University of Thessaly

Μεταπτυχιακή εργασία

Θέμα:

"Control Smart-home equipment through Android Application"

Βαραλής Αργύριος

UNIVERSITY OF THESSALY

Επιβλέπων καθηγητής:

Αθανάσιος Κοράκης

Συνεπιβλέπων καθηγητές:

Τασιούλας Λέανδρος

Αργυρίου Αντώνιος

Βόλος, Σεπτέμβριος 2014

University of Thessaly

Acknowledgments

I offer my sincerest gratitude to my supervisor, Dr Korakis Athanasios. Also, i express my thanks to my co-supervisors Leandros Tassiulas, and Argyriou Antonios. Moreover I am grateful to Ioannis Kazdaridis for the guidance across the whole period of this work. Most special thanks go to my parents and my friends who supported me with patience and love through this long process.

University of Thessaly

Abstract

Nowadays a large percentage of the utilization of pc's has been substituted by mobile devices, such as phones and tablets. More and more our homes have become technologically advanced. Many home appliances can be controlled wirelessly or connect to the Internet. The ease of use and portability are the main reasons that made mobile devices so widespread, so why not use them to control a smart-home. Thousands of applications are constructed, considering the needs of every user, covering different platforms such as android, iOS, Windows.

The purpose of this report is the design and construction of an Android application, which controls and monitors energy consumption of electrical appliances. NITLab [1] developed a Power Meter framework consisted of Power Meter Devices [2] and respective User Interface. This framework is capable of sampling the Power Consumption of a connected electric device. Moreover, it can sense environmental conditions using a temperature and humidity sensor and a light intensity photo-resistor sensor as well. Until now users could only control and view data measures through a web browser via http requests, thus an Android application has been developed as an alternative and more practical way. The entire hardware infrastructure is analyzed in detail and different use cases of the application are demonstrated. Measurements of various electrical appliances are depicted as an example of use.

University of Thessaly

Table of Contents

University of Thessaly

Table of Figures

1. Introduction

NITLab has designed and constructed the hardware of the power meter devices which are constituted by various boards, wireless modules and sensors. Until now users could control, monitor and configure the devices by sending http requests through a web browser where the data is displayed. As an alternative and handier way to utilize these actions an application has been developed using Eclipse ADT bundle [3] which provides the android SDK tools and a version of Eclipse IDE with built-in Android Development tools. In the next chapters this work is presented in more detail. In the second section a detailed description of the power meter hardware devices is presented, following the topology of the use cases and the possible data exchanged of the devices is shown in section 3. Android components which were used are outlined in section 4 and finally in section 5 use cases of the application are shown while measurements of various electrical devices are presented. An alternative way of communicating with the devices utilizing Bluetooth connectivity is presented in section 6 as future work.

2. Hardware

In this section the hardware is anatomized; the power meter device and the Gateway.

2.1 Power meter device

NITlab developed a Power Meter framework consisted of Power Meter Device. The Power Meter Device features:

- **Arduino Pro Micro microcontroller (Figure 1) board featuring the ATMega 32U4 running at 3.3V/8MHz microcontroller [4].**
  The Pro Micro has an ATmega32U4 on board. The USB transceiver inside the 32U4 allows users to add USB connectivity on-board and do away with bulky external USB interface. The boards features are; 4 channels of 10-bit ADC, 5 PWM pins, 12 DIOs as well as hardware serial connections Rx and Tx. Running at 8MHz and 3.3V.
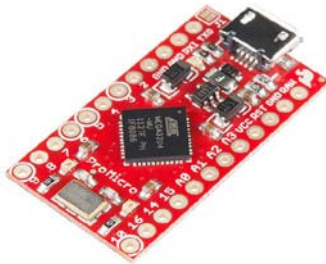
Figure 1 : Pro Micro controller

- **ACS712 current sensor [5] that provides precise current measurements**
  The Allegro ACS712 provides economical and precise solutions for AC or DC current sensing in industrial, commercial, and communications systems. The device package allows for easy implementation by the customer. Typical applications include motor control, load detection and management, switched-mode power supplies, and over current fault protection. The device is not intended for automotive applications.

**Figure 2 : ACS712 current
sensor**

- **Featuring Electrical Power Relay**

  A power relay is a switch which uses an electromagnetic coil in order to close
  or open a circuit. Power relays also contain an armature, a spring and one or
  several contacts. If the power relay is designed to normally be open, when
  power is applied, the electromagnet attracts the armature, which is then pulled
  in the coil's direction until it reaches a contact, therefore closing the circuit. If
  the relay is designed to be normally closed, the electromagnetic coil pulls the
  armature away from the contact, therefore opening the circuit.

- **Xbee S2 [6] module for wireless communication**

  XBee Series 2 improves on the power output and data protocol. Series 2
  modules allow users to create complex mesh networks based on the XBee ZB
  ZigBee mesh firmware. These modules allow a very reliable and simple
  communication between microcontrollers, computers, systems and anything
  with a serial port. Point to point and multi-point networks are supported.

**Figure 3 : Xbee S2 module for
wireless communication**

8

- **Sht11 temperature & humidity sensor**

  The digital humidity and temperature sensor SHTx is a reflow solderable sensor. The SHT1x-series contains a low cost version with the SHT10, a standard version with the SHT11 and a high end version with the SHT15. As every other Sensirion sensor type of the SHTxx family, they are fully calibrated and provide a digital output. SHT11 is the standard version, which offers +/-3% RH accuracy.

- **Light intensity photo-resistor sensor**

  A photo-resistor or light-dependent resistor (LDR) or photocell is a light-controlled variable resistor. The resistance of a photo-resistor decreases with increasing incident light intensity; in other words, it exhibits photoconductivity. A photo-resistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits.

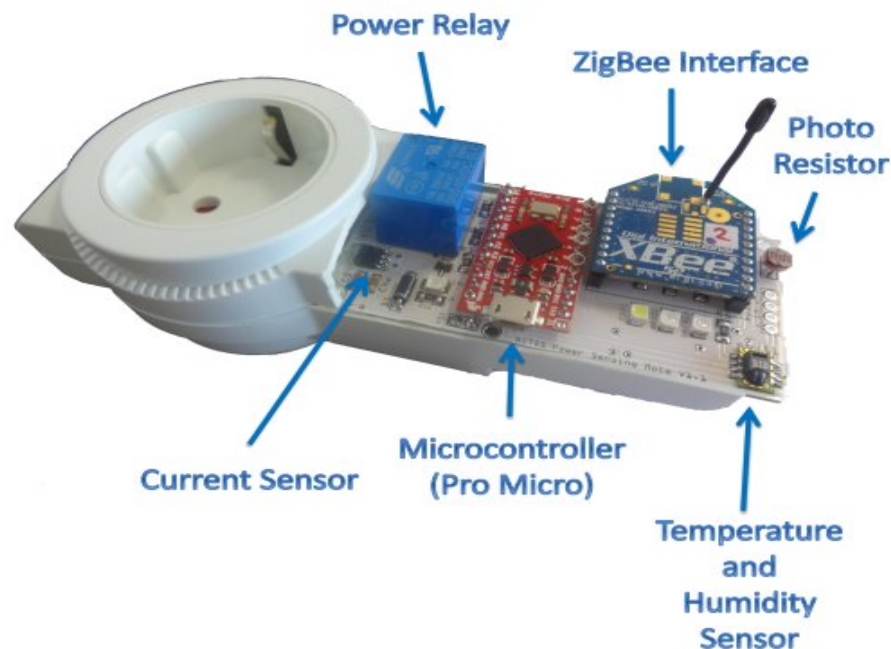The complete power meter device is depicted in Figure 4.

