A bottom up sensor testbed

Sergio Almendros Díaz

TFG UPF / YEAR 2014

DIRECTOR/S OF THE TFG:
Jaume Barceló
DEPARTMENT:
Departament de Tecnologies de la Informació i les Comunicacions (DTIC)

Universitat
Pompeu Fabra
Barcelona

Acknowledgments

Abstract

This project aims to deploy a sensor node network taking advantage of the Guifi network, an open, free and neutral network, and the visualization of this data in an intuitive and meaningful way, tailoring the presentation of the data to people without specific scientific knowledge.

The main objective is to give citizens a trace to improve their environmental conscience, giving them the necessary information to know the state of their surroundings and so they can act accordingly. This way anybody can act consequently if the sensors detect harmful situations, for example, not going to a park where the gas sensor detects bad air quality.

Arduino Yun is the main component of the sensor node. Through a series of sensors attached to it and the Internet connection provided by Guifi, the node can communicate with a storage platform for sensor data, in this case, Opencities.

Finally, an Android application downloads the sensor data from Opencities and display them on a map.

Resum

Aquest projecte té com a objectiu desplegar una xarxa de nodes sensors prenent avantatge de la xarxa Guifi, una xarxa oberta, lliure i neutral, i la visualització d'aquestes dades d'una forma intuïtiva i significativa, adaptant la presentació de les dades per a persones que no tenen coneixements científics específics.
L'objectiu principal és donar als ciutadans un camí per millorar la seva consciència ambiental, donant-lis la informació necessària per conèixer l'estat del seu entorn i que així puguin actuar en conseqüència. D'aquesta manera tothom pot actuar en conseqüència si els sensors detecten situacions perilloses, per exemple, no anar a un parc en el qual el sensor de gas detecta una dolenta qualitat de l'aire.
Arduino Yun és el component principal del node sensor. A través d'una sèrie de sensors connectats a ell i la connexió a Internet proporcionada per Guifi, el node es pot comunicar amb una plataforma d'emmagatzematge per a dades de sensors, en aquest cas, Opencities.
Finalment, una aplicació per Android descàrrega les dades de sensors d'Opencities i els mostra en un mapa.

Contents

List of figures **xii**

1 INTRODUCTION **1**

2 STATE OF THE ART **3**
2.1 Introduction 3
2.2 Sensor networks and smart cities 4
2.2.1 Amsterdam smart city 4
2.2.2 Santander smart city 4
2.3 Companies 4
2.3.1 Smartcitizen 4
2.3.2 Libelium 5
2.4 Opendata services 5
2.4.1 Opencities 5
2.4.2 Xively 6
2.4.3 Sentilo 6
2.5 Sensor boards 6
2.5.1 Arduino YUN 6
2.5.2 Raspberry Pi 8
2.5.3 Picoboard 8

3 TECHNOLOGIES **11**
3.1 Arduino-Raspberry PI comparison 11
3.2 Sensors 12
3.2.1 LM35: Temperature 12
3.2.2 Light Dependent Resistor (LDR) 12
3.2.3 Emartee Mini Sound Sensor and Analog Sound Sensor Board Microphone MIC Controller: Noise 14
3.2.4 Aosong DHT22 and DHT11: Humidity 14
3.2.5 MQ135: Gas sensor 14
3.2.6 BreadBoard with all the sensors 16

3.3 Upload sensor data 16
3.3.1 GeoJSON 16
3.4 Community network 19
3.5 Storage Resource Broker 19
3.6 Visualization platform 21

4 BOTTOM UP SENSOR TESTBED 23
4.1 Arduino Development 24
4.1.1 Collect sensor data 24
4.1.2 Communication with Opencities 24
4.2 Android app 26
4.2.1 Summary 26
4.2.2 Interface 26
4.2.3 Code 26
4.3 Repository 31

5 TESTBED DEPLOYMENT AND RESULTS 33
5.1 Sensor node 34
5.1.1 Connection to the Internet 34
5.1.2 Install necessary packets 36
5.1.3 Copy the scripts 36
5.1.4 Attach the sensors 36
5.1.5 Arduino Code 36
5.2 Actual Testbed 37
5.2.1 Results 37

6 CONCLUSIONS 41

7 FUTURE WORK 43

BIBLIOGRAPHY 45

A PILOT CHARTER 47
A.1 Pilot Charter 47
A.1.1 Pilot purpose or justification 47
A.1.2 Measurable pilot objectives and related success criteria . . 47
A.1.3 High-level requirements 47
A.1.4 High-level pilot description 48
A.1.5 High-level risks 48
A.1.6 Summary milestone schedule 48
A.1.7 Summary budget 49

B PLANNING REPORT **51**
B.1 Planning Report 51
B.1.1 Familiarization with the Arduino Yun 51
B.1.2 Preliminary testbed 51
B.1.3 Collect Data from sensors 51
B.1.4 Install Sentilo 52
B.1.5 Communication with Sentilo 52
B.1.6 Real deployment 52
B.1.7 Interface 52
B.1.8 Sentilo module 52
B.1.9 Final report 52
B.1.10 Gantt chart 53

C CLASS DIAGRAM **55**

D TESTBED GRAPHICS **59**

List of Figures

2.1 Smart Citizen Node 5
2.2 Libellium Projects 6
2.3 Arduino YUN 7
2.4 Arduino YUN Bridge 7
2.5 Raspberry Pi model B 8
2.6 Picoboard 9

