()) {

                deviceWithOldestMeasurement = d.first;

}
                } else {

                Log.v(TAG, "Candidate " + d.first.getAddress() + " was more recent than others");
                }
```

```
                } else {
                Log.v(TAG, "Skipping candidate " + d.first.getAddress());

                                }
                                                }
```

6.2 Send a connection request

A method was used to send the connection request called connectLeDevice. This method has three parameters a device address, a Boolean parameter and a context. This same method was used to send a disconnect request, too. This Boolean parameter will decide whether it will connect or disconnect. This method will take the device address as a parameter. Then it will create an intent of the Catcher Service class. Then gattServiceIntent and serviceConnection are used to bind the service. This method has also stores the connection time.

```
connectLeDevice(deviceWithOldestMeasurement,true,mOwningService.getBaseContext());

                private void connectLeDevice(final BluetoothDevice device,
                                                final boolean enable,
                                                Context context ) {

                                if (enable) {
                                                Log.e("BtGw", "connectLeDevice() enable" + de-
vice.getAddress());
                                                mDeviceAddress = device.getAddress();
                                                Intent gattServiceIntent = new       In-
tent(context,CatcherService.class);
                                                isBound=       con-
text.bindService(gattServiceIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
                                                mConnected = System.currentTimeMillis();
                                } else {
                                                if ( mCatcherService != null ) {
                                                                mCatcherService.close();
                                                }
                                                if ( mOwningService.getBaseContext() != null &&
mServiceConnection != null ) {

                                                                if (isBound){
                                                                                mOwningS-
ervice.getBaseContext().unbindService(mServiceConnection);
                                                                                isBound=false;
                                                                }
                                                                else{

                                                                                //do nothing.
```

```
                                                    }
                                                }
                                                mDeviceAddress = null;
                                                mDevice = null;
                                                mConnected = -1;
                                                Log.e("BtGw", "connectLeDevice() out after discon-
nect " );
                                            }
                                        }
```

6.3 Connect to a GATT server

The next task is to connect to the GATT server on the device. A method connect was used to connect to the GATT server. Another method connectGATT was called from the inside of this method actually used to connect to the GATT server. This method takes three parameters, a context, a Boolean, which will decide whether the device will connect automatically or not and a reference to BluetoothGattCallBack. When the connection request is sent, a BluetooGatt instance is returned in reply. This is used later on to conduct GATT client operations. BluetoothGattCallback is used to deliver the results to the client.

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

```
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {

                                    /**
                                     * Callback method that BT stack calls when GATT server at de-
vice is connected/disconnected
                                     */
                                    @Override
                                    public void onConnectionStateChange(BluetoothGatt gatt, int
status, int newState) {
                                                if (newState == BluetoothPro-
file.STATE_CONNECTED) {
                                                                CatcherCommsControl-
ler.getInstance().BTDeviceStateChangedCallBack(ACTION_GATT_CONNECTED);

                        Log.d(TAG, "Connected to GATT server.");
if (mBluetoothGatt != null) {

                        boolean result = mBluetoothGatt.discoverServices();
Log.d(TAG, "mBluetoothGatt.discoverServices() result = " + result);

} else {
```

```
Log.w(TAG, "onConnectionStateChange() => mBluetoothGatt == null");
                                                                    }

} else if (newState == BluetoothProfile.STATE_DISCONNECTED) {

CatcherCommsControl-
ler.getInstance().BTDeviceStateChangedCallBack(ACTION_GATT_DISCONNECTED);

Log.d(TAG, "Disconnected from GATT server.");
                                                    }
                                        }
```

6.4 Discover Services

The device is now connected to the application. The callback function BluetoothGattCallback.onConnectionStateChange will be called with a new state. This callback will set the argument to BluetoothProfile.STATE_CONNECTED. The discover Service can be initiated now. This call will decide whether the services are supported by the device. If the services are supported by the device, then a callback BluetoothGattCallBack.onServicesDiscovered will be received. The following code is an example of how to discover services.

```
bluetoothGatt.discoverServices();
```

```
for (BluetoothGattDescriptor descriptor : characteristic.getDescriptors()) {
                    //find descriptor UUID that matches Client Characteristic Configuration)
                    // and then call setValue on that descriptor

                    descriptor.setValue( Blue-
toothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                    bluetoothGatt.writeDescriptor(descriptor);
                }
```

6.5 Read measurements from the device

Everything is now set to read the measurements from device. Now a list of characteristics from services are available. It is now time to get a descriptor and enable a notification flag. Each service, characteristics, descriptor has its own notification. When the device sends measurements, or notifications then a callback will be received by the application named BluetoothGat-

tCallback.onCharacteristicsChanged. The following code will illustrate how to get notifications.