Figure 4 : Power meter device

2.2 Gateway

Gateway 's main features:

- **Arduino Ethernet Board [7]**

  The Arduino Ethernet is a microcontroller board based on the ATmega328. It has 14 digital input/output pins, 6 analog inputs, a 16 MHz crystal oscillator, a RJ45 connection, a power jack, an ICSP header, and a reset button.

  Pins 10, 11, 12 and 13 are reserved for interfacing with the Ethernet module and should not be used otherwise. This reduces the number of available pins to 9, with 4 available as PWM outputs. An onboard microSD card reader, which can be used to store files for serving over the network, is accessible through the SD Library. Pin 10 is reserved for the Wiznet interface; SS for the SD card is on Pin 4.
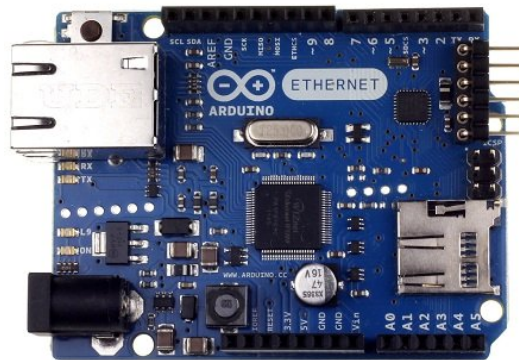
Figure 5 : Arduino Ethernet board

- **Custom-made shield**
- **Xbee S2**

  The same module introduced in 2.1

Gateway receives mesurments from the power meter device and displays data on a web interface. The collected mesasurments are refreshed every 1 second and is

10

connected to the Internet through an Ethernet cable, where an IP is set manually to facilititate application communicate with it. The Xbee located on the Gateway associates with the Xbee's located on the power meter devices, receives results and sends configurations. The Arduino Ethernet board runs a WebServer thtat allows users to communicate with power meter devices through HTTP protocol.

The complete Gateway device is depicted in Figure 6.

Figure 6 : Gateway device

3. Topology and data returned

3.1 Topology

To map the topology of a possible case scenario, each plug has attached an electrical appliance which the user wants to control or measure its power consumption. The microcontroller, located on the power meter device, collects the all the measurements; power consumption, values of photocell sensor, value of the temperature sensor and value of humidity sensor. Through a wireless interface each plug sends the measurements to the Gateway. User utilizing the android application can control the plugs (on/off), monitor the current and the cumulative power consumption, also a live graph depicting the current consumption is incorporated and finally the user can configure the Gateway.
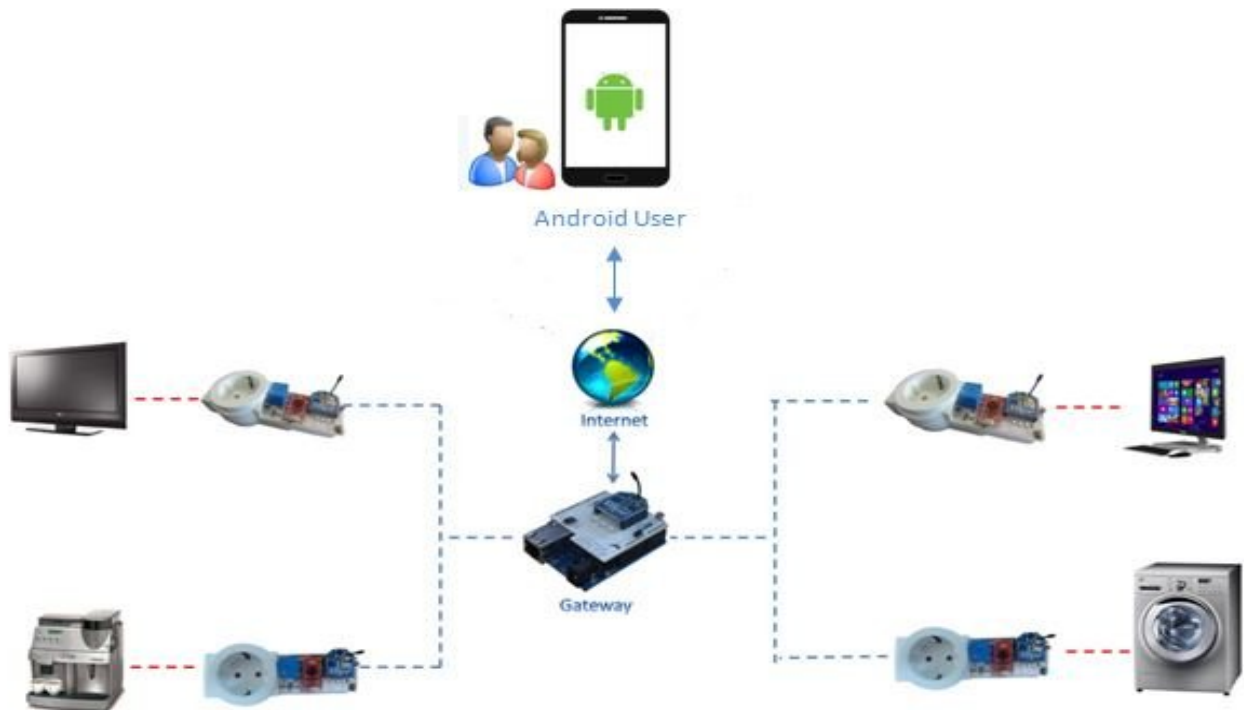

Figure 7 : Topology

3.2 Data transferred

Gateway needs to be triggered from the Android User (AU) to send back the measurements. The trigger is an http request with the necessary format. All the possible triggers are presented and explained. Commands towards the Gateway (IP : 192.168.1.20) :

- 192.168.1.20/
  - Returns the measurements of all plugs.
- 192.168.1.20/1-on
  - Turns on plug 1.
- 192.168.1.20/1-off
  - Turns off the plug 1.
- 192.168.1.20/1-switch
  - Turns on plug 1 if it is off, or turns it off if it is on.
- 192.168.1.20/1-calibrate
  - It is called without needing any appliance to be connected to the plug and tunes all the sensors.
- 192.168.1.20/1-reset
  - Resets the counters; Amperes per hour, Watts per hour and Time since last reset (in seconds), for plug 1.
- 192.168.1.20/1-sampling-20-40-0
  - Sets new parameters for sampling. The values are samples per period, periods per measurement and measurements interval respectively.

To demonstrate the commands Curl [8] was used on Linux OS. The power meter used for the experiments is the one with node id 1. All the commands return "ok" except the one that returns the measurements of the devices. The ip of the Gateway is "192.168.1.20". Some examples are presented below:

So with a reset request (Figure 8) we have:

Figure 8 : Http request and answer from the Gateway

13

University of Thessaly

The data structure that the Gateway returns to the AU is illustrated below:

<node_id> <instantaneous consumption in (Amperes)> <total consumption (Amperes per hour)> <total consumption (Watts per hour)> <time since last reset of the total consumption counters (seconds)> <value of the photocell sensor (percentage)> <value of the temperature sensor (Celsius)> <value of the humidity sensor (percentage)> <input voltage of the Arduino (volts)> <device status off or on (integer 0 or 1)> <samples per period (integer 10-80)> <periods per measurement (integer 1-50)> <measurements interval milliseconds (integer 0-600000)>

