

SMART CAMPUS IN THE SMART CITY

A Degree Thesis

Submitted to the Faculty of the

**Escola Tècnica d'Enginyeria de Telecomunicació de
Barcelona**

Universitat Politècnica de Catalunya

by

Sònia Gudayol Marquès

In partial fulfilment

of the requirements for the degree in

SISTEMES DE TELECOMUNICACIÓ ENGINEERING

Advisor: Anna Calveras Auge

Barcelona, June 2016

Abstract

This project falls into the concept of IoT based Smart City and aims to carry out the deployment of a complete system in a University campus environment. Additionally, the development of a Global Connector has been evaluated, which would allow a generalized utilization of different platforms to carry this out in an agile and efficient manner. The scope of the project includes the Sensors, which generate the data; four different Platforms, which integrate the sensors and manage the data; a Server, which stores the data and performs data publishing and retrieving functions; a Connector, which is a software program that connects the platforms with the server; and the Client Application, which retrieves and utilizes the data. After evaluating different alternatives, the best option for the multi-platform Global Connector has been found to be a program that would allow the generation of code for the connector for any of these platforms.

Resum

Aquest projecte s'emmarca dins el concepte Smart City basat en IoT i té per objectiu dur a terme el desplegament d'un sistema complert en un entorn de campus universitari. Addicionalment, s'ha evaluat un Connector Global que permetés una utilització generalitzada de diferents plataformes per tal de dur a terme aquest tipus de funcions d'una manera àgil i eficient. L'àmbit del projecte inclou els Sensors, que generen dades; quatre Plataformes diferents, que integren els sensors i gestionen les dades; un Servidor, que emmagatzema les dades i realitza funcions de publicació i recuperació de dades; un Connector, que és un programa software que connecta les plataformes amb el servidor; i l'Aplicació Client, que recupera i fa ús de les dades. Després d'evaluar diferents alternatives s'ha vist que la millor opció pel Connector Global multi-plataforma era desenvolupar un programa que permetés generar el codi del connector per a qualsevol de les plataformes.

Resumen

Este proyecto se enmarca dentro el concepto Smart City basado en IoT y tiene como objetivo el despliegue de un sistema completo en un entorno de campus universitario. Adicionalmente, se ha evaluado un Conector Global que permita una utilización generalizada de distintas plataformas para llevar a cabo este tipo de funciones de manera ágil y eficiente. El ámbito del proyecto incluye los Sensores, que generan los datos; cuatro Plataformas distintas, que integran los sensores y gestionan los datos; un Servidor, que almacena los datos y realiza funciones de publicación y recuperación de datos; un Conector, programa software que conecta las plataformas con el servidor; y la Aplicación Cliente, que recupera y hace uso de los datos. Después de evaluar distintas alternativas se ha visto que la mejor opción para el Conector Global multi-plataforma era desarrollar un programa que permitís generar el código del conector para cualquiera de las plataformas.

Acknowledgements

First and foremost I would like to offer my sincerest gratitude to my advisor Anna Calveras Auge, first for conceding me the opportunity to do my thesis about a field that so deeply interests me; and, above all, for the ever constant guidance and counseling throughout this period.

I would also like to thank the students Jose Ignacio Mimbrero and Gabriel Espin for their help and for answering any question I had about their projects, even when they had their own busy schedules.

Revision history and approval record

Revision	Date	Purpose
0	30/05/2016	Document creation
1	20/06/2016	Document revision
2	27/06/2016	Document revision

Document distribution list

Name	e-mail
Sònia Gudayol Marquès	sonia.gudayol@alu-etsetb.upc.edu
Anna Calveras Auge	anna.calveras@entel.upc.edu

Written by:		Reviewed and approved by:	
Date	20/06/2016	Date	27/06/2016
Name	Sònia Gudayol Marquès	Name	Anna Calveras Auge
Position	Project Author	Position	Project Supervisor