3.1 Sensor boards 12
3.2 Temperature sensor 13
3.3 Light sensor 13
3.4 Noise sensor 14
3.5 DHT22 sensor 15
3.6 DHT11 sensor 15
3.7 DHT11 sensor 16
3.8 Sensor node Prototype with DHT22 17
3.9 Sensor node Prototype with DHT11 18
3.10 GeoJSON message 19
3.11 Guifi sensor map 20

4.1 General View 23
4.2 Arduino sketch Flow Chart 25
4.3 Python Script Flow Chart 27
4.4 App Screenshots 28
4.5 Reduce Class Diagram of the Android App 29
4.6 Android App Flow Chart 30
4.7 Github Repository 31

5.1 TestBed Prototype 33
5.2 Yun web Password 34
5.3 Yun web Diagnostic 35
5.4 Yun web Configuration 35
5.5 Testbed Map 37

5.6 Testbed Results 38
5.7 Testbed Results 39
5.8 Graphic Air Quality 40

B.1 Gantt Chart 53

C.1 Class Diagram of the Android App part 2 57

D.1 Complete Graphic Air Quality 59
D.2 Complete Graphic Humidity 60
D.3 Complete Graphic Light 61
D.4 Complete Graphic Noise 62
D.5 Complete Graphic Temperature 63

Chapter 1

INTRODUCTION

The development of this project has been divided into three phases: collect data from environmental sensors, display it to end users in an intuitive way, and make a real testbed to demonstrate its performance.

As a sensor node has been used an Arduino[Arduino, 2014] YUN, which allows to obtain easily analog reads from a sensor. With a Power over Ethernet module, it can be attached to Guifi[1] nodes and send the sensor data to a sensor platform, e.g. Opencities [Opencities, 2014].

The Bottom-up[2] pattern has been used to build the sensor testbed, where the end users, in this case, Guifi users, are the ones who have to assemble the sensor nodes and connect them to their Guifi nodes to create the sensor network. With the bottom-up model, the data is provided and used by the end users.

In this project an Android application has been developed to visualize the sensor data and make it accessible to the end users. A sensor testbed has been deployed to gather sensor data, and test the technologies used and developed.

This project is an attempt to understand the structure, the technological challenges and operations of sensor networks and how they can help us to take conscience about our environment.

The thesis is structured as follows: the actual state of sensor networks (Chapter 2), which technologies had been used (Chapter 3), and how the project has been developed (Chapter 4). In chapter 5 the testbed deployment is illustrated and some resulting data are shown. Finally, the conclusions (Chapter 6) and future work (Chapter 7).

[1] `http://guifi.net/`
[2] `http://bubforeurope.net`

Chapter 2

STATE OF THE ART

2.1 Introduction

Sensor networks started as a mechanism of defense developed by the United States Navy during the Cold War. With acoustic sensors they searched for Soviet submarines. This research continued at Universities, trying to make the sensors smaller, and with the possibility of real-time data [Chong and Kumar, 2003].

Sensor networks should not affect the environment, in our case, visually. For this purpose we must ensure that the nodes are relatively small, so the board and sensors had been selected accordingly.

To choose the processor board, it has to be look into the following characteristics:

- **Power:** A sensor node can obtain energy in several ways: batteries, solar energy, connecting to an already deployed network... The important thing is not to increase the size of the sensor node, in our case, is power by an already deployed network.

- **CPU:** A powerful CPU is not necessary to process sensor data, so this helps to decrease the size of the board and reduce power consumption.

- **Communication:** There has to be a way to send the collected data outside, usually wireless or through a wired network. If it has wireless communication, the topology and location of the sensor nodes has to be chosen properly. But if it is through a wired network, it should take advantage of something already deployed, in our case, the Guifi network.

Currently the sensors are small enough to put several into a shield with the same size as the board mentioned before.

Smart cities are the next step. A city capable of having real-time information, not only about the environment, but also from the number of cars that pass on a road, to the amount of rain water in a day. This kind of information could help to manage more efficiently the city.

It is important to share this information so everyone can know the state of their environment. There are already some sensor networks functioning, some of them are government-funded and they often apply a policy where the data is not completely public, while other sensor networks are private-owned and the data are possibly open (e.g. smartcitizen).

2.2 Sensor networks and smart cities

In this section a few projects on sensor networks deployed are introduced:

2.2.1 Amsterdam smart city

Amsterdam has many projects concerning the smart city concept, like the "Flexible street lighting", which allows the government to monitor the street and switch off the lights to save energy, or the "Smart parking" which let drivers to know if there are free spots to park, and, in consequence, reduce air pollution [smart city, 2014].

2.2.2 Santander smart city

Santander has his own sensor network testbed for environmental monitoring, outdoor parking area management, and traffic intensity monitoring [santander, 2014].

2.3 Companies

In the following sections some companies that are in the business of sensor networks are described.

2.3.1 Smartcitizen

Smart Citizen[1] is platform that offers a sensor board based on Arduino to monitor the environment (figure 2.1), and permits to upload this data to their own public database.

[1] `http://www.smartcitizen.me/`

Figure 2.1: Smart Citizen Node.

2.3.2 Libelium

Libelium is an Internet of things platform provider[2], which supplies an open source sensor platform for the Internet of things. They have a variety of products, some interesting ones are shown:

- e-Health: A sensor shield for Arduino and Raspberry Pi for body monitoring: pulse, oxygen in blood, airflow (breathing), body temperature, electrocardiogram, glucometer, galvanic skin response, blood pressure, patient position, and muscle/electromyography sensor (Figure 2.2a).
- Waspmote: A sensor node where it is possible attach more than 60 sensors, solar powered, and though to fit onto street light poles. (Figure 2.2b).
- Smart Water: A wireless sensor platform for water quality monitoring, it provides real-time data. (Figure 2.2c).