When the Gateway receives the command "192.168.1.20/" it returns the available measurements for all the devices connected. Each value is separated from the other with space and every device is separated from the other with '\n'. An example of this command is we assume that we have three plugs is shown below:
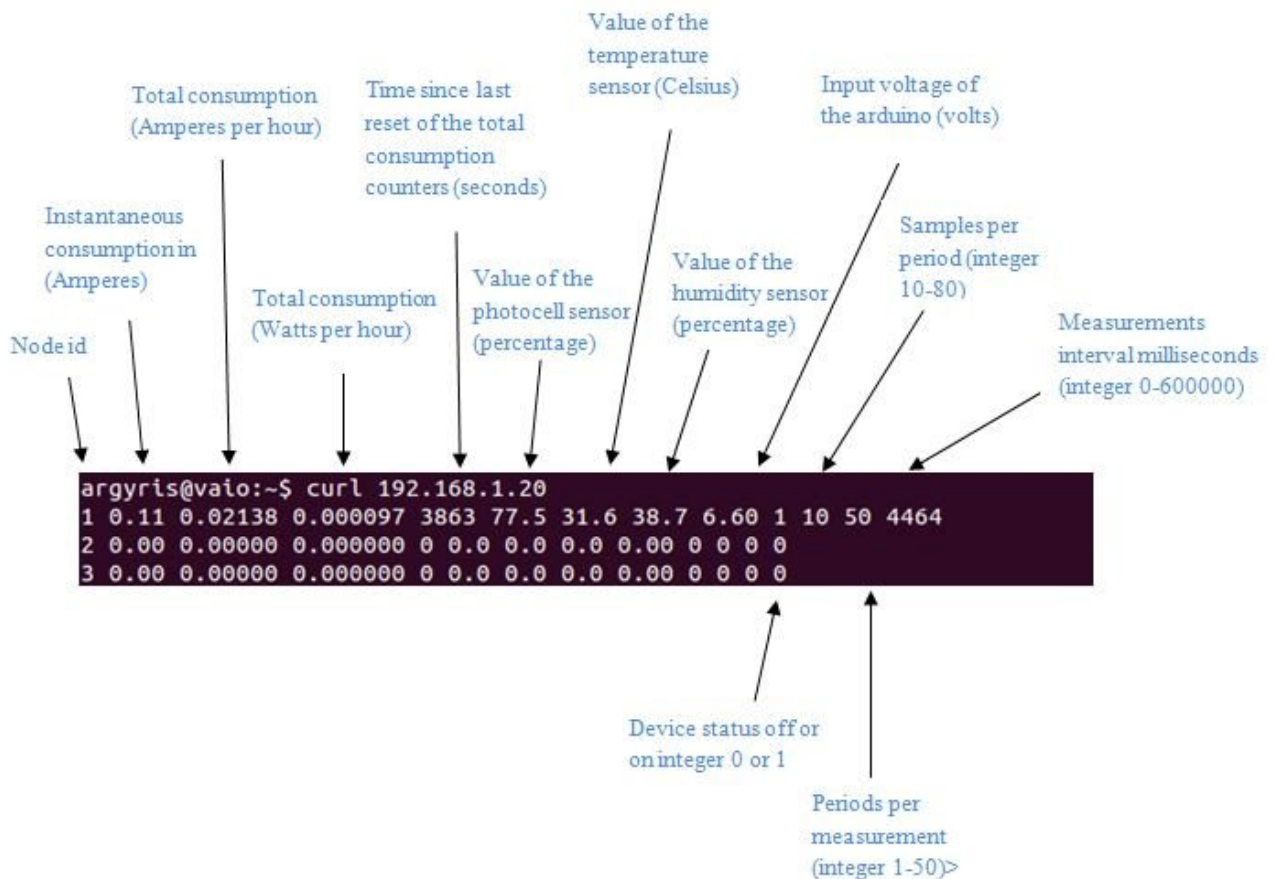

Figure 9 : Explanation of fields of the response

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 10:11:27 EET - 137.108.70.7

University of Thessaly

An example of sampling in Figure 10 shows the change of values in last three values; samples per period, periods per measurement, measurements interval milliseconds.

.

```
argyris@vaio:~$ curl 192.168.1.20/
1 0.00 0.00009 0.000000 77 77.5 30.6 39.9 6.60 1 10 50 4464
2 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
3 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
argyris@vaio:~$ curl 192.168.1.20/1-sampling-20-40-1000
ok
argyris@vaio:~$ curl 192.168.1.20/
1 0.08 0.00057 0.000003 168 77.5 31.2 39.1 6.60 1 20 40 1000
2 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
3 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
```

Figure 10 : Demonstration of changing the three parameters

And the effect of reset depicted in Figure 11 as all the counters of the device are set to zero.

```
argyris@vaio:~$ curl 192.168.1.20/
1 0.00 0.00000 0.000000 0 77.6 31.3 39.0 6.60 1 10 50 4464
2 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
3 0.00 0.00000 0.000000 0 0.0 0.0 0.0 0.00 0 0 0 0
```

Figure 11 : Reset effect on all counters.

In case of a disconnected plug the Gateway returns DISCONNECTED for respective plug. Moreover in the event of a wrong "node id", it returns the message "invalid node id" and in the case of an invalid command, it returns "invalid command". In all the other cases the Gateway returns the message "ok", without meaning that the command has delivered to the respective device.

University of Thessaly

4. Android Application

For the construction of the application Eclipse Android Developer Tools (ADT) Bundle was used, which provides all the necessary components, including the Android SDK tools and a version of the Eclipse IDE with built-in Android Developer tools. The programming language is **Java**.

4.1 Android components of application

Some important Android components will be presented so as to be able to proceed to the explanation of the application:

- Activities
- Fragments
- User Interface
- Swipe view with tabs
- Action Bar
- Async task
- GraphView
- Shared Preferences

4.1.1 Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. An application could have more than one Activity. All Activities are declared in the Manifest.xml file, but one of them must be set to be presented first. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses

the *Back* button, it is popped from the stack (and destroyed) and the previous activity resumes.

Lifecycle of an Activity is presented in Figure. To create an activity, you must create a subclass of Activity (or an existing subclass of it). In the subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle, such as when the activity is being created, stopped, resumed, or destroyed. The two most important callback methods are:

- **onCreate()**

  You must implement this method. The system calls this when creating your activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call setContentView() to define the layout for the activity's user interface.