```
for (BluetoothGattDescriptor descriptor : characteristic.getDescriptors()) {

//find descriptor UUID that matches Client Characteristic Configuration)

// and then call setValue on that descriptor

                descriptor.setValue( Blue-
toothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                bluetoothGatt.writeDescriptor(descriptor);
            }
```

```
@Override
            public void onCharacteristicChanged(BluetoothGatt gatt, final   BluetoothGat-
tCharacteristic characteristic) {
                //read the characteristic data
                byte[] data = characteristic.getValue();
                    }
```

6.6 Test 2nd Iteration processes

In the second iteration it is very important to test whether the device can connect to the application and read measurements correctly. Details of all tests are described in separate chapter. Testing is very important in this phase of the application to make sure that the application can connect properly. Eclipse Log is a very useful tool to monitor whether the application has connected to the device properly. HCI snoop log is also very important in this phase of the application. All Bluetooth data packets can be found in HCI Log which can play a major role to determine whether the Bluetooth is connecting as expected or not.

7 THIRD DEVELOPMENT PHASE

After the second iteration process, the application can now connect to a IoT device and read measurements. In this phase the application will disconnect from the device and will display measurements or a broadcast value to the user interface of the application. So the main task of this phase is to send disconnect request and disconnect from the GATT server, If the application still remains the connection, then forcefully disconnect from the device, show measurements or broadcast value to the user interface.

7.1 Send a disconnect request

To disconnect the application from the device, the same method connectLeDevice was used which was previously used to send the connect request. This time the method was used with different parameters. The parameter is Boolean. If the parameter is true, then the application will send a connect request and if the parameter is false, then the application will send a disconnect request. This time the method will take the parameter false to disconnect the device.

```
onnectLeDevice(mDevice,false, // false means disconnect null);
```

```
connectLeDevice(deviceWithOldestMeasurement,true,mOwningService.getBaseContext());

                private void connectLeDevice(final BluetoothDevice device,
                                             final boolean enable,
                                             Context context ) {

                                if (enable) {
                                             Log.e("BtGw", "connectLeDevice() enable" + de-
vice.getAddress());
                                             mDeviceAddress = device.getAddress();
                                             Intent gattServiceIntent = new       In-
tent(context,CatcherService.class);
                                             isBound=       con-
text.bindService(gattServiceIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
                                             mConnected = System.currentTimeMillis();
                                } else {
                                             if ( mCatcherService != null ) {
                                                          mCatcherService.close();
```

```
                                        }
                                        if ( mOwningService.getBaseContext() != null &&
mServiceConnection != null ) {

                                                if (isBound){
                                                        mOwningS-
ervice.getBaseContext().unbindService(mServiceConnection);
                                                        isBound=false;
                                                }
                                                else{

                                                        //do nothing.
                                                }
                                        }
                                        mDeviceAddress = null;
                                        mDevice = null;
                                        mConnected = -1;
                                        Log.e("BtGw", "connectLeDevice() out after discon-
nect " );
                                }
                }
```

7.2 Disconnect from the GATT server

The disconnect request was sent in the previous state. Now it is time to disconnect from the GATT server. There are two methods available to disconnect from the GATT server. One is a BluetoothGatt.disconnect method and other is a BluetoothGatt.close method. There are many advantages of the BluetoothGatt.close method over the BluetoothGatt.disconnect method. Sometime the BluetoothGatt.disconnect method cannot disconnect from the GATT server properly. Therefore, the BluetoothGatt.close method was used in this application. The following code shows how to use the BluetoothGatt.close method to disconnect from the server.

```
public void close() {
                        if (mBluetoothGatt == null) {
                                return;
                        }

                        mBluetoothGatt.close();
                        mBluetoothGatt = null;
                }
```

7.3 Forcefully disconnect

Sometimes after sending the disconnect command, the application cannot disconnect properly. To make sure that the application disconnects from the GATT server, a method called forcefullyCloseConnection was used. The same method which was used to send theconnect or disconnect request was used here with a different parameter. This method takes three parameters, a device address, a Boolean value to send the connect or disconnect request and the context. The device address and context were set null and the Boolean parameter false to disconnect forcefully. The following code will show how to use the connectLeDevice method to disconnect forcefully.