2.4 Opendata services

The sensor boards are continuously collecting data, which has to be stored. The data can be stored remotely or locally, but collecting the data from every sensor board it is unpractical, for this reason an online storage service, opendata, is needed.

2.4.1 Opencities

Opencities [Opencities, 2014] is a platform to browse, visualize, and download open data from different participants. Opencities has a very simple API to upload and download data, a web page to visualize the stored data.

[2] `http://www.libelium.com/`

(a) Libelium ehealth (b) Libelium Waspmote (c) Libelium Smart Water

Figure 2.2: Libellium Projects

2.4.2 Xively

Xively [3] offers an Internet of Things platform as a service, basically it lets you store sensor data, download it, and visualize it through graphics.

2.4.3 Sentilo

Sentilo [4] is an open source platform for Smart Cities, it allows the user to use their own service to store the data, but they do not intend to become the database for everyone, it is only as an example. The Sentilo platform does not depend on a central server and can be installed on a private server. It also provides a interface to show the data.

2.5 Sensor boards

In this section are discussed some of the options for a sensor node.

2.5.1 Arduino YUN

The Arduino YUN is a micro controller board with two processors (figure 2.3), an ATmega32u4 (Arduino), and an Atheros AR9331 (Which runs a Linux distribution named OpenWrt-Yun). It has an Ethernet and WiFi module, a USB-A port, a micro-SD card slot, 20 digital input/output ping, 16 MHz crystal oscillator, ICSP header, and 3 reset buttons.

[3] `https://xively.com`
[4] `http://www.sentilo.io`

Figure 2.3: Arduino YUN.

The peculiarity about the Arduino YUN is that the processor for the Arduino sketches can communicate with the Linux processor through the bridge library (figure 2.4), which allows to write python scripts and execute them.

A power over Ethernet module (PoE) can be attach to the Arduino, which has a particular interest for the Guifi network, where the antennas are powered through PoE.

Figure 2.4: Arduino YUN Bridge.

2.5.2 Raspberry Pi

Raspberry Pi[5] is a single-board computer, produced in two models, A and B. The model B (Figure 2.5) is more appropriate for this project because it has an Ethernet controller. It is composed by an HDMI, a micro USB, an USB 2.0 connector, an SD card slot, Input/Output (GPIO) pins, an RCA connector, an audio jack, an Ethernet controller, and a Broadcom BCM2835 processor.

It is possible to attach sensors to it, and it supports Linux environment.

Figure 2.5: Raspberry Pi model B.

2.5.3 Picoboard

PicoBoard[6] (Figure 2.6) is a board to interact with the world, it can be programmed by Scratch projects. It is less flexible than the previously mentioned, it is composed by a button, a light and a sound sensor, a slider, and alligator clips which can be use to build custom sensors.

[5] http://www.raspberrypi.org/
[6] http://www.picocricket.com/picoboard.html

Figure 2.6: Picoboard.

Chapter 3

TECHNOLOGIES

This chapter is focused on the technologies used to develop this project.

3.1 Arduino-Raspberry PI comparison

For this project, two possible boards have been considered: the Arduino Yun (Yun) and the Raspberry Pi model B (RPiB).

The Yun is a microcontroller while the RPiB is a full computer, which makes it more powerful, even more than needed, so both Yun and RPiB serve.

The two of them have a Linux environment, but with the Yun, the normal way to interact is by an Arduino sketch. So RPiB considers that the developer has some prior Linux knowledge, while the Yun is better for beginners. Also, the Arduino IDE provides a variety of programs that help to start.

The sensor board is the biggest element of the sensor node, so the size is very important, the smaller the better. In this case, the Yun is smaller than the RPiB. On the other hand, the sensor node will be attach on Guifi nodes, which are powered by Power over Ethernet (PoE). For the Yun there is the possibility of a PoE module (not available yet), which will power the board. The figure 3.1a and 3.1b show the comparison between the size of an Arduino Ethernet (similar to the YUN) and the RPiB both powered by PoE. It is clearly visible that the Arduino is more compact.

Another advantage that YUN offers is the integrated WiFi module, which offers more flexibility when there is no Ethernet available.

Finally, the Arduino Yun has been used instead of the Raspberry Pi basically because of the size that will become the sensor node, but if in the future will be necessary a more powerful processor, it will be better to use a Raspberry Pi.

(a) Arduino Ethernet PoE

(b) Raspberry Pi (B) PoE

Figure 3.1: Sensor boards

3.2 Sensors

The goal is to analyze the environment around us, and sensors are a mean to detect physical or chemical variables. It has been decided to use low cost environmental sensors that are easily accessible and usable. These sensors measure the aspects that may be more useful for citizens: temperature, light, noise, humidity, and air quality.

A sensor is a device which transform a physical measure to an output signal (Voltage or Digits) that can be read by another device, such as an Arduino. In this project five sensors had been used to measure temperature, light, noise, humidity, and gas. To show how the sensors are connected to the Arduino YUN the program fritzing[1] has been used.