Table of contents

Abstract	2
Resum.....	3
Resumen.....	4
Acknowledgements	5
Revision history and approval record	6
List of figures.....	9
List of tables.....	11
1. Introduction	12
1.1 Project objectives	12
1.2 Project requirements.....	12
1.3 Project specifications.....	12
1.4 Methodology.....	12
1.5 Work plan and Gantt diagram.....	13
1.6 Document structure	17
2. Smart Campus in Smart Cities	18
3. Sensors	20
3.1 Precision Light Sensor (phidget 1127)	21
3.2 Force Sensor (phidget 1106).....	21
3.3 Magnetic switch	22
3.4 Motion Sensor (phidget 1111).....	23
4. Provider platforms.....	24
4.1 Zolertia Z1	24
4.1.1 Z1 networks	24
4.1.2 Z1 Sensor connection	25
4.2 Arduino.....	27
4.2.1 Arduino sensor connection.....	28
4.3 Intel Edison.....	30
4.3.1 Intel Edison sensor connection.....	31
4.4 Raspberry Pi	33
4.4.1 Raspberry Pi sensor connection.....	34
5. Sentilo server.....	36

5.1 Sentilo resources and services.....	37
5.2 Sentilo interaction through HTTP REST API.....	38
5.3 Sentilo Catalog Web.....	40
6. Global connector	42
6.1 Programming language compatibility	43
6.2 Individual connector flowchart and structure	44
6.3 Global connector.....	49
6.4 Individual platform specifications	50
6.4.1 Arduino specifications	51
6.4.2 Intel Edison specifications	54
6.4.3 Raspberry Pi specifications	55
6.4.4 Zolertia Z1 specifications.....	56
7. Client application	60
7.1 HTTP requests and responses	61
7.2 Connection with Java class	63
7.3 Web application with JSP.....	66
8. Integration.....	68
9. Budget.....	79
10. Conclusions.....	80
11. Future lines of work.....	81
Bibliography	82
Appendices.....	84
I Sentilo	84
II Z1: Contiki OS 2.7	88
III Z1: COAP elements	90
IV Raspberry Pi.....	94
V Intel Edison and Arduino.....	96
Glossary	97

List of figures

Figure 1: Smart Campus architecture	19
Figure 2: Smart Campus architecture and network.....	20
Figure 3: Precision light sensor (from phidgets.com).....	21
Figure 4: Force sensor (from phidgets.com).....	22
Figure 5: Magnetic switch (from seeedstudio.com).....	22
Figure 6: Motion sensor (from phidgets.com).....	23
Figure 7: Zolertia Z1 module (from zolertia.sourceforge.net).....	24
Figure 8: Z1 network.....	25
Figure 9: Zolertia Z1 pins	26
Figure 10: Zolertia Z1 schematic.....	26
Figure 11: Arduino and Geniuno UNO (from arduino.cc).....	27
Figure 12: Arduino pins (from arduino.cc).....	29
Figure 13: Arduino schematic	29
Figure 14: Arduino and Ethernet Shield schematic	30
Figure 15: Intel Edison module (from intel.com).....	30
Figure 16: Intel Edison Arduino pins (from intel.com)	32
Figure 17: Intel Edison schematic	32
Figure 18: Raspberry Pi 2 Model B (from raspberrypi.org)	33
Figure 19: MCP3002 schematic (from microchip.com).....	34
Figure 20: Raspberry Pi pins (from dev.px4.io)	35
Figure 21: Raspberry Pi connection schematic.....	35
Figure 22: Sentilo architecture (from sentilo.io).....	37
Figure 23: Sentilo Catalog Web's Universal viewer.....	40
Figure 24: Sentilo Catalog Web view of sensor observation	42
Figure 25: Previous project scope	44
Figure 26: Current project scope	44
Figure 27: Connector main flowchart	45
Figure 28: Sensor reading flowchart.....	46
Figure 29: Prepare data flowchart	47
Figure 30: Send to Sentilo flowchart	48
Figure 31: Global connector flowchart.....	49
Figure 32: Arduino IDE	51
Figure 33: Arduino sketch layout.....	52
Figure 34: Z1 Connector MainApp GUI.....	60