```
connectLeDevice(null,false, // false means disconnect null);
```

7.4 Display broadcast values in the user interface

In this stage as the application can receive broadcast data, connect and disconnect to the device, data can be saved in the local memory of the phone. Then next task is to make a user interface where the user can see broadcast or measurement data. The AfterLoginScreen class was used to display the broadcast value in the screen. The broadcast value was chosen to be displayed in the user interface because broadcast data is more available than measurement data. It is possible to get a more recent value in the user interface by using broadcast data. An Android list view was used in the application to display the temperature value and RSSI (Received Signal Strength Indicator) in the user interface. Measurements and broadcast values were sent to the server. All values are also available in the web service of *IoLiving*.

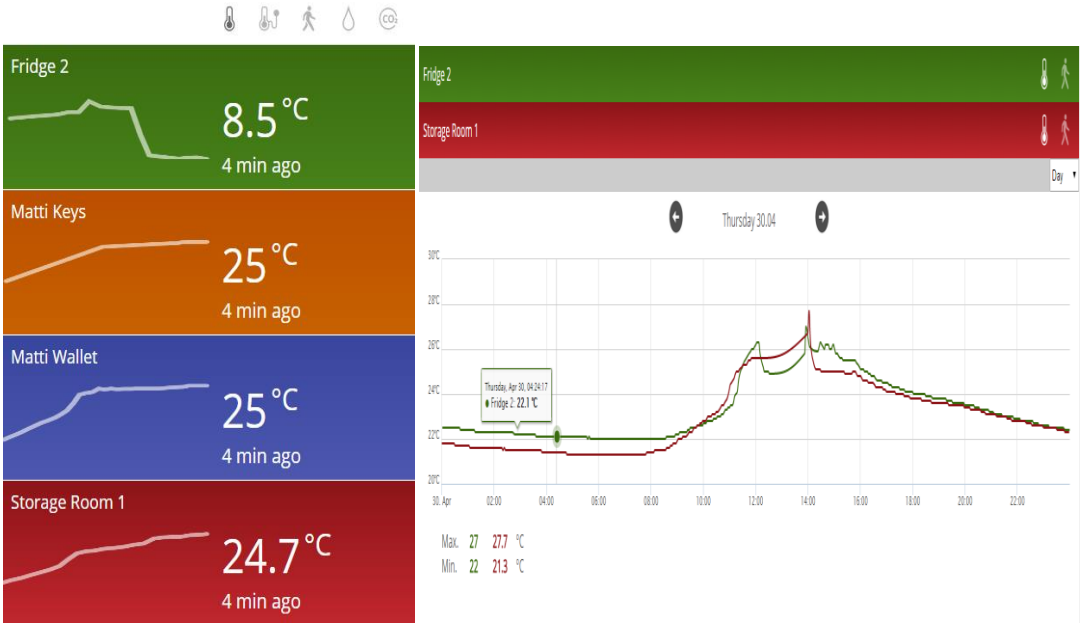

FIGURE 12.IoLiving web shows the temperature and graph

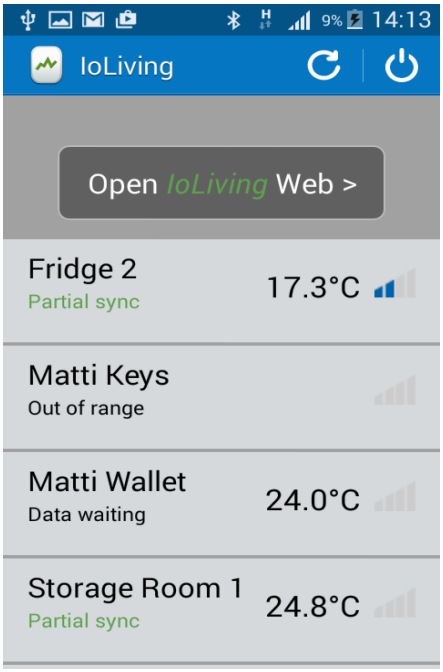

FIGURE 13.IoLiving Android application shows temperature and Radio Signal Strength

7.5 Test 3rd Iteration processes

After the 3rd iteration has been completed, now the application is fully working. Now the application can connect to the IoT devices and read the measurement and save value to the local memory of the phone to use it in the user interface and to disconnect from the device. This phase of the application was tested using the Eclipse log and HCI snoop log. The log shows that the application can now disconnect correctly and it can display values in the user interface. Details of all tests are described in a separate chapter.

8 TEST CASES

Many kinds of tests in detail were run to make sure that the application is working as expected. The Eclipse log and HCI snoop log were used mainly to see the test result. Inside every method there is an Android log to see whether the application is going through those methods to complete the task. The Bluetooth stack is also proving a log of different stages of connection, which is also helpful to figure out whether the application is working properly or not. The most important test cases are described here. The most important test cases are listed below.

Test cases:

1. Does the Bluetooth adapter initialize correctly or not?
2. Does Bluetooth broadcast data or not?
3. Does Bluetooth broadcast correct data or not?
4. Does the application save broadcast data correctly or not?
5. Does Bluetooth make the connection attempt correctly or not?
6. Does Bluetooth close the connection correctly or not?
7. Does the thread grow more than expected or not?
8. Does Watch Dog work correctly or not?
9. Does the application maintain time correctly or not?
10. Record Bluetooth communication to analyze it later.

8.1 Test a Bluetooth initialization

The first thing in this application is to test whether the Bluetooth initialize correctly or not. To make sure that an Android Bluetooth has initialized properly, a log message needs to be put in the method where it initializes Bluetooth. If the Bluetooth does not initialize correctly, then the following log message appears in the Eclipse Log.