3.2.1 LM35: Temperature

LM35 [Figure 3.2a] is a sensor to measure temperature, in the figure 3.2b is shown how to connect it to the Arduino. [Instruments, 2013]

3.2.2 Light Dependent Resistor (LDR)

LDR [Figure 3.3a] is a light sensor, in the figure 3.3b is depicted how to connect it to the Arduino.

[1] `http://fritzing.org/`

(a) LM35 sensor

(b) Temperature Sensor Breadboard

Figure 3.2: Temperature sensor

(a) LDR sensor

(b) Light Sensor Breadboard

Figure 3.3: Light sensor

(a) Mini Sound Sensor

(b) Analog noise sensor

(c) Noise Sensor Breadboard

Figure 3.4: Noise sensor

3.2.3 Emartee Mini Sound Sensor and Analog Sound Sensor Board Microphone MIC Controller: Noise

This two sensors [Emartee, 2013] are used to measured noise levels [Figure 3.4a] and [Figure 3.4b]. The code to read the noise values is the same for both. In the figure 3.4c is shown how to connect them to the Arduino.

3.2.4 Aosong DHT22 and DHT11: Humidity

DHT22 [Figure 3.5a] and DHT11 [Figure 3.6a] are humidity and temperature sensors, although the humidity measure is the only one used. The output is digital, and to read it, an external library[2] is needed. The Arduino and the humidity sensor have to be connected as shown in the figure 3.5b for the DHT22 and as shown in the figure 3.6b for the DHT11 [Aosong Electronics Co., 2013].

3.2.5 MQ135: Gas sensor

MQ135 is a gas sensor [Figure 3.7a], and is used to measure air quality. Figure 3.7b illustrates how to connect it to the Arduino.

The Mq135 sensor reacts to the concentration of the following gases: NH3, NOx, alcohol, Benzene, smoke, CO2, etc.. [Futurlec, 2013].

[2] `https://github.com/adafruit/DHT-sensor-library`

(a) DHT22 sensor

(b) DHT22 Sensor Breadboard

Figure 3.5: DHT22 sensor

(a) DHT11 sensor

(b) DHT11 Sensor Breadboard

Figure 3.6: DHT11 sensor

(a) MQ135 Air Quality sensor

(b) Gas Sensor Breadboard

Figure 3.7: DHT11 sensor

3.2.6 BreadBoard with all the sensors

In the figure 3.8 is shown the final prototype with the DHT22 sensor, and in the figure 3.9 with the DHT11 sensor.

3.3 Upload sensor data

A main part of the project is upload the data from the sensors to a platform so that everyone can access them.

The GeoJSON message (Explained in section 3.3.1) includes data from 5 sensors, which makes its size too large for the memory of the Arduino. For this type of operations is possible to run a script in the Linux environment through the bridge library (Figure 2.4).

This being the path that is followed, there are several options for developing this script, for example, programmed in Java, C / C + +, python, etc..

Python has been used because of the following reasons: fast and perfect for prototyping programming, the code is shorter and therefore, easier to understand, and modular.

3.3.1 GeoJSON

A GeoJSON[3] is a format for encoding a variety of geographic data structures. The GeoJSON that has been used is a collection of features, every feature contains a

[3] `http://geojson.org/`

Figure 3.8: Sensor node Prototype with DHT22.

Figure 3.9: Sensor node Prototype with DHT11.

geometry object, in our case, a “point” with the longitude and latitude of the sensor node, and some properties required: ID, name, datasetID, datasetName, address, description, timestamp, value of the sensor, and unit.

The figure 3.3.1 shows an example of a GeoJSON message that is used in this project.


```
{
    "type": "FeatureCollection",
    "name": "dummy",
    "timeStamp": "2014-06-12T08:54:59.424Z",
    "features": [
        {
            "type": "Feature",
            "tags": [
                "red",
                "tall",
                "cheap",
                "upf"
            ],
            "geometry": {
                "type": "Point",
                "coordinates": [
                    2.18946,
                    41.403809
                ]
            }
        }]
}
```


Figure 3.10: GeoJSON message.

3.4 Community network

Guifi [4] is the network where the Arduinos will be deployed, and the one providing the access to Opencities through the Internet. Guifi is a network created by people interested in building an open, free and neutral network infrastructure.

Because of their philosophy of participation, Guifi is the perfect network for the deployment of this sensor nodes. Also their network is large enough to become a useful sensor network (Figure 3.11).

3.5 Storage Resource Broker

Opencities [Opencities, 2014] is the opendata service that has been chosen, for these reasons:

- The developers are at UPF, so the process of improving both projects (feedback, bug fixing, ...) can be fast and effective.

[4] https://www.guifi.net/

Figure 3.11: Guifi sensor map.

- Easy API to upload and download the data.

The role of Opencities is to be the intermediary between data creators and data users. In Opencities, the sensor data is uploaded into a dataset. Every user has an unique API key, and can create one or more datasets with a unique ID for each one.

3.6 Visualization platform

For the purpose of the sensor data visualization there are several options, in plain text, in a map, in graphics, etc. In this project a map is used to display the data. This can be done on a web page, in a mobile application, tablet application, etc.

Since the goal is that a user checks it for a small period of time, the best option is a mobile application. Additionally, almost everybody has an smartphone.

There are three mobile operating system, iOS, Android, or Windows Phone. There is another option that is to developed the application in html, javascript, and CSS, and then compile it with phonegap[5], and it builds the application for the three operating systems. But it ended up being much more difficult than to create an application directly to an operating system. Of the three operating systems, the one with more market share has been chosen to reach as many people as possible.