Figure 35: Client application design	61
Figure 36: Application permissions on Sentilo Catalog Web	62
Figure 37: getData flowchart	63
Figure 38: getObservation flowchart	64
Figure 39: JSON reading order	65
Figure 40: Integration of the complete syste	68
Figure 41: Arduino integration with a magnetic switch sensor	69
Figure 42: Arduino connector integration	69
Figure 43: Arduino integration results as shown on SCW	70
Figure 44: Intel Edison integration with a sensor	70
Figure 45: Intel Edison connector integration	71
Figure 46: Intel Edison integration results as shown on SCW	71
Figure 47: Raspberry Pi integration with a sensor	72
Figure 48: Raspberry Pi connector integration	72
Figure 49: Raspberry Pi integration results as shown on SCW	73
Figure 50: Console view of the Global Connector	73
Figure 51: Results of the Global Connector integration	74
Figure 52: Integration of Zolertia Z1 nodes	74
Figure 53: Border Router running log	75
Figure 54: Connector's MainApp GUI	75
Figure 55: Raspberry Pi integration results as shown on SCW	76
Figure 56: Sentilo server running log	76
Figure 57: Web app main menu	77
Figure 58: Web app spaces view	77
Figure 59: Web app floorplan with results	78
Figure 60: Web app results graph	78

List of tables

Table 1: Precision light sensor specifications.....	21
Table 2: Force sensor specifications.....	22
Table 3: Magnetic switch specifications.....	23
Table 4: Motion sensor specifications.....	23
Table 5: Arduino boards categorized by purpose.....	27
Table 6: Ethernet chips and corresponding libraries.....	28
Table 7: Raspberry Pi, MCP3002 and sensor pin connection	36
Table 8: PUT method for Sentilo	39
Table 9: DELETE method for Sentilo.....	39
Table 10: GET method for Sentilo.....	39
Table 11: Sentilo response codes.....	40
Table 12: Programming compatibility accross platforms	43
Table 13: HTTP message with observation to send.....	48
Table 14: Sensor class functions	57
Table 15: Mote class functions.....	58
Table 16: SensorNetwork class functions	59
Table 17: Client application HTTP request.....	61
Table 18: Budget of materials	79
Table 19: Budget of work.....	80
Table 20: Budget total	80

1. Introduction

1.1 Project objectives

This project aims to apply the Smart City ideal, and particularly the IoT approach, to a more limited environment of a University campus. Following this, the three fundamental parts of a Smart City project's usual architecture will be implemented; 1) providers, which are sensors that are connected to a platform, 2) server, which stores the sensor data and handles data publishing and retrieving, and 3) an application that utilizes the sensor data to offer a service. In this project the platforms Arduino, Intel Edison, Raspberry Pi and Z1 will be studied. While the application will be designed to interact with the server, the provider platforms do not originally provide this service. A connector program will be created for this purpose. This project will also aim to create one global connector that can be run on any of these platforms. And, in order to facilitate the continuity of this initiative, all necessary documentation has been provided for all platforms and connectors. Finally, this project will also provide documentation for all platforms.

1.2 Project requirements

1. Sensors will provide the data that will be published on the server through the platform.
2. Platforms will run the connector program to send the data to the server.
3. The server that will be implemented will be Sentilo.
4. The application will retrieve the data from the server.
5. The application will be able to be accessed through most devices like computers, phones, and tablets, independently of their operating system.
6. The documentation will cover the description of the different elements in the three layers, as well as provide a guide for their set up and launching.

1.3 Project specifications

- The platforms that will be studied and implemented are Arduino, Intel Edison, Raspberry Pi and Zolertia Z1.
- The server will be based on the Sentilo sensor and actuator platform.
- Retrieving and publishing of data will be done with HTTP requests.

1.4 Methodology

This work takes on the work from Jose Ignacio Mimbrero Catalán's Master Thesis 'Contiki applications for Z1 motes for 6LowPAN' and Gabriel Espín Sanchez's Degree Thesis 'Plataforma de sensores móviles, para medir la contaminación en un entorno urbano'. The first studied the

Zolertia Z1 platform, and the former studied the Intel Edison platform. This project takes specifically on their platform set up, Sentilo server, and connector program to Sentilo.

The research methods that will be used on this project will be documentary analysis for those sections of the project that have been previously implemented and documented, such as the server and the Intel Edison and Z1 platforms. And for those that have not been previously implemented, the research method will be a mix of documentary analysis and observational research.