- **onPause()**

  The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

There are several other lifecycle callback methods that user should use in order to provide a fluid user experience between activities and handle unexpected interruptions that cause your activity to be stopped and even destroyed.
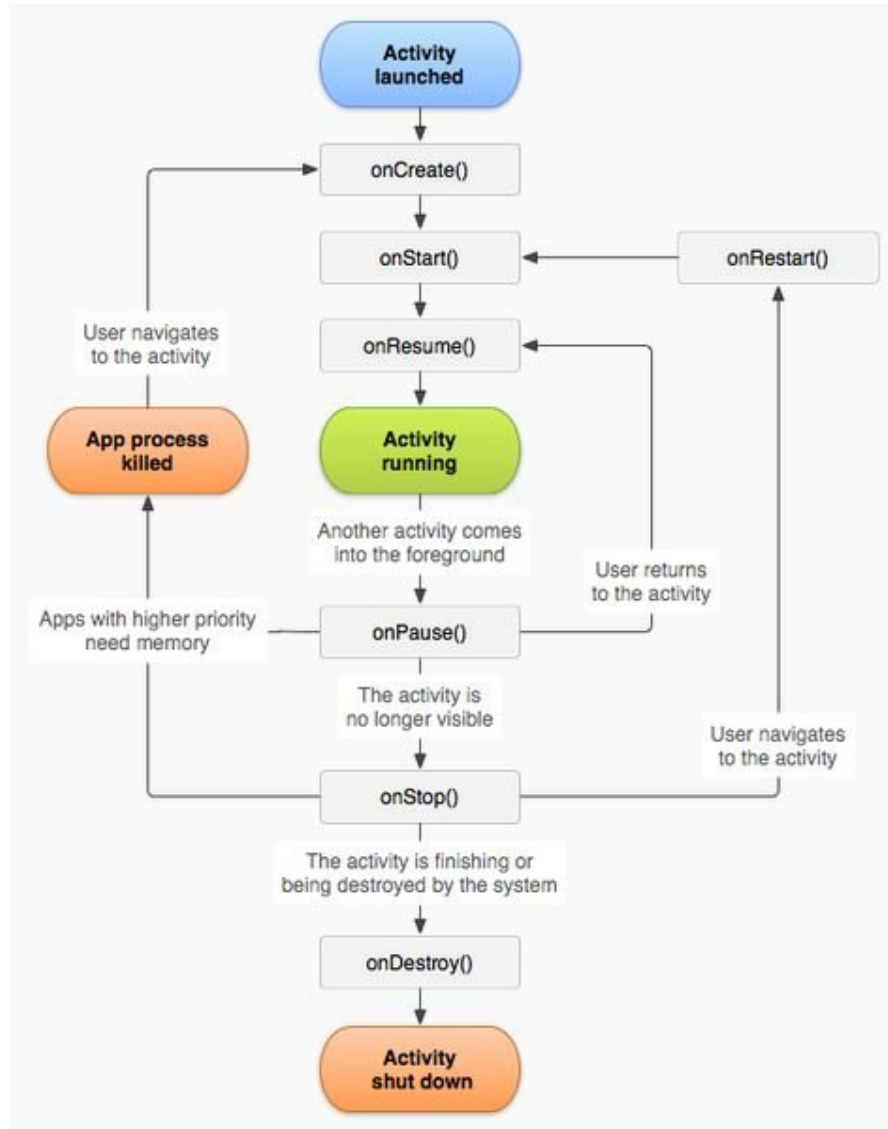
Figure 12 : Lifecycle of an Activity.

4.1.2 Fragments

A Fragment represents a behavior or a portion of user interface in an Activity. Multiple fragments can be combined in a single activity to build a multi-pane User Interface (UI) and reuse a fragment in multiple activities. Fragment can be regarded as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running.

A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the resumed lifecycle state), user can manipulate each fragment independently, such as add or remove them. When user performs such a fragment transaction, also can add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.

To create a fragment, a subclass of Fragment (or an existing subclass of it) must be created. The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as onCreate() , onStart(), onPause(), and onStop().

Usually, at least the following lifecycle methods should be implemented:

- **onCreate()**

    The system calls this when creating the fragment. Within the implementation, user should initialize essential components of the fragment that are wanted to retain when the fragment is paused or stopped, then resumed.

- **onCreateView()**

    The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for a fragment, user must return

a View from this method that is the root of the fragment's layout. *Null* can be returned if the fragment does not provide a UI.

- **onPause()**

  The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where any changes should be commited, which should be persisted beyond the current user session (because the user might not come back).
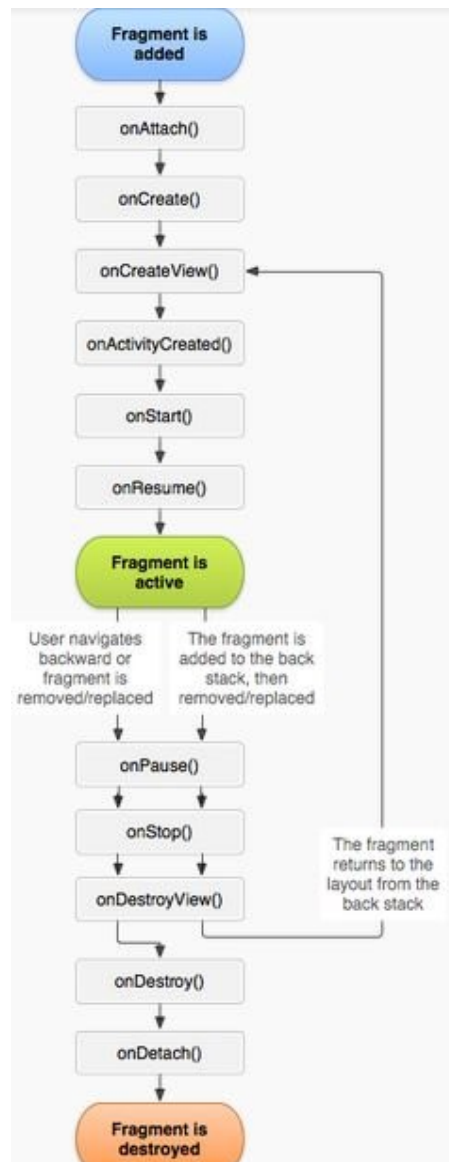
Figure 13 : Lifecycle of a Fragment.

4.1.3   User Interface

User interface is everything that the user can see and interact with. Android provides a variety of pre-build UI components such as structured layout objects and UI controls that allow user to build the graphical user interface for an application. All user interface elements in an Android application are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface. Android provides a collection of both View and ViewGroup subclasses that offer common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

A layout defines the visual structure for a user interface, such as the UI for an activity. It can be declared a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**. Application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Most common Layouts are:

- **LinearLayout** is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute.
- **RelativeLayout** is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements or in positions relative to the parent RealtiveLayout area.
- **ListView** is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a

21

source such as an array or database query and converts each item result into a view that's placed into the list.

- **GridView** is a ViewGroup that displays items in a two-dimensional, scrollable grid.

4.1.4   Swipe Views with tabs

Swipe views provide lateral navigation between sibling screens such as tabs with a horizontal finger gesture. User can create swipe views in an application using the ViewPager widget, available in the Support Library. The ViewPager is a layout widget in which each child view is a separate page (a separate tab) in the layout.

To insert child views that represent each page, you need to hook this layout to a pagerAdapter . There are two kinds of adapter you can use:

- **FragmentPagerAdapter**

  This is best when navigating between sibling screens representing a fixed, small number of pages.

- **FragmentPagerAdapter**

  This is best for paging across a collection of objects for which the number of pages is undetermined. It destroys fragments as the user navigates to other pages, minimizing memory usage.

4.1.5   Action Bar

The action bar is a window feature that identifies the user location, and provides user actions and navigation modes. Using the action bar (Figure 14) offers users a familiar interface across applications that the system gracefully adapts for different screen configurations.

Action Bar provides several key functions:

- Makes important actions prominent and accessible in a predictable way.
- Supports consistent navigation and view switching within apps.
- Reduces clutter by providing an action overflow for rarely used actions.

- Provides a dedicated space for giving your app an identity.

Figure 14 : Action Bar.

4.1.6   Async Task

AsyncTask is a class that enables proper and easy use of the UI thread. This class allows performing background operations and publishing results on the UI thread without having to manipulate threads and/or handlers. AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks are ideally used for short operations (a few seconds at the most.). An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

- **onPreExecute**

  Invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.

- **doInBackground**

  Invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to

23

publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.

- **onProgressUpdate**

  Invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing.

- **onPostExecute**

  Invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

4.1.7   GraphView

GraphView is a library for Android to programmatically create flexible and nice-looking diagrams.

Some features of GraphView library are listed below:

- **Two chart types**

  Line Chart and Bar Chart.

- **Draw multiple series of data**

  Let the diagram show more than one series in a graph. You can set a color and a description for every series.