Android is a mobile operating system from Google, it runs on smartphones, and applications are programmed in Java. Java is a computer programming language that is object-oriented. This application will be tested on a Sony Xperia Z1, with an Android 4.4.2.

[5] `http://phonegap.com/`

Chapter 4

BOTTOM UP SENSOR TESTBED

This chapter is focused on the process followed to finish the project, which has two main parts, the software to collect and send the data and the Android application to show it. A scheme of the project is shown in Figure 4.1.

Figure 4.1: General View.

4.1 Arduino Development

Two scripts has been needed, one to collect the data, and other to send it. This is because the memory to run an Arduino sketch is very low, and the creation of the GeoJSON message is too big with respect to the available memory. That is why a python script has been used, called first by the Arduino sketch.

The Arduino sketch is responsible of collecting the data, write it down in a logData file, and call the python script with the collected values and an unique ID. Finally the python script has to create a GeoJSON message and send it to Opencities.

4.1.1 Collect sensor data

To collect almost all the data the sketch does not need to include external libraries, except for the humidity sensors (DHT22 and DHT11)[1].

The following libraries were needed to develop the code: FileIO, Process library, and Bridge.

The Arduino sketch is code in a very simple way, the setup function will initialize the Bridge library to communicate with the Linux environment, the Serial library for debugging purposes, and the FileSystem to log the process. The other function by default is the "loop", which calls three functions: readSensors, readFile, and executePythonScript.

- **readSensors:** It call 5 different functions to read every sensor, the reason for doing a separate function is to make the code more clear.
- **readFile:** This function logs the process, saves the ID of the message, the sensor values, and a timestamp.
- **executePythonScript:** The script in python located in the SD card is in charge to create the GeoJSON message with all the sensor data and upload it to Opencities. This script is called by the Arduino sketch.

The figure 4.2 explain how the Arduino sketch works.

4.1.2 Communication with Opencities

To communicate with Opencities it is necessary to create an account, and a dataset to store the sensor data. At the end, the user has an API key and a dataset ID, the required information to upload and download data.

[1] `https://github.com/adafruit/DHT-sensor-library`

Figure 4.2: Arduino sketch Flow Chart.

The communication with Opencities is done by means of a python script. This are the packages needed into the Linux environment of the Arduino: distribute, python-openssl, geojson, geopy, and httplib2.

The python script has three process, first it stores all the sensor data pass by the Arduino sketch into a class. Then it creates a GeoJSON message with all the data, and all the parameters needed to upload it into Opencities. Finally upload the GeoJSON message.

With all this libraries the script can communicate with Opencities and store the sensor data collected by the Arduino. The 4.3 figure explains how the python script works.

4.2 Android app

4.2.1 Summary

To make easy to visualize the results of the testbed, an Android application has been developed which shows the sensor data in a map. The application shows the data in two ways, with markers that show the actual value in that point, and also with heatmap points, the larger the value, the more intense the red will be (Figure 4.4).

4.2.2 Interface

The application interface is a unique map view where the user can zoom to a limit, go to his location, and use the top buttons to change the sensor read. From left to right, the first button is the Marker button, the user decides whether the markers are displayed or not, and the next buttons refer to the type of sensor data to show as markers and/or as heatmap points (Temperature, Humidity, Noise, Light, and Air Quality). If the Marker button is checked, the user can click on the marker in the map and it will show the value of the temperature, humidity,... and the unit (Figure 4.4).

4.2.3 Code

First of all, to create this application the Google Maps Android API v2[2] for the map view, and the Google Maps Android API utility library[3] for the heatmaps had been used.

[2] `https://developers.google.com/maps/documentation/android/`
[3] `http://googlemaps.github.io/android-maps-utils/`

Figure 4.3: Python Script Flow Chart.

Figure 4.4: App Screenshots

In this application each class has a defined function, starting with MainActivity, which is the one that initialize the other classes, calls all the functions needed in the other classes to get the application started.

It also has all the code to interact with the interface. For example, if the user clicks on the marker button, this class has to call all the necessary functions and display all the markers into the map.

Next comes the dataBase class, which, as the name suggest, is the class that stores and processes the data downloaded from Opencities. This is a Singleton class, which means that there will only be one instance of this class and any instance of any class could access to the data stored on it.

To stored the data from Opencities, there has been created a set of classes equivalent with the GeoJSON message downloaded to avoid confusion. At the end, the data is stored in features, which is composed primarily of Geometry and Properties.

Then a class called HttpAsyncTask that handles the communication with Opencities stores the downloaded data using an instance of dataBase.

Finally, a GPSTracker class is responsible for obtaining user location to place the map on the user location, because it will be the place that the users want to see.

The figure 4.5 shows a reduce class diagram.

The explanation of the code is in the flow chart 4.6.

Figure 4.5: Reduce Class Diagram of the Android App.

Figure 4.6: Android App Flow Chart.

4.3 Repository

All the code, report, figures, etc has been stored in a public repository[4], so anyone could see it, download a copy, and change it if necessary.

The figure 4.7 illustrates how the repository is distributed, each folder represents part of the process that has been done on this project.

Figure 4.7: Github Repository.

The most important folders are:

- **Sensor Code:** Where has been stored the code that go into the Arduino (Arduino sketch and python script).
- **WorkspaceAndroid:** This folder has been used as workspace for the developing of the mobile application using the IDE (Integrated Development Environment) Eclipse.
- **Final Report:** Is where it has been documented the project and has been written in latex, a system for the creation of technical and scientific documentation.