1.5 Work plan and Gantt diagram

Zolertia Z1 on PC	WP #1	
Major constituent:	Sheet 1 of 9	
Short description: Implementation of the Z1 platform and network on a Linux computer.	Planned start date: 08/02/2016	
	Planned end date: 07/03/2016	
	Start event: 08/02/2016 End event: 14/04/2016	
Internal task 1: Installing the environment on the Linux computer. This includes the OS for the Z1 platform, Contiki, and an IDE to run the connector.	Deliverables:	Dates:
Internal task 2: Testing the sensors.		
Internal task 3: Publishing data on Sentilo and visualizing it on the Sentilo Catalog Web		

Zolertia Z1 on Raspberry Pi	WP #2	
Major constituent:	Sheet 2 of 9	
Short description: Implementation of the Z1 platform and network on a Raspberry Pi.	Planned start date: 15/02/2016	
	Planned end date: 21/03/2016	
	Start event: 07/03/2016 End event: 04/05/2016	
Internal task 1: Installing the environment on the Raspberry Pi. This includes the OS for the Z1 platform, Contiki.	Deliverables:	Dates:

Internal task 2: Testing the Z1 and the connector.		
Internal task 3: Publishing data on Sentilo and visualizing it on the Sentilo Catalog Web		

Integration of Sentilo server	WP #3	
Major constituent:	Sheet 3 of 9	
Short description: Setting up the Sentilo server on the lab computer.	Planned start date: 22/03/2016	
	Planned end date: 01/04/2016	
	Start event: 04/04/2016 End event: 17/04/2016	
Internal task 1: Setting up the Sentilo server on the computer.	Deliverables:	Dates:
Internal task 2: Testing the Sentilo server and the Sentilo Catalog Web.		

Intel Edison as Sentilo provider	WP #4	
Major constituent:	Sheet 4 of 9	
Short description: Implementation of the Intel Edison as provider to Sentilo.	Planned start date: 14/03/2016	
	Planned end date: 28/03/2016	
	Start event: 18/04/2016 End event: 02/05/2016	
Internal task 1: Setting up the Intel Edison platform.	Deliverables: Connector source code	Dates: 5/06/2016
Internal task 2: Hardware integration with the platform and the sensors.		
Internal task 3: Testing the connector to Sentilo.		
Internal task 4: Modification of the connector to fit the global connector common structure.		

Raspberry Pi as Sentilo provider	WP #5	
Major constituent:	Sheet 5 of 9	
Short description: Study and development of the Raspberry Pi as provider to Sentilo.	Planned start date: 21/03/2016 Planned end date: 18/04/2016	
	Start event: 1/05/2016 End event: 4/06/2016	
Internal task 1: Study of the different ways to access the Raspberry Pi's GPIO data, in particular for analog sensors.	Deliverables: Connector source code	Dates: 12/06/2016
Internal task 2: Hardware integration with the platform and the sensors.		
Internal task 3: Development of the Raspberry Pi connector.		
Internal task 4: Testing the connector to Sentilo.		

Arduino as Sentilo provider	WP #6	
Major constituent:	Sheet 6 of 9	
Short description: Study and development of the Arduino as provider to Sentilo.	Planned start date: 04/04/2016 Planned end date: 18/04/2016	
	Start event: 09/05/2016 End event: 4/06/2016	
Internal task 1: Study of the different ways to access the Arduino pins.	Deliverables: Connector source code	Dates: 19/06/2016
Internal task 2: Hardware integration with the platform and the sensors.		
Internal task 3: Development of the Arduino connector.		
Internal task 4: Testing the connector to Sentilo.		

Global connector	WP #7	
Major constituent:	Sheet 7 of 9	
Short description: Development of a global connector.	Planned start date: 04/04/2016 Planned end date: 18/04/2016	
	Start event: 23/05/2016 End event: 13/04/2016	
Internal task 1: Study the programming compatibility between the platforms.	Deliverables: Connector source code	Dates: 19/06/2016
Internal task 2: Development of the global connector.		
Internal task 3: Testing of the global connector.		

Web application as client	WP #8	
Major constituent:	Sheet 8 of 9	
Short description: Development of a client application.	Planned start date: 21/03/2016 Planned end date: 31/05/2016	
	Start event: 01/06/2016 End event: 20/06/2016	
Internal task 1: Study the best way to handle the requests for the application.	Deliverables: Client application source code	Dates: 20/06/2016
Internal task 2: Development of the web application.		
Internal task 3: Deployment and testing of web application.		