- **Show legend**

  A legend can be displayed inline the chart. The width and the vertical align (top, middle, bottom) can be set.

- **Custom labels**

  The labels for the x- and y-axis are generated automatically. But user can set his own labels, Strings are possible.

- **Handle incomplete data**

  It's possible to give the data in different frequency.

- **Viewport**

  User can limit the viewport so that only a part of the data will be displayed.

24

- **Scrolling**

  User can scroll with a finger touch move gesture.

- **Scaling / Zooming**

  Since Android 2.3! With two-finger touch scale gesture (Multi-touch), the viewport can be changed.

- **Background (line graph)**

  Optionally draws a light background under the diagram stroke.

- **Custom Style**

  Change the color and thickness, label font size/color and more

- **Realtime / Live**

  Append new data live or reset the whole data

- **GraphViewDataInterface**

  User can use his own model as data, by implementing GraphViewDataInterface.
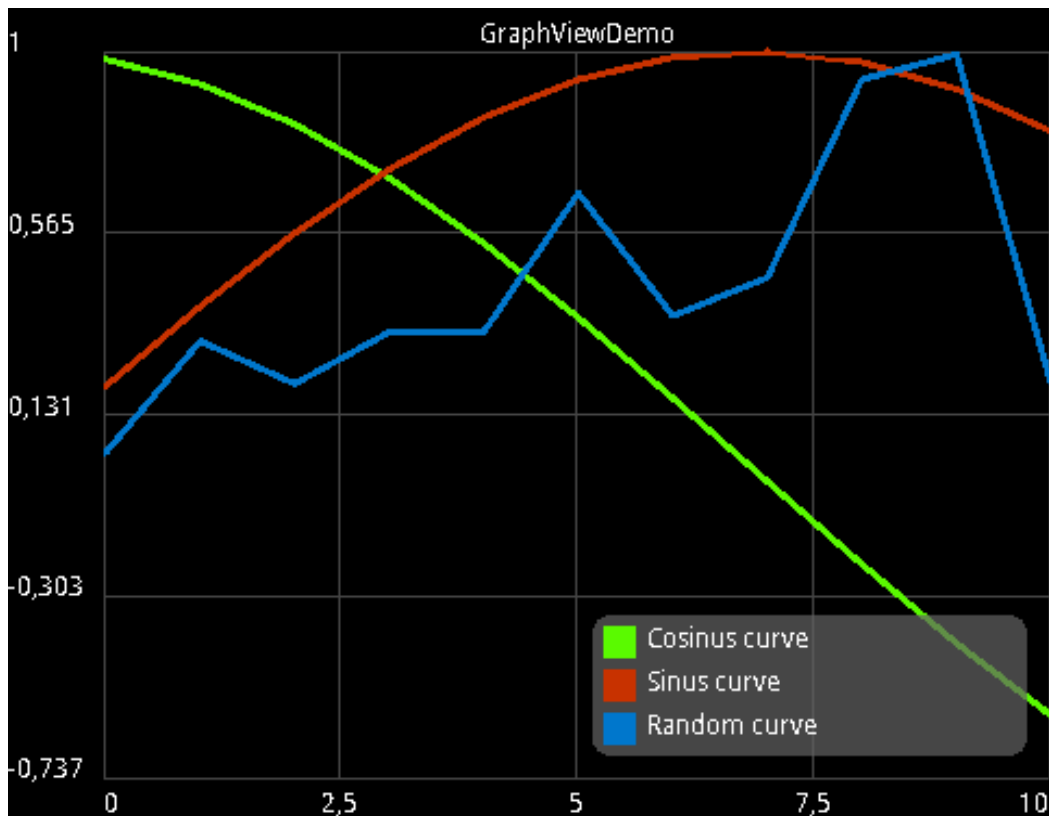
Figure 15 : A graph using GraphView.

University of Thessaly

4.1.8 Shared Preferences

Shared Preferences are used to save and retrieve data, where values will persist across user session. Data in Shared Preferences will be persistent even though user closes the application.

- **getSharedPreferences()** is the method to get values from Shared Preferences.
- An **Editor** is needed to edit and save data changes in Shared Preferences.
- Shared Preferences can store data; **Booleans**, **floats**, **ints**, **longs** and **strings**.

University of Thessaly

4.2 Main features of the app

Application can control, monitor and configure five plugs and it is constituted by; a fragment to control the plugs, a fragment to monitor and another one to configure them, a "Settings" window and an Activity to handle the real-time graph.

All http requests are made by an Async task class called *RequestTask* and response is taken back by the use of a listener and the *OnLoadFinishedListener* interface.

4.2.1   Fragment Control

The first Fragment (Figure 16) is used to control the plugs, turning them on and off. Toggle buttons are used to change the setting between two states. Whenever a button is clicked an http request is sent to the Gateway. The format of the http request is 192.168.1.20/x-on (or off), where x is the number of the plug you want to modify its state. Every time the Fragment starts (onResume function) an http request is sent to obtain the states of all plugs and set the ToggleButtons to the state that the plug is.

Figure 16 : Fragment Control.

4.2.2 Fragment Monitoring

The second Fragment (Figure 17) is used to monitor the measurements of concern device. In this application we are concerned about the instantaneous and cumulative consumption measured in Amperes. So when the Fragment begins, the application starts to send http requests, receive measured values and project these measurements to the proper fields. When the Gateway does not send data from a plug the field of this plug remains empty. The send request interval is defined in a variable that is saved in Shared Preferences. As the Gateway refreshes every one second there is no sense of asking for measurements faster than one second. The default send request interval is 1000 msec (1 sec).

In order to extract the needed measurement values from the Gateway's reply, proper parsing, to the reply String, has been done. The response is returned to String type with the form we saw in 4.2, thus first the response is separated in lines with the command "split("\\r?\\n")" based on "\n" character and each row is stored to a String array. After each line is separated we separate again each line based on spaces, using the command "split("\\s+")" and stored again to another String array. Therefore as the format is known, the needed part is pulled from the final array.

Figure 17 : Fragment Monitoring.

A live graph has been incorporated by using GraphView Library that shows the instantaneous power consumption of each plug. The plugs that are not connected are set to 0.0. When the button "Realtime consumption" is pressed a new Activity is called to perform the Real-time graph. A snapshot of the instantaneous consumption of Plug 1 (red line) is depicted in Figure 18. The other four plugs have zero consumption, so their lines are stack to zero.
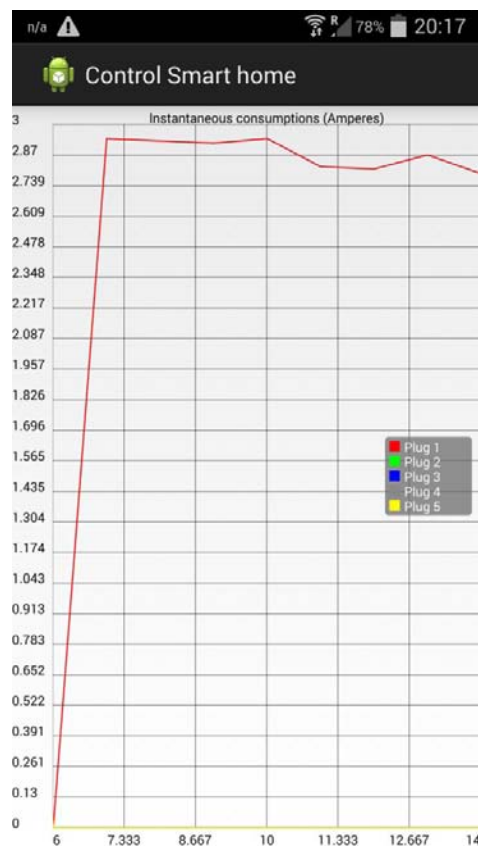

Figure 18 : Real time graph.