The less important folders are:

[4] `https://github.com/SergioAlmendros/A-bottom-up-sensor-testbed`

- **Collect Data From Sensors:** Here it has been stored the code to get the value of each sensor separately, in case someone wants to use only one of the sensors and does not want to read the full code.
- **Collect Data From Sensors Nodes:** Using the logData files that have been created during the testbed, an octave script reads this data and displays it in graphs.
- **Deliverables:** In this folder it has been stored the initial documentation of the project that are in the appendices A and B.
- **Presentation:** Finally, here is the slide presentation that has been made to explain the project, also written in latex.

During all the project there have been constant uploads of new information to the repository, as it was evolving.

Chapter 5

TESTBED DEPLOYMENT AND RESULTS

This chapter explains the procedure to deploy a testbed step by step.

In the figure 5.1 there is a photograph of the prototype used in this testbed. It is composed of an Arduino YUN, an Arduino UNO to power the Arduino YUN because the micro-USB was broken, a microSD card, a breadboard, and all the sensor connected (temperature, humidity, noise, light, and gas) to the Arduino YUN.

Figure 5.1: TestBed Prototype.

5.1 Sensor node

This section shows the process to configure the sensor node [Guide, 2014]:

5.1.1 Connection to the Internet

First of all an Internet connection has to be provided to the Arduino.

5.1.1.1 Through Ethernet

This is the fastest way to provide of Internet connection, the Arduino behave the same way as a computer, automatically obtains an IP address.

5.1.1.2 Through WiFi

This is the slowest way. The process is the following:

1. First power the Arduino YUN.
2. The Arduino creates his own WiFi network (ArduinoYun-XXXXXXXXXXXX), and with a computer connect to it.
3. When it is connected to the YUN network, go to a web browser and go to `http://arduino.local` or `192.168.240.1`. Put the password, which is "arduino", as shown in the figure 5.2.

Figure 5.2: Yun web Password.

4. The next page is an information page, click on the configure button (Figure 5.3).
5. Give a unique name to the Yun, and the network to connect (Figure 5.4).

Figure 5.3: Yun web Diagnostic.

Figure 5.4: Yun web Configuration.

6. Press the configure & restart button.

7. Finally connect to the same network as the Yun is connected.

5.1.2 Install necessary packets

The Arduino runs an Arduino script and a python script, for python scripts some packets have to be installed.

To install any packet, connect to the Arduino (as shown before):

ssh root@X.Y.Z.W

Now install all the necessary packets:

opkg update
opkg install distribute
opkg install python-openssl
easy_install install geojson
easy_install install geopy
easy_install install httplib2

5.1.3 Copy the scripts

First of all, it is necessary to create some directories, so once the ssh command has been done, go to "/mnt/sda1", make the following commands:

mkdir arduino
cd arduino
mkdir www

Copy the python script "main.py" into the SD-Card. There are two ways to do this, putting the SDCard into a computer an saving the file directly, or copying the file into the Arduino through the network with the following command:

scp main.py root@192.168.2.149:/mnt/sda1/arduino/www/main.py

5.1.4 Attach the sensors

Now that the python step is done, the sensors had to be attach to the Arduino Yun (Figure 3.8 or Figure 3.9).

5.1.5 Arduino Code

To upload an Arduino sketch to the Yun the IDE (Arduino 1.5.5) has to be used. There are two ways to upload a sketch, through an USB cable connected to the Arduino, or through the Internet. If the Arduino and the computer are in the same network, the Arduino will appear in the IDE as an option to upload.

5.2 Actual Testbed

There were three Arduinos, so the sensor nodes had to be mounted and put them in three different places. The location should be in free space.

The unique ID has to be introduced manually into the Arduino sketch with the Arduino IDE and the location into the python script by entering into the Arduino by secure shell as it was explained earlier, go to the folder where the “main.py” is, and modified the following line by using “nano”, a text editor:

self.address = ’Sagrada Familia, Carrer de Mallorca, Barcelona’

After a day of collecting data, the three logData files had been analyze, and some graphs to show the data had been made.

The figure 5.8 shows the nodes deployed, the location tries to be a little bit different to get distinct values.

Figure 5.5: Testbed Map.

5.2.1 Results

Thanks to the testbed a sensor network has been deployed, with only three nodes, for economical reasons, but it has been shown that the network could grow without problems.

A mobile application has been made, and shows an example that could led to more people to create other applications that work with the sensor data stored in Opencities. The data has been stored on a platform and the mobile application has

(a) Graphic Temperature

(b) Graphic Light

Figure 5.6: Testbed Results

been able to access them, in this case a problem happened, the application was not designed to that amount of data, so it needs improvement.

All the data produced during this testbed has been stored in the SD card of the Arduinos, and put them into graphics, as shown in the following figures: 5.6a, 5.6b, 5.7a, 5.7b, 5.8.

But the following graphs show only 100 minutes of the testbed to make it easier to read. The full graphs are in the appendix D.

Seeing the graphs has been inferred the following:

The temperature is clearly higher in Badalona, but the three locations are not far apart, so this value should not vary much. It is important to notice that this is a low quality sensor so it is possible that it may have failed during the testbed since the graphs of Montgat and Barcelona are similar between them, but very different in comparison to Badalona.

In the case of light, the sensor node in Barcelona receives direct sunlight during most of the day, but the node in Montgat and Badalona does not. Consequently, it is normal that the Barcelona node had a much higher light value. It is important to consider that this node is located in an office, so is reasonable to think that the sudden drop from 100 lux to 10 or less may be because someone turn off the light of the office, and that the sunlight was gone a long time ago.