```
Log.e(TAG, “Could not open BT adapter!”);
```

From above log it can be determined easily that the Bluetooth did not initialize properly.

8.2 Test a Bluetooth data broadcast

If Bluetooth broadcasts data correctly, then the Eclipse log will show a scan result provided by the Bluetooth stack and also from the Android log. If there is the scan result visible in the Eclipse log, then it is clear that there is scanning going on. If not, then it is necessary to check if there is any device around or not. If there is a device around, then it is clear that an Android code needs to be checked to figure out the bug. The following log is an example of the Eclipse log to determine scanning.

```
05-18 10:58:23.959: D/BluetoothAdapter(2345): onScanResult() - Device=C8:E8:21:D4:7A:E4 RSSI=-65

05-18 10:58:23.959: V/BtGw-cc(2345): onLeScan in

05-18 10:58:23.959: V/BtGw-cc(2345): C8:E8:21:D4:7A:E4 UUID 43520

05-18 10:58:23.959: V/BtGw-cc(2345): C8:E8:21:D4:7A:E4 UUID 6159

05-18 10:58:23.959: V/BtGw-cc(2345): C8:E8:21:D4:7A:E4 UUID 6154
```

8.3 Test broadcast data

This test needs to run in a different environment. Put the device far away from the phone and check if the RSSI value decreases or not. Again put the device close to the phone and check if the RSSI value increases or not. If the RSSI value changes according to the device movement, then it indicates that the device is broadcasting correct data. The device is also broadcasting temperature data. So it also needs to check whether the temperature data is correct or not. Put the device in a cold or hot place in known temperature and check whether the temperature of the place matches with the broadcast temperature. The de-

vice is also broadcasting the MAC address of the device. Check if the MAC address is correct or not.

8.4 Test if broadcast data is saved or not

There is an Android log in the method which is used to save broadcast data. If the application saves broadcast data correctly, then the log message should appear in the Eclipse log. The following log is an example log to determine whether the application saves the data in a database.

```
Log.d(“TAG”,”Data saved in the log correctly”);
```

8.5 Test a Bluetooth connection

The Android Bluetooth stack generates its own log while making a connection with the device. More logs were put inside the method which is handling the connection process. By a analyzing that log method it can be easily determined whether the connection attempt is successful or not. The following log message generates in a successful connection attempt.

```
05-18 10:40:51.139: V/BtGw-cc(2345): Skipping candidate EF:D4:25:32:A2:88

05-18 10:40:51.139: V/BtGw-cc(2345): Skipping candidate FE:5A:7D:37:A7:B4

05-18 10:40:51.139: V/BtGw-cc(2345): Skipping candidate E5:CB:E9:7A:57:70

05-18 10:40:51.139: V/BtGw-cc(2345): Skipping candidate C9:2F:BB:3B:A0:60

05-18 10:40:51.139: D/BtGw-cc(2345): Connect to: FD:81:C0:9D:72:02

05-18 10:40:51.179: D/BtGw-cs(2345): CatcherService onCreate()

05-18 10:40:51.179: D/BtGw-cs(2345): CatcherService onBind()

05-18 10:40:51.179: D/BtGw-cc(2345): CatcherCommsController onServiceConnected()

05-18 10:40:51.179: D/BtGw-cs(2345): connect(...)

05-18 10:40:51.179: D/BluetoothGatt(2345): connect() - device: FD:81:C0:9D:72:02, auto: false
```

```
05-18 10:40:51.179: D/BluetoothGatt(2345): registerApp()

05-18 10:40:51.179: D/BluetoothGatt(2345): registerApp() - UUID=7610e48d-2447-4e61-bd84-
9925592ca97e

05-18 10:40:51.179: D/BluetoothGatt(2345): onClientRegistered() - status=0 clientIf=5

05-18 10:40:51.179: D/BtGw-cs(2345): Trying to create a new connec-
tion.android.bluetooth.BluetoothGatt@22079de0

05-18 10:40:59.999: D/BtGw-s(2345): inside no task.

05-18 10:41:00.009: D/BtGw-s(2345): WatchDog asleep for 30s

05-18 10:41:00.009: D/BtGw-s(2345): WatchDog Sleeping.

05-18 10:41:00.009: D/BtGw-s(2345): Service.OnStartCommand in with action_sync(timer-
handler and on Boot)

05-18 10:41:05.679: D/BluetoothGatt(2345): onClientConnectionState() - status=0 clientIf=5
device=FD:81:C0:9D:72:02

05-18 10:41:05.679: V/BtGw-cc(2345): BTDeviceStateChangedCallBack
com.ceruus.catcher.ACTION_GATT_CONNECTED -1 null

05-18 10:41:05.679: D/BtGw-cs(2345): Connected to GATT server.

05-18 10:41:05.679: D/BluetoothGatt(2345): discoverServices() - device: FD:81:C0:9D:72:02

05-18 10:41:05.679: D/BtGw-cs(2345): mBluetoothGatt.discoverServices() result = true

05-18 10:41:05.689: D/BluetoothGatt(2345): onGetService() - Device=FD:81:C0:9D:72:02
UUID=00001800-0000-1000-8000-00805f9b34fb
```

8.6 Test a Bluetooth disconnect process

The Eclipse log needs to monitor to determine whether the application disconnects correctly. If the application closes correctly then the Bluetooth stack will generate some message saying Bluetooth GATT close. This message confirms that the disconnection process was successful. The following log message generates after a successful disconnection.

```
05-18 11:41:34.229: D/BtGw-cs(2345): measurementStreamRawData.size: 5

05-18 11:41:34.229: D/BtGw-cs(2345): START measurement stream parsing and saving.

05-18 11:41:34.229: V/BtGw-cc(2345): BTDeviceStateChangedCallBack
com.ceruus.catcher.ACTION_MEASUREMENT_STREAM_READ_COUNT 5 null

05-18 11:41:34.259: V/BtGw-d(2345): Saved (true) measurement for device F1:73:C4:2F:82:34
len = 100 using ts 1431938494188

05-18 11:41:34.259: E/BtGw-d(2345): Number of measurements in db for that catcher = 1 ts = 1

05-18 11:41:34.259: D/BtGw-cs(2345): ENDED measurement stream parsing and saving.

05-18 11:41:34.259: D/BtGw-cs(2345): measurementStreamActive: false

05-18 11:41:34.259: V/BtGw-cc(2345): BTDeviceStateChangedCallBack
com.ceruus.catcher.ACTION_MEASUREMENT_STREAM_READ_ENDED -1 null

05-18 11:41:34.259: D/BtGw-cs(2345): gatt server closed

05-18 11:41:34.259: D/BluetoothGatt(2345): close()

05-18 11:41:34.259: D/BluetoothGatt(2345): unregisterApp() - mClientIf=5

05-18 11:41:34.259: D/BluetoothGatt(2345): cancelOpen() - device: F1:73:C4:2F:82:34

05-18 11:41:34.269: D/BtGw-cc(2345): connectLeDevice() out after disconnect

05-18 11:41:34.269: D/BtGw-cs(2345): CatcherService onUnbind()

05-18 11:41:34.269: D/BtGw-cs(2345): CatcherService onDestroy()
```

8.7 Test if an unnecessary thread is growing or not

This application creates many threads. The test needs to determine whether the application is producing more thread than expected. There are exit points to go to the next task when one task is complete. If the exit point puts in a wrong place, then the application creates more and more thread. From the Eclipse log it can be easily determined whether the application creates more thread than expected. For example, there will be two scan or two connection attempt, If

there is any unusual behaviour seen in the Eclipse log, then it needs to check whether all exit points are correct or not.

8.8 Test the Watch Dog Functionality

If watch Dog works correctly, it should show a log message in the Eclipse log in every 10 seconds. Watch Dog is one kind of timer so the value of second should increase every time it appears in the log. Below the log message helps to determine whether Watch Dog is working as expected or not.

```
05-18 11:58:00.019: D/BtGw-s(2345): WatchDog asleep for 30s

05-18 11:58:00.019: D/BtGw-s(2345): WatchDog Sleeping.
```

8.9 Test application time

A test was run using five devices to see whether the application is actually maintaining the expected time or not. The following table shows time taken by five devices in different phases of the application. This test takes around 30 minutes time to complete. Times shown in the table are only minutes and seconds.

For one particular device:

- One cycle (Scan – Connection - Sync): It will typically take around 1 – 2 minutes. This time will alternate.
- So if it takes around 1 minute to complete the cycle once, then it will take around 2 minutes next time (If there is more than one device).
- Connection time will be around 6 minutes always.
- Sync time will typically be around 6 minutes (If there is only one device).

So if it takes around 4 minutes (If there is more than one device) to complete the sync once, then it will take around 7 minutes (if there is more than one device) next tim

Cy-cle No.	IoLiving 343E			IoLiving 72CE			IoLiving 8DA6			IoLiving D2DB			IoLiving EF36		
	Scan	Con-nect	Sync	Scan	Con-nect	Sync	Scan	Con-nect	Sync	Scan	Con-nect	Sync	Scan	Con-nect	Sync
1			11:48			11:48			11:48			11:48			11:48
2	12:05	12:29													
3							13:30	13:55	14:10						
4										14:30	14:50				
5				15:31	15:55	15:56						15:56			
6													16:31	16:55	
7	17:30	17:55	18:30												
8							18:48	19:15							
9									20:30	19:48	20:15				
10				20:48	21:15										
11						23:04							22:00	22:27	23:04
12	23:20	23:45													
13			24:57				24:20	24:48	24:57						
14										25:21	25:48				

15				26:16	26:41	26:57						26:57			
16													27:16	27:45	
17	28:16	28:41	29:23												29:23
18							29:30	30:01							
19									32:02	30:55	31:22	32:02			
20				32:20	32:47										
21						34:03							33:20	33:47	34:03
22	34:20	34:48													
23			36:36				35:34	36:01	36:36						
24										36:54	37:21				
25				38:16	38:41	39:00						39:00			
26													39:16	39:41	
27	40:16	40:41	40:57												40:57
28							41:16	41:42							
29									44:00	43:05	43:30	44:00			

FIGURE 14.Table shows the time consumption of different process of the application

8.10 Record a Bluetooth communication to analyze it later

There is an option in an Android phone to record all Bluetooth communication as a log. This log is very helpful later on to analyze all Bluetooth communication done by the application. WireShark was used to analyze Bluetooth communication. To record all Bluetooth communication, first it needs to enable the Bluetooth HCI snoop log. To Enable the Bluetooth HCI snoop log, the following step needs to be done.

Steps:

1. Go to Setting.
2. Click About Phone.
3. Click Software information.
4. Click Build number 7 times, it will enable a Developer option.
5. Go back to settings and click a developer option.
6. Click Enable a Bluetooth HCI snoop log before starting the application.
7. Target a folder where the HCI log will be downloaded.
8. Select the folder, then press Shift and right click the mouse button.
9. From the menu select “Open Command Window Here”.
10. When a window will open “adb pull/sdcard/btsnoop_hci.log” needs to be written in the command window.
11. Press Enter.
12. This file can be opened using Wireshark.

The HCI snoop log contains each and every Bluetooth communication. To detect a Bluetooth related problem, this log is very helpful. The picture below shows an example of the Bluetooth HCI snoop log.

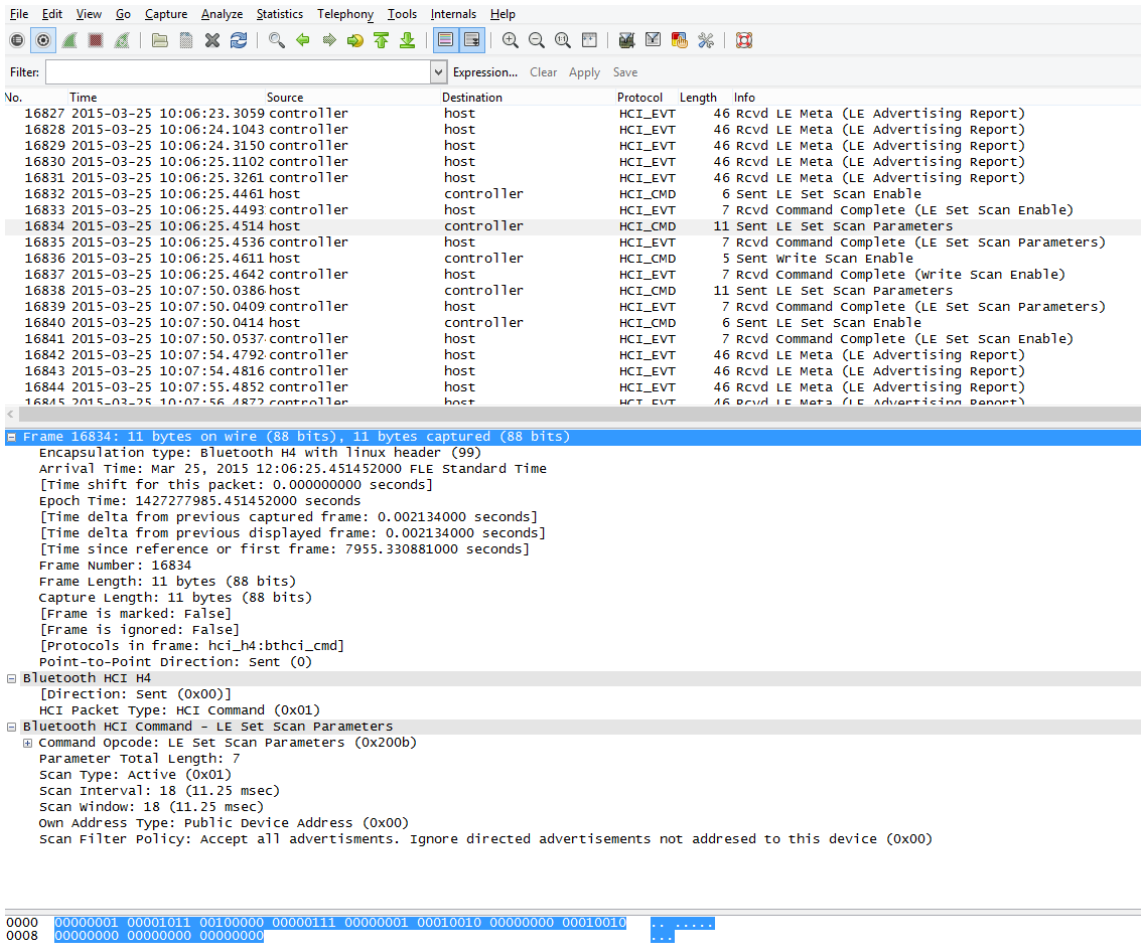

FIGURE 15.HCI snoop log data analysis by using Wireshark

9 CHALLENGES TO ESTABLISH A CONNECTION

The Current Bluetooth stack available has a limitation. Even in some cases the any other issue is interrupting the connection process. It can be seen that the application is facing few problems to make a successful connection. I have listed a few common problems facing by this application.

Problems face by the application currently:

1. Failed to discover the service.
2. Gatt Error.
3. Failed to discover a proprietary service or the service is null.
4. Android Bluetooth stack is choosing a wrong connection method.
5. Bluetooth does not list any BLE device.

9.1 Failed to discover services

When the application tries to discover services, there is a status to determine whether the service discovery was successful or not. If the method onServiceDiscoverd returns a status 0, then it indicates that the service was discovered successfully. But in some cases it is returning 129 which is a GATT_INTERNAL_ERROR 0*0081.

```
onServicesDiscovered() status = 129 or (0×0081)
```

According to Google specification, it is GATT_INTERNAL_ERROR:

```
1. #define GATT_INTERNAL_ERROR 0x0081
```

9.2 Gatt Error

While the application is in the connection process, the application faces another error called Gatt Error. This is the most common error faced by the application while trying to establish a connection.If the method onClientConnectionState

returns a status 0, then it indicates that the connection was made successfully. But in some cases it is returning 133 which is a GATT_INTERNAL_ERROR 0*0085.

```
onServicesDiscovered() status = 133 or (0×0085)
```

According to Google specification, it is GATT_INTERNAL_ERROR:

```
1.  #define GATT_INTERNAL_ERROR 0x0085
```

9.3 Failed to discover a Proprietary service or the service is null

Sometimes there is no service available or Gateway failed to discover a proprietary service. In that case Gateway receives a service status null.

```
ACTION_GATT_DEVICE_REJECTED - 1 NULL
```

This problem does not occur very often. Gateway faces this problem rarely.

9.4 Android Bluetooth Stack is choosing a wrong connection method

After the Bluetooth HCI Log study, it can be noticed that the Android Bluetooth Stack is choosing the wrong Bluetooth connection method for the first few hours when the application starts for the first time. It is choosing the Classic Bluetooth connection for the first few hours and then it is deciding itself to try the Bluetooth LE connection method. When the connection is successful with the Bluetooth LE connection method, the Bluetooth Stack starts using continuously the Bluetooth LE connection method. The picture below shows the HCI log for the Bluetooth LE connection.

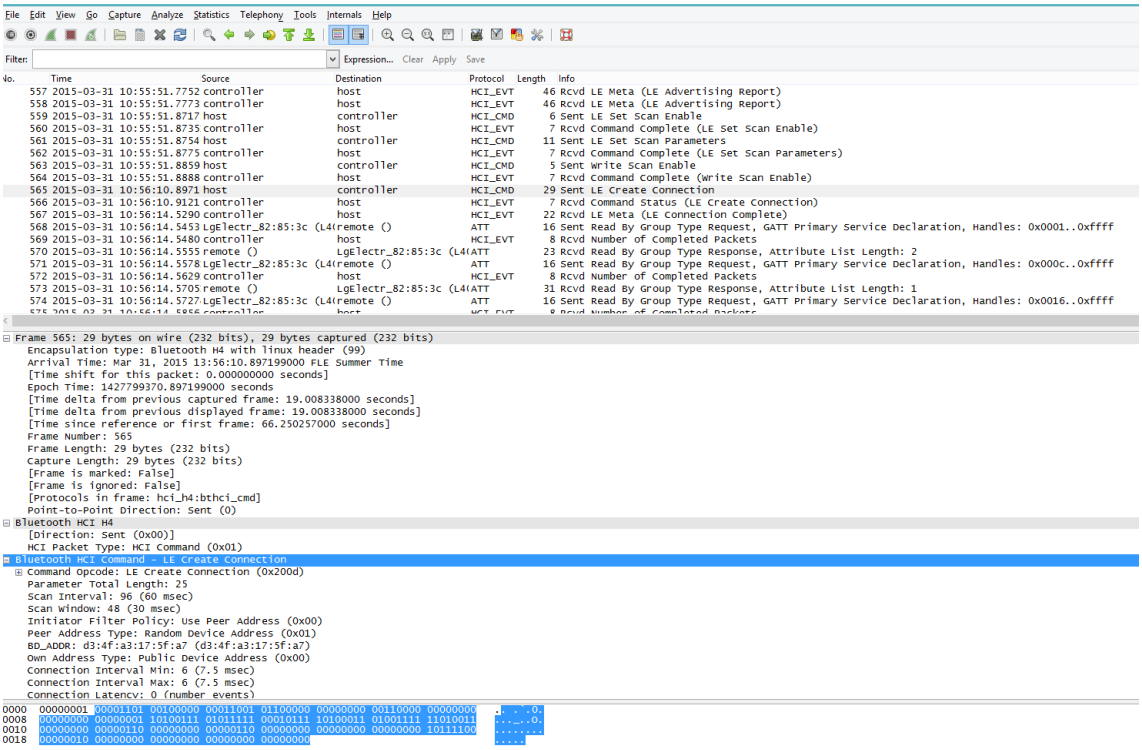

FIGURE 16.HCI snoop log shows Bluetooth LE connection command

9.5 Application does not list any BLE device

In some special case it can be seen that the application does not list any device while scanning, but there are devices around. This problem does not occur very often. After restarting, the Bluetooth stack application can list the BLE device again.

10 CONCLUSION

The Bluetooth Low Energy connection is needed to communicate with Bluetooth Low Energy devices. This thesis described different phases of Bluetooth Low Energy Communication. Testing the application played a very important role to figure out problems in the Bluetooth Low Energy connection process. Few problems have been noticed in the connection process but overall the performance of Bluetooth communication is very satisfactory. One of those major problems was the Bluetooth could not list any Bluetooth Low Energy devices after scanning. Restarting the Bluetooth stack can handle the problem efficiently. Another problem faced by the application was getting an error in the connection process. In those cases extra connection attempts were helpful to establish a successful connection.

The current application is scanning for all Bluetooth Low Energy devices around but a future version of this application will scan for only the selected device. As a result the scan process will be faster and the application will take less time to complete the Bluetooth connection process with the Bluetooth Low Energy devices.

Though there are few challenges to establish a Bluetooth connection with IoT devices, the result shows that this application can now successfully establish a connection to devices. By all counts, and with proven results, it is no wonder that the thesis can be considered as a successful one.

REFERENCES

Ali M Aljuaied, S. 2001.Bluetooth Technology and Its Implementation in Sensing Devices. Monterey, California.

Barrett, J., Internet of Things, Date of Retrieval 21.05.2015
https://www.youtube.com/results?search_query=internet+of+things

Bluetooth, Bluetooth GATT Profile, Date of Retrieval 21.05.2015
https://developer.bluetooth.org/TechnologyOverview/Pages/GATT.aspx
https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

Bluetooth, Bluetooth Profile, Date of Retrieval 21.05.2015
https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx

Bluetooth, Bluetooth SIG, Date of Retrieval 21.05.2015
https://developer.bluetooth.org/AboutUs/Pages/SIG-Membership.aspx

Bluetooth, 2015, Bluetooth Smart, Date of Retrieval 26.04.2015
http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx

Huang, F. 2013.Web Technologies for the Internet of Things. Espoo.

Iconfinder, Android Smart phone icon, Date of Retrieval 26.04.2015
https://www.iconfinder.com/icons/211118/android_smart_phone_icon

*Io**Living***, Internal Source, *Io**Living*** documents about Bluetooth Low Energy, Date of Retrieval 21.05.2015

IoLiving, IoLiving Device, Date of Retrieval 26.04.2015
https://www.ioliving.com/wp-content/themes/ioliving/img/ioliving_m_catcher.png

Pixshark, cloud-server, Date of Retrieval 26.04.2015
http://pixshark.com/cloud-server-png.htm

Townsend,K., Devidson,R. & Akiba,Getting Started with Bluetooth Low Energy, Date of Retrieval 21.05.2015
https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/cover.html