4.2.3 Fragment Configure

Third and last Fragment (Figure 19) deals with the configuration of the Gateway and sensors. There are buttons to calibrate and reset the plugs. Each button sends the http request for the corresponding plug. Calibrate button tunes the sensors located on the power meter device, without needing any electrical device plugged on power meter device. Reset button sets to zero the counters; Amperes per hour, Watts per hour and time since the last reset. By hitting submit button five http requests are send respectively to the five power meter devices and sets new parameters of sampling. There are three EditText fields for each device; Samples per period, periods per measurement and measurements interval. Each field has constrains to restrict user of inserting incorrect values. More specifically;

- **Samples per period** should be an integer between 10 - 80. In case user leaves empty the field the smaller value allowed (10) is automatically set. If the user sets a smaller value than the one allowed the lesser value is automatically set (10) and respectively when a larger value is inserted the larger allowed(80) value is automatically set.
- **Periods per measurement** should be an integer between 1 - 50. The same mechanism of checking the correctness of the values exists with lower bound 1 and upper bound 50.
- **Measurment Interval** should be an integer between 0 – 60000. Again the same mechanism exists with lower bound set to 0 and upper one set to 60000.

Values of EditText field are stored in Shared Preferences so user doesn't to set new values every time he opens the application.
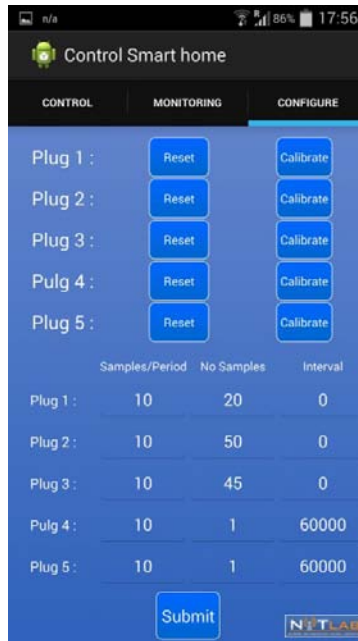
University of Thessaly

Figure 19 : Fragment Configure.

4.2.4   Settings window

Android devices occupy a menu key, which displays a list of options available for the current application. The application includes a menu (Figure 20) that is available to all three Fragments. Two attributes are customizable:
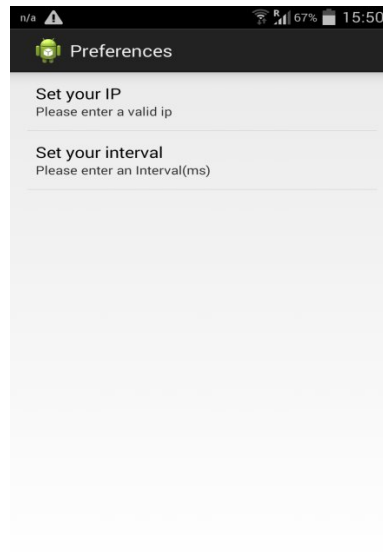

Figure 20 : Settings window.

- **Gateway's IP (Figure 21)**

  The ip field has IPv4 validator by using a Regular Expression. If the ip is not valid or the field is empty the default ip is set to "**0.0.0.0**".
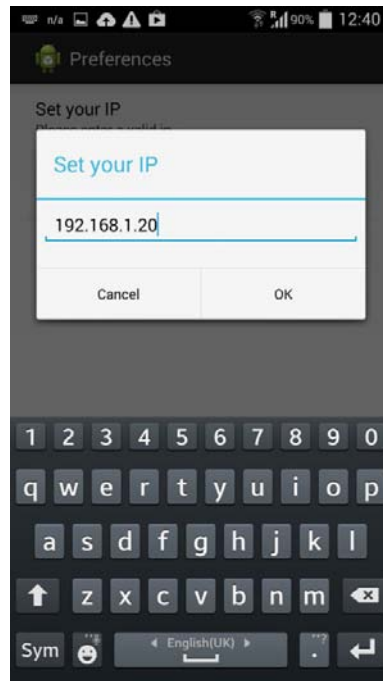
Figure 21 : Setting IP.

- **Request Interval (Figure 22)**

  The request interval is used to customize the rate that the application sends http requests to the Gateway for the monitoring Fragment and the Real-time graph. If the field is empty the default rate is **1000 msec**.
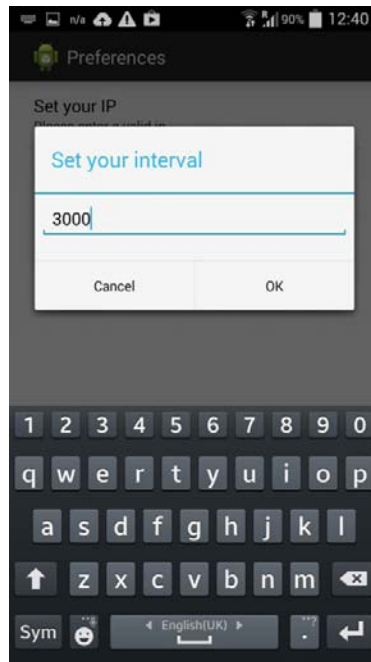
University of Thessaly

Figure 22 : Interval Setting.

5. Use Cases

To demonstrate the application, home electrical devices will be plugged to the power meter device, measured and configured. A laptop, a stereo, a lamp and a coffee machine.

In the Control section we note the ToggleButton, changing on and off. In figure 23 we see the device on "ON" and on "OFF" state.
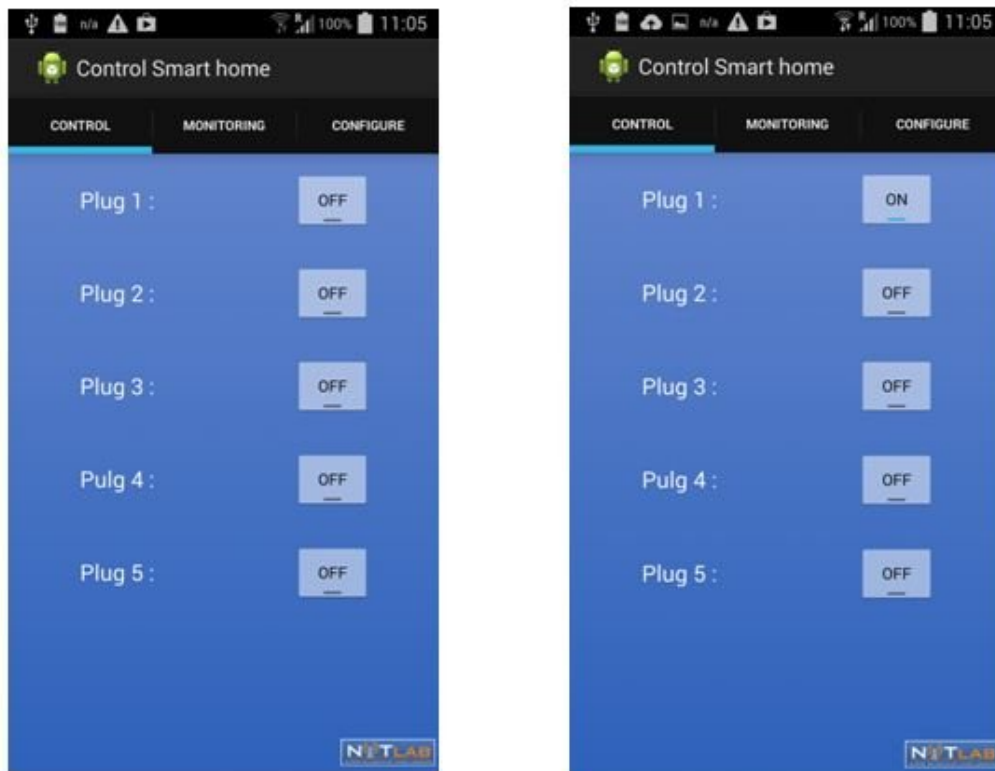

Figure 23 : ON / OFF state of button.