The Barcelona node collects noise values much higher than the other two nodes. The Badalona and Montgat nodes are in areas with punctual car traffic, while the Barcelona node is in a busy area, causing more noise.

In the case of the relative humidity, the Badalona node is near a river and the Montgat node is near the sea, so it is logical that the Barcelona node collects less relative humidity.

(a) Graphic Noise

(b) Graphic Humidity

Figure 5.7: Testbed Results

The Badalona node captures better air quality than the Barcelona and Montgat nodes. The Barcelona node collects constant values, more or less, which is normal because there is in an area with high car traffic. The Badalona node collects a decrease in air quality as time passes, which coincides with the arrival of workers who park in the area. In the case of Montgat, there is a big difference in a time slot. This node is located near a highway, so it is possible that this time slot coincide when there is more traffic on the highway.

Figure 5.8: Graphic Air Quality.

Chapter 6

CONCLUSIONS

During this project a deployment of a sensor network had been made with optimal results, this has proved that the sensor nodes have function properly. Therefore, it has been shown that anyone can deploy its own network in an inexpensive way, with open source hardware and software, and easy to use. A mobile application has been developed to serve as an example, so the citizens who want to create their own find the process easier. All the code that it has been created during this project is open source and is online, in a Github repository.

In conclusion, the project had satisfied the goals presented at the start, which are share sensor data on an open network, and let the users visualize it.

Chapter 7

FUTURE WORK

This project can be improved in two ways, the sensor node and the Android application.

In the case of the sensor node it will be best if a prototype is build for the node to be in the outside, and, also to make the Arduino Power over Ethernet. On the other hand, the mobile application showed some issues, when the data is too big, the process takes a little bit, which could make the users stop using it. But there is also a line that could not be make in this project, and is to show how the data changes during a period of time.

Finally the project could be diffuse by a web page or conference on sensors.

Bibliography

[Aosong Electronics Co., 2013] Aosong Electronics Co., L. (2013). Digital-output relative humidity & temperature sensor/module dht22. https://cdn.shopify.com/s/files/1/0045/8932/files/DHT22.pdf?100745. [Online; accessed 10-February-2014].

[Arduino, 2014] Arduino (2014). Arduino. http://arduino.cc/. [Online; accessed 8-January-2014].

[Chong and Kumar, 2003] Chong, C.-Y. and Kumar, S. P. (2003). Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256.

[Emartee, 2013] Emartee (2013). Mini sound sensor - emartee.com. http://www.emartee.com/product/42148/Mini%20Sound%20Sensor%20%20Arduino%20Compatible. [Online; accessed 10-February-2014].

[Futurlec, 2013] Futurlec (2013). Mq-135 gas sensor. https://www.futurlec.com/Datasheet/Sensor/MQ-135.pdf. [Online; accessed 12-June-2014].

[Guide, 2014] Guide, A. (2014). Arduino guide. http://arduino.cc/en/Guide/ArduinoYun. [Online; accessed 8-January-2014].

[Instruments, 2013] Instruments, T. (2013). Lm35. www.ti.com/lit/ds/symlink/lm35.pdf. [Online; accessed 8-January-2014].

[Opencities, 2014] Opencities (2014). Opencities. http://opendata.nets.upf.edu. [Online; accessed March-2014].

[santander, 2014] santander, S. (2014). Smart city in santander. http://www.smartsantander.eu/. [Online; accessed 10/02/14].

[smart city, 2014] smart city, A. (2014). Smart city in amsterdam. http://amsterdamsmartcity.com/. [Online; accessed 07/02/14].

Appendix A

PILOT CHARTER

A.1 Pilot Charter

Fellow: Sergio Almendros Diaz
Mentor:
Advisor: Jaume Barcelo

A.1.1 Pilot purpose or justification

The purpose of this pilot is to build a sensor platform that can be attached to guifi nodes to gather and share sensory data.

A.1.2 Measurable pilot objectives and related success criteria

- Gather data about temperature, humidity, light, and noise.
- Share the data as open data.
- Deploy at least two nodes and gather data for at least two weeks.

A.1.3 High-level requirements

- Outdoor enclosure.
- Use open hardware and open software to the possible extent.
- Use standardized interfaces to integrate with other projects.

A.1.4 High-level pilot description

The goal is to use an arduino platform to create a bottom-up broadband wireless sensor networks. As guifi.net has already over 20,000 nodes, the idea is to co-locate the sensory platforms together with the guifi.net nodes and use the guifi.net network to transmit the data. This data should be gathered and shared. Ideally, the pilot should include a presentation interface for the users to visualize the data.

A.1.5 High-level risks

A possible risk is that the prototypes are not rugged enough for outdoor environments. It is also a risk that the prototype is not stable and needs to be reset very often.

A.1.6 Summary milestone schedule

- From 20/09/2013 to 23/09/2013
  - Establish the general idea of the TFG and specifics goals.
- From 23/09/2013 to 11/10/2013
  - Specify the tasks to do and make a planning.
- From 11/10/2013 to 30/10/2013
  - Connect first sensors to the Arduino.
- From 31/10/2013 to 10/01/2014
  - Connect to guifi network and upload data to an open data platform.
- From 10/01/2014 to 01/06/2014
  - Integration of sensors and communication aspects.
  - Install prototypes.
  - Data sharing and visualization.
  - Data analysis and evaluation of the testbed.
- From 02/06/2014 to 30/06/20014
  - Preparation of the final memory.
- From 01/07/2014 to the date of the presentation
  - Make the presentation.