Project documentation and thesis defense	WP #9	
Major constituent:	Sheet 9 of 9	
Short description: Development of a client application.	Planned start date: 15/02/2016 Planned end date: 11/07/2016	
	Start event: 01/05/2016 End event: 11/07/2016	
Internal task 1: Document all the devices' functionality.	Deliverables:	Dates:

Internal task 2: Document their set up or other additional requirements for the devices to work with the system.	Documentation	20/06/2016
Internal task 3: Preparing of the thesis defense.		

	February			March				April				May				June				July							
	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23				
WP1 Z1 PC																											
WP2 Z1 RP																											
WP3 Sentilo																											
WP4 IE																											
WP5 RP																											
WP6 AR																											
WP7 GC																											
WP8 App																											
WP9 Doc																											

There has been a deviation from the original work plan. Firstly, there was a delay in the first two WP due to missing documentation of the Z1 platform. And lastly, the Global connector was modified to include the study of the platforms individually.

1.6 Document structure

This Final Report will start with the motivation and description of the system. It will be followed by the technologies of the sensors, the platforms and the server. After that will go the development of the connector and the development of the client application. Finalizing this, there will be the sections of integration, budget, conclusions and future work. Additionally, the appendices will include what is left of the documentation.

2. Smart Campus in Smart Cities

Smart Cities is a term that has been widely used in the last decade, when in reality it has been present for much longer, as it describes the wish to change the environment we live in, to adapt to the changing need of its citizens. Nowadays, Smart Cities follow the ideal of intelligent, efficient and sustainable cities to improve the quality of life. There is much that can be improved, or monitored in order to better its performance.

Although the first instances of Smart Cities did not arise until 2005, in the 70s some cities were already using computer databases and cluster analysis to gather data and produce reports to study neighborhood demographics and housing quality, and to make a better distribution of resources to reduce poverty. In 2005, Cisco began working on the Connected Urban Development programme, a project to make cities more sustainable through the use of the company's know-how. Its pilot projects included working in cities like San Francisco, Amsterdam and Seoul. These progresses from Cisco were followed by IBM, who also wanted to make cities smarter through IT with its Smarter Planet initiative. But while Cisco focused on the improvement of new and old cities, IBM focused more on information management and analytics. These two companies sparked the imagination of Smart Cities worldwide.

In some cases the idea of Smart Cities has permeated into the city ideals, allowing multiple projects to be tested and deployed. Some of these cities are Amsterdam, Barcelona, and Stockholm. These are the cases where Smart City projects have been implemented on already established cities. However, at the other end of the spectrum are the cities that have been built from scratch with that ideal. Songdo city, South Korea, is one of the most ambitious and expensive projects of building a Smart City. It promotes the idea of a greener city, with green zones, numerous bicycle paths, and its characteristic underground waste management system. However, due to the living costs being quite high and due to a not optimal transportation connection to the capital, Seoul, this project has not been accepted by citizens as expected, and has a low percentage of occupation, both in residence and commercial space.

Milton Keynes, however, is a new city in England that has committed to the idea of Smart Cities. As a new city, it not only faces the challenges of thinking the Smart City from scratch, but it also has to deal with other challenges like its rapidly growing urbanization. What is also interesting is that they follow the top-down approach as well as the bottom-up approach. The first refers to city councils and designed companies making the decisions on the projects. The former involves regular citizens in identifying the needs, creating the projects and implementing them. [1] [2] [3]

Most of the projects seem to meet on the same concept, the Internet of Things. The IoT follows the idea of a network composed by everyday objects which will have Internet connectivity to

send and retrieve data. This often requires sensors, a platform they may be integrated on, and a program to send, receive and interpret data.

This project aims to apply the idea of Smart Cities in a more limited environment, the university campus. The purpose is to exhaustively study the opportunities that Smart Cities offer for improvement in this reduced space, so that this system may be scalable to larger areas while maintaining its efficiency.

This particular ideal of Smart Campus makes its approach using a variety of sensors in a wide range of spaces. The information generated by the sensors may be used as an alternative source to revise the energetic efficiency on the spaces they are deployed on, by monitoring levels like temperature, light, and presence detection.