5.1 Measure home electrical appliances

To exhibit the application measurements have been taken to various home devices. In Figure 24 we measure a laptop.
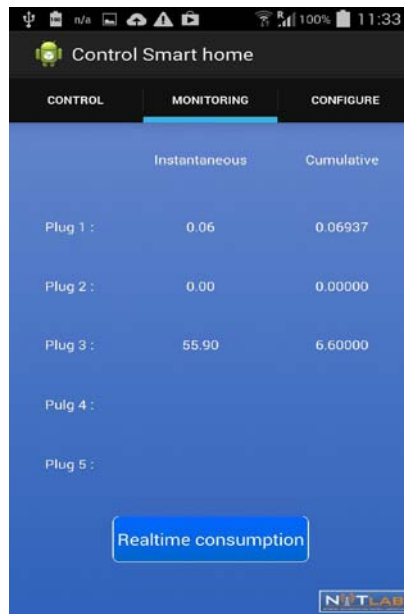

Figure 24 : Laptop consumption.
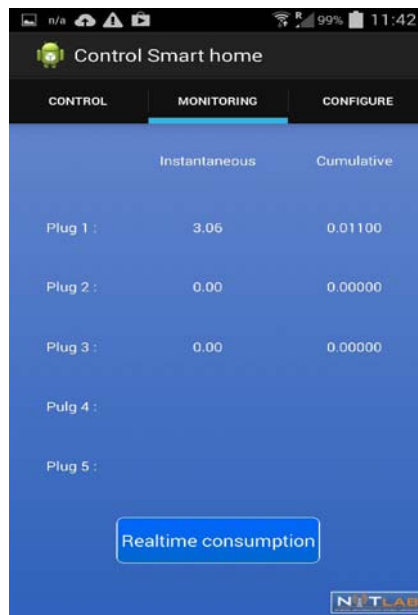
In Figure 25 a coffee machine is measured.
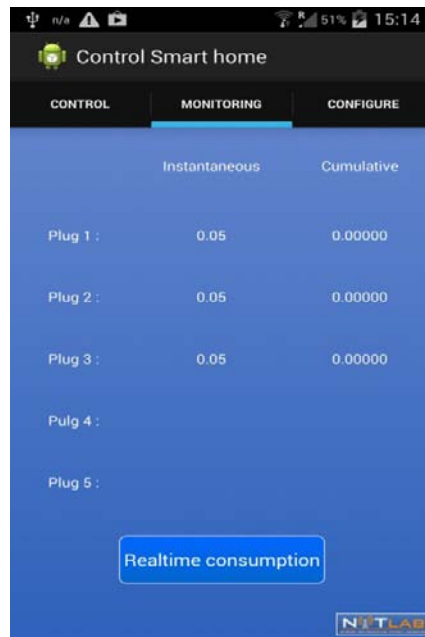

Figure 25 : Coffee machine consumption

35

University of Thessaly

In Figure 26 a lamp is measured.

Figure 26 : Lamp consumption.

And finally a stereo is measured in Figure 27

Figure 27 : Stereo consumption.

University of Thessaly
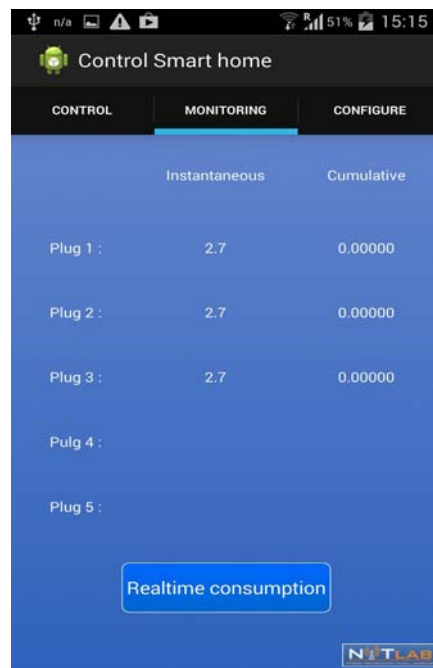
A graph that incorporates the measurement of instantaneous consumption is shown in Figure 28.
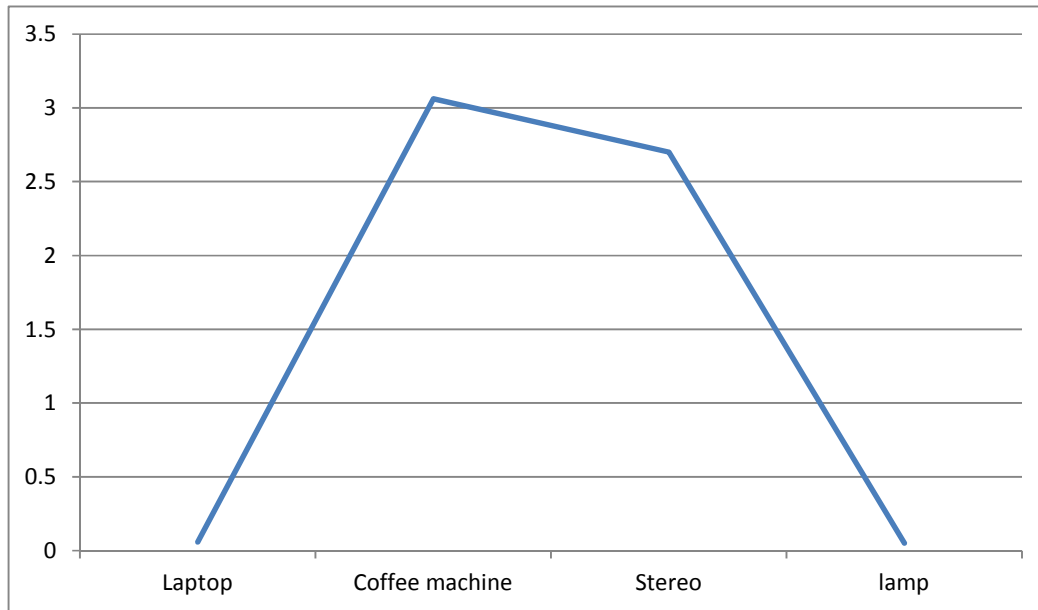

Figure 28 : Graph of consumptions.