A.1.7 Summary budget

The cost of this pilot will be approximately 4000 €. This quantity is for the scholarship to the student that will develop this pilot, budget for attending a conference or visiting collaborators, and the purchase of the necessary hardware.

Appendix B

PLANNING REPORT

B.1 Planning Report

The following sections explain the tasks that I will do in the course of this project.

B.1.1 Familiarization with the Arduino Yun

In this project I will be working with an arduino Yun, but I never worked before with any type of arduino, so the first task is to start coding different kind of programs. Then I will have to learn how to interact with the linux in the arduino Yun.

B.1.2 Preliminary testbed

I want to do an easy example to how to connect an arduino with a server running in my computer, what I want to do is establish a bridge between an arduino program and the linux within the arduino to be able to communicate with a server in my laptop, and send a string with the value returned by a sensor. This is a reduce problem of the real ”bottom-up sensor testbed” because, at the end, in every arduino will be a program that will have to send a message to a server with the data of the sensors attached to it.

B.1.3 Collect Data from sensors

First I will connect a temperature sensor to the arduino YUN, then, I will develop a program to collect the information from it, and send it to a server. When the temperature sensor works, I will do the same process with a humidity, light, and noise sensor.

B.1.4 Install Sentilo

Sentilo (www.sentilo.io) is an open source sensor and actuator platform that I will install in my laptop to act as the server between the sensor network and the interface for the users to visualize the data.

B.1.5 Communication with Sentilo

I will adapt the messages that the arduino send to fit with the Sentilo.

B.1.6 Real deployment

At this moment, the part of the arduino and the server will be done, so I will test the server installing the arduino in real nodes of the guifi network, for example, the node in the Universitat Pompeu Fabra, and any other node that allow me to install it. The arduino will have a temperature, humidity, light, and noise sensor.

B.1.7 Interface

I want to do an interface for any user to understand the meaning of the temperature, humidity, light, and noise values. This interface will be develop for an android mobile application.

B.1.8 Sentilo module

I will contribute to Sentilo and other sensor data brokerage platforms accommodating the sensor testbed deployed in the previous tasks.

B.1.9 Final report

This task have to be done in parallel with all the other ones, and its purpose is document all the work that I will do.

B.1.10 Gantt chart

Figure B.1: Gantt Chart

Appendix C

CLASS DIAGRAM

<<Java Class>>
MainActivity
com.upf.abottomupsensortestbed43
btnShowLocation: Button
MainActivity()
onCreate(Bundle):void
GET(String):String
convertInputStreamToString(InputStream):String
~dataBase
0..1
<<Java Class>>
HttpAsyncTask
com.upf.abottomupsensortestbed43
HttpAsyncTask()
doInBackground(String[]):String
onPostExecute(String):void
<<Java Class>>
DataBase
com.upf.abottomupsensortestbed43
state: String
map: GoogleMap
currentLocation: LatLng
textView: TextView
bTemp: Button
bHum: Button
bNoise: Button
blight: Button
bG: Button
bM: Button
DATASETID: String
APIKEY: String
featuresByCoordinates: HashMap<String,LinkedList<Feature>>
heatmaplistTemp: ArrayList<WeightedLatLng>
heatmaplistHum: ArrayList<WeightedLatLng>
heatmaplistNoise: ArrayList<WeightedLatLng>
heatmaplistlight: ArrayList<WeightedLatLng>
heatmaplistG: ArrayList<WeightedLatLng>
lfeaturesMarkersG: ArrayList<Feature>
dselected: GradientDrawable
dunselected: GradientDrawable
markerSelected: boolean
bSelected: String
getDATASETID():String
setDATASETID(String):void
getAPIKEY():String
setAPIKEY(String):void
DataBase()
getInstance():DataBase
getLfeatures():ArrayList<Feature>
setLfeatures(ArrayList<Feature>):void
getState():String
setState(String):void
toString():String
fillLists():void
addWeightedLatLng(LatLng,float,String):void
createFeaturesByCoordinates():void
addMarkers(String):void
addHeatMaps(String):void
~gps
0..1
<<Java Class>>
GPSTracker
com.upf.abottomupsensortestbed43
mContext: Context
isGPSEnabled: boolean
isNetworkEnabled: boolean
canGetLocation: boolean
location: Location
latitude: double
longitude: double
MIN_DISTANCE_CHANGE_FOR_UPDATES: long
MIN_TIME_BW_UPDATES: long
locationManager: LocationManager
GPSTracker(Context)
getLocation():Location
stopUsingGPS():void
getLatitude():double
getLongitude():double
canGetLocation():boolean
showSettingsAlert():void
onLocationChanged(Location):void
onProviderDisabled(String):void
onProviderEnabled(String):void
onStatusChanged(String,int,Bundle):void
onBind(Intent):IBinder
-mInstance
0..1
-lfeaturesMarkersH
-lfeaturesMarkersT
-lfeaturesMarkersN
-lfeaturesMarkersL
-Lfeatures

Figure C.1: Class Diagram of the Android App part 2.

Appendix D

TESTBED GRAPHICS

Figure D.1: Complete Graphic Air Quality.

Figure D.2: Complete Graphic Humidity.

Figure D.3: Complete Graphic Light.

Figure D.4: Complete Graphic Noise.

Figure D.5: Complete Graphic Temperature.