These sensors can connect to the Internet and send their observations to a server by using a platform. There are many sensors that can be set up in many sensors, but this also results in many connectors (applications that connect the platform device to an Internet server to store the observations). This creates the need for there to be a standardized or global connector, able to serve many platforms indistinctly.

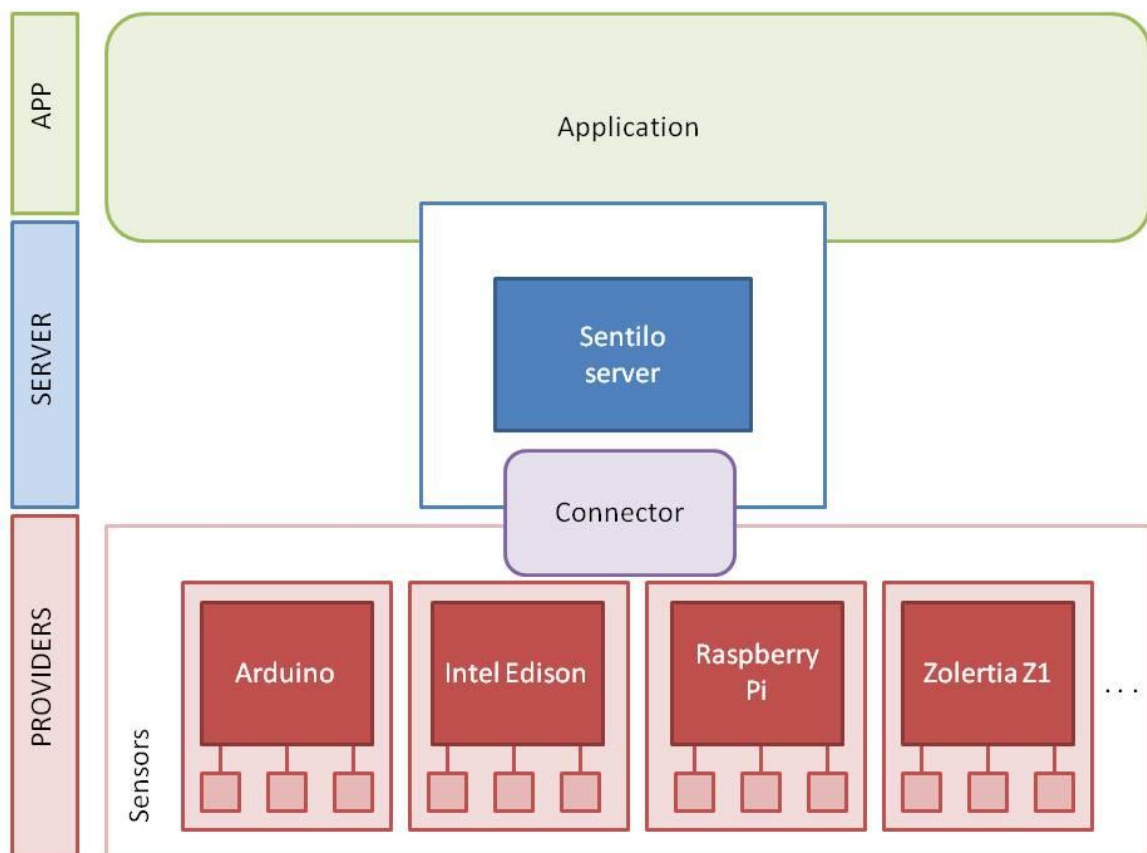


Figure 1: Smart Campus architecture

In this project, the server that will be used will be based on Sentilo, which is an open source sensor platform. The providers, constituted by the sensors and their platforms, will send information to the Sentilo server through the Connector. The platforms that this project will cover are Arduino, Intel Edison, Raspberry Pi and Zolertia Z1.

The application, which is the other end of the system, will extract the information from the server and visualize it in order to monitor the data. The application will perform the connection to Sentilo server by itself, so there will be no need for another connector agent.

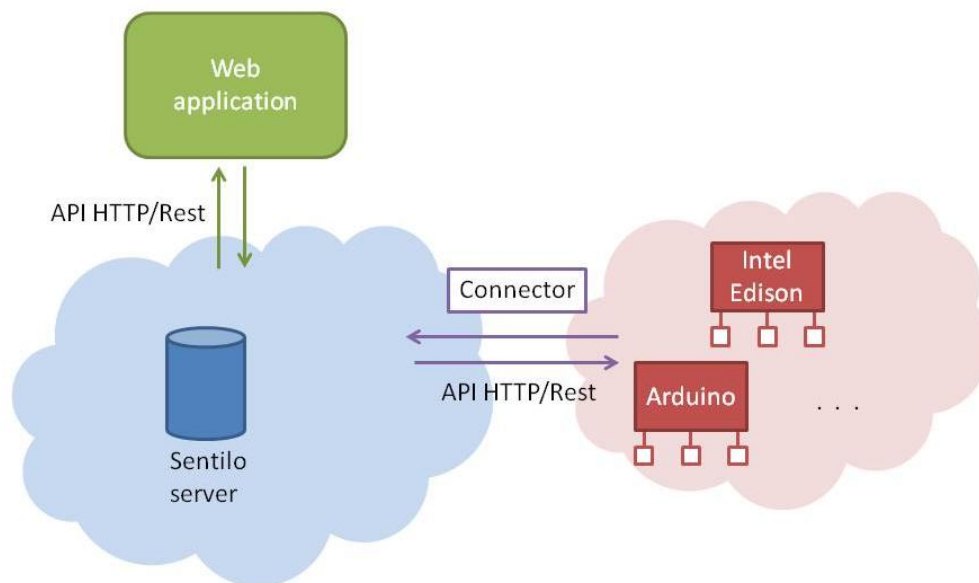


Figure 2: Smart Campus architecture and network

3. Sensors

Sensors are hardware modules that can produce data based on observations. All sensors can be categorized in two categories, depending on the type of data they can produce.

Analog sensors: the resulting observations are an integer value. Most platforms have an analog input range of 10-bit, so the values will go from 0 to 1023. Some platforms have a range of 12-bit, with values from 0 to 4095.

Digital sensors: the resulting observations may be '1' (high) or '0' (low).

This project will work with a number of analog and digital sensors.

3.1 Precision Light Sensor (phidget 1127)

The Precision Light Sensor is an analog sensor. It measures the light level in lux, and it can measure from 1 lux to 1000 lux.



Figure 3: Precision light sensor (from phidgets.com)

Specifications

Voltage supply	Minimum light level	Maximum light level
3.3V	1 lux	660 lux
5V	1 lux	1000 lux

Table 1: Precision light sensor specifications

The platforms this sensor will be connected to can read input data in 10-bit, so the resulting input from this sensor will have to be modified.

If the supply is 3.3V

$$light(lux) = \frac{light(bit)}{1024} \times 660$$

If the supply is 5V

$$light(lux) = \frac{light(bit)}{1024} \times 1000$$

3.2 Force Sensor (phidget 1106)

The Fore Sensor is an analog sensor. It measures the force applied to the button from 0 to 1000, but it is not intended to be a precision sensor nor a weight measurement device.

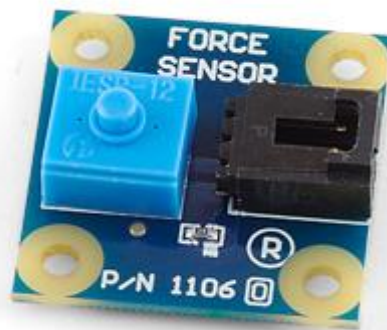


Figure 4: Force sensor (from phidgets.com)

Specifications

Voltage supply	Minimum force level	Maximum force level
5V	1 N	39.2 N

Table 2: Force sensor specifications

Following this data range, the resulting data from the sensor will have to be modified.

$$Force(N) = \frac{Force(bit)}{1024} \times 39.2$$

3.3 Magnetic switch

The Magnetic Switch sensor is a sensor that detects the close presence of a magnet.



Figure 5: Magnetic switch (from seeedstudio.com)

Specifications

Voltage supply	Minimum level	Maximum level
4.75V (minimum)	LOW	HIGH
5.25V (maximum)	LOW	HIGH

Table 3: Magnetic switch specifications

3.4 Motion Sensor (phidget 1111)

The Motion Sensor (or presence detector) detects variations changes in the infrared radiation, effectively noting the movement of a body of differing temperature from the rest of the room.



Figure 6: Motion sensor (from phidgets.com)

Specifications

Voltage supply	Minimum level	Maximum level
4.8V (minimum)	LOW	HIGH
5.8V (maximum)	LOW	HIGH

Table 4: Motion sensor specifications

Some platforms have already integrated sensors. Those will be explained along with their platform in the next section.

4. Provider platforms

The sensors that have been described cannot connect to the Internet alone. Therefore, they have to be integrated on a platform. In this system, a set of platform and connected or integrated sensors is a provider platform.

4.1 Zolertia Z1

The Z1 is a platform module aimed at the conformation of low power IoT wireless sensor networks. All modules have a MSP430(F2617) microcontroller, a microUSB connection for the power supply and the communication with the computer, and can be connected to the Internet.



Figure 7: Zolertia Z1 module (from zolertia.sourceforge.net)

The Z1's operating system is Contiki (another option is TinyOS), which is an open source OS aimed at IoT projects, and it enables the connection of the modules to the Internet. It supports the standard IPv4 and IPv6, and also 6lowpan, RPL, COAP. All Contiki applications are programmed in C.

4.1.1 Z1 networks

The Z1 modules can be connected to the Internet with their built-in antenna. Networks based on these modules have one Border Router, and one or more Servers. All modules are the same, and they can be programmed as the different elements of the network.

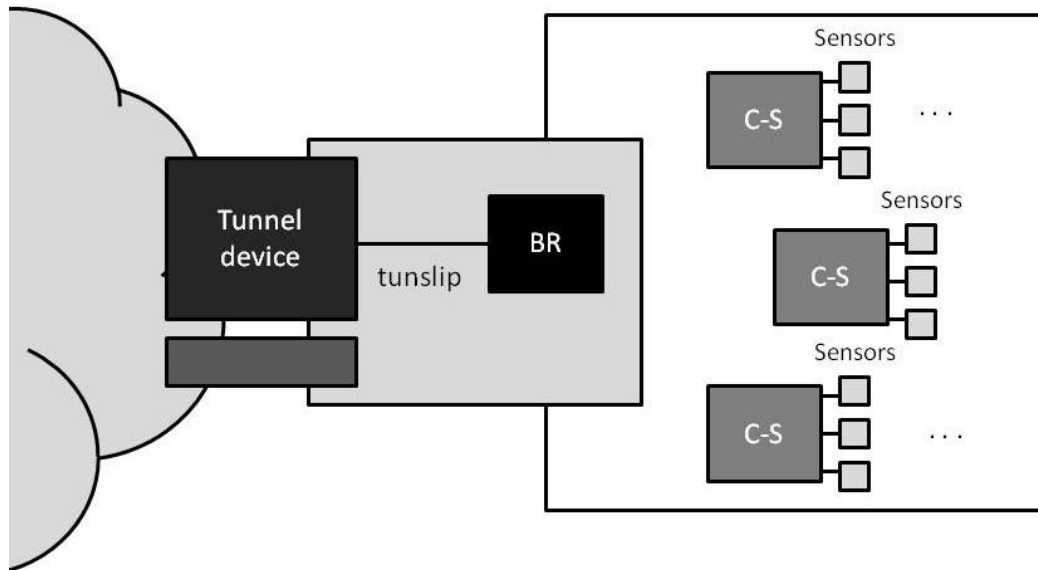


Figure 8: Z1 network

Tunnel device: may be any host computer and, additionally, Raspberry Pi, as explained further in the Integration section. The only requirements of the tunnel device are that it has to run on Linux, have Internet connection, and a USB port.

Tunslip: creates a bridge between the tunnel device and the Z1 node acting as Border Router for IP traffic over serial line. This is through a virtual network interface on the tunnel device, which also uses Serial Line Internet Protocol to encapsulate and enable the IP traffic.

Border Router: the Border Router Z1 node has to be connected to the host computer through USB. The Border Router communicates with the Servers using the COAP protocol, and connects to the tunnel device using Tunslip.

Server nodes: these nodes are the