6. Future work

A Bluetooth way of controlling devices was also examined, as an alternative to wireless communication. By using a Bluetooth modem and an Arduino Ethernet board an Android application was constructed to control a led. In Figure 1 it is depicted the topology of the Bluetooth modem and the Arduino board as well with the entire wiring scheme between them. The Bluetooth modem has a PWR pin to get
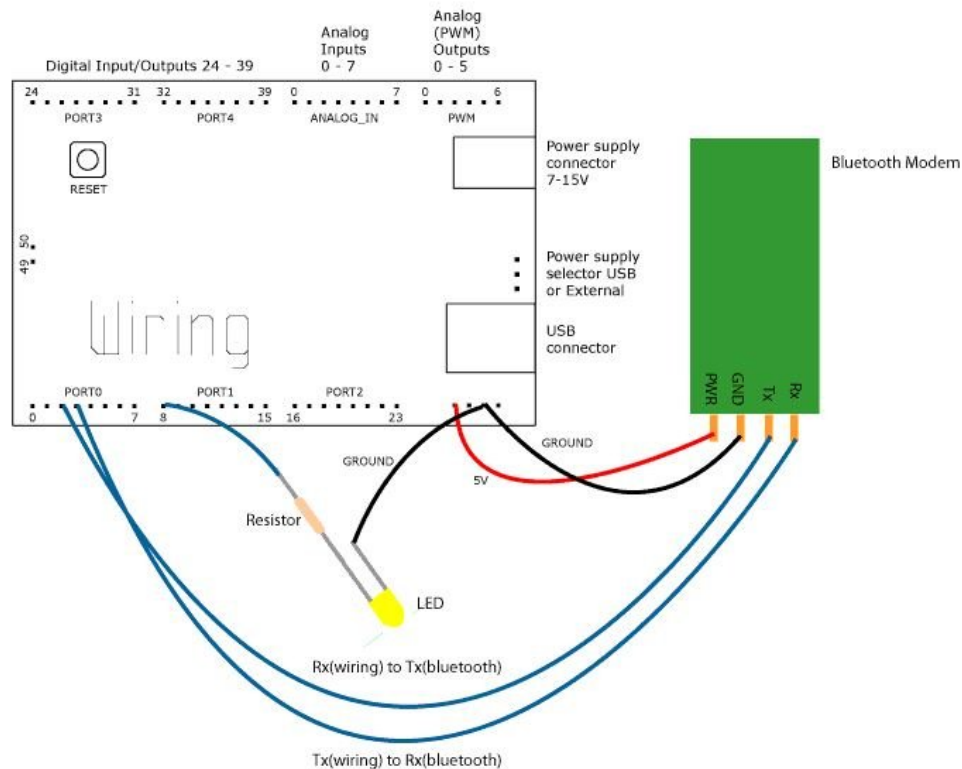

Figure 29 : Arduino – Bluetooth modem topology

power, a GND pin connected to the ground, a Tx pin to transmit data and a Rx pin to receive data. On the Arduino board the Tx is connected with the Rx of the modem and Rx on board connected to Tx of the modem, so as Bluetooth modem and Arduino board can communicate. Arduino board works using the code below, where the application could send characters to the Bluetooth modem, if the character was an "H" the led went "ON" and if the character "L" the led went "OFF".

University of Thessaly

```
CODE:

void setup() {

pinMode(ledpin, OUTPUT); // pin 48 (on-board LED) as OUTPUT
Serial.begin(9600); // start serial communication at 9600bps


}


void loop() {

if( Serial.available() ) // if data is available to read
{
val = Serial.read(); // read it and store it in 'val'
}
if( val == 'H' ) // if 'H' was received
{
digitalWrite(ledpin, HIGH); // turn ON the LED
} else {
digitalWrite(ledpin, LOW); // otherwise turn it OFF
}
delay(100); // wait 100ms for next reading
```

Hardware used (Figure 322):

- **Bluetooth Modem - BlueSMiRF HID**
  BlueSMIRF is a Bluetooth modem, simple and powerful tool for creating
  wireless peripheral devices which can be universally recognized and used
  without the installation of special drivers.
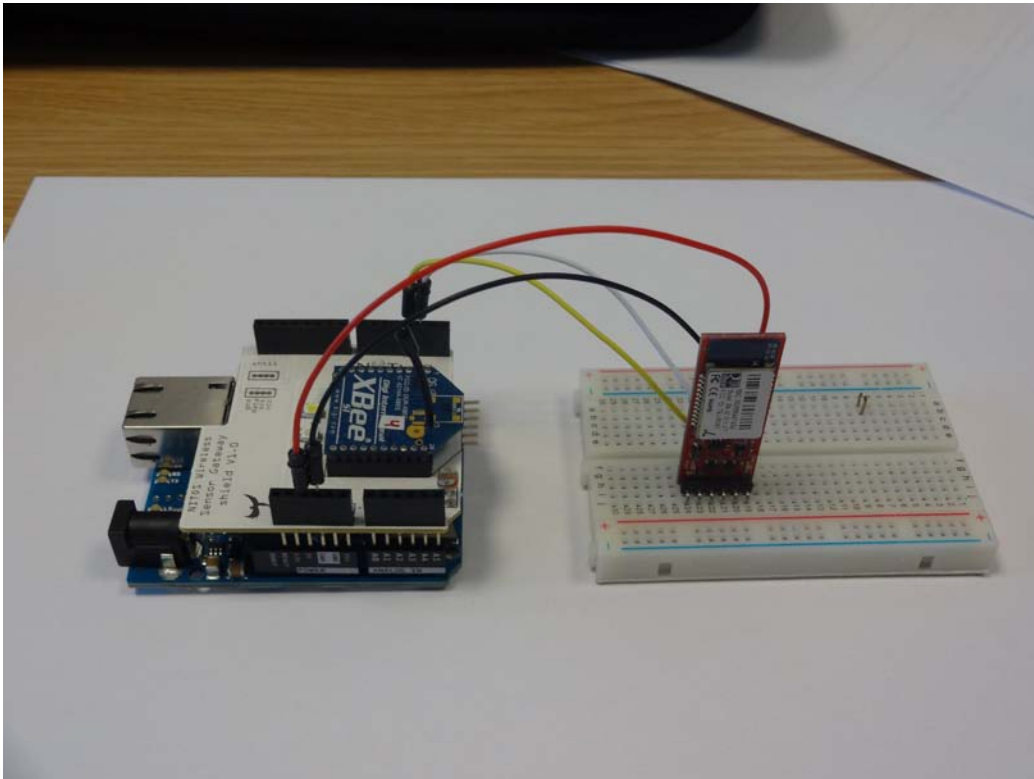
39

- **Arduino Ethernet Board**
  See in 2.2.

Figure 30 : Harware topology

Some Andoid key features [9] to use and handle Bluetooth are:

- **BluetoothAdapter**

  Represents the local Bluetooth adapter. The BluetoothAdapter is the entry-point for all Bluetooth interaction. Using this, user can discover other Bluetooth devices, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for communications from other devices.

40

- **BluetoothSocket**

  Represents the interface for a Bluetooth socket (similar to a TCP Socket). This is the connection point that allows an application to exchange data with another Bluetooth device via InputStream and OutputStream.

- **Connecting Devices**

  In order to create a connection between applications on two devices, user must implement both the server-side and client-side mechanisms, because one device must open a server socket and the other one must initiate the connection (using the server device's MAC address to initiate a connection). The server and client are considered connected to each other when they each have a connected BluetoothSocket on the same RFCOMM channel. At this point, each device can obtain input and output streams and data transfer can begin, which is discussed in the section about Managing a Connection.

- **Managing Connection**

  When successfully connected two (or more) devices, each one will have a connected BluetoothSocket. Using the BluetoothSocket, the general procedure to transfer arbitrary data is simple:

  o Get the InputSteram and OutputSteam that handle transmissions through the socket, via getInputStream()and getOutputStream(), respectively.

  o Read and write data to the streams with read(byte[]) and write(byte[]).

University of Thessaly

7. References

[1] NITLab homepage.

[2] NITLab's power meter device, link.

[3] Eclipse ADT, link.

[4] Pro Micro – 3.3V/8MHz, link.

[5] ACS712 current sensor, link.

[6] Xbee S2, link.

[7] Arduino Ethernet Board, link.

[8] Curl main page.

[9] Android Bluetooth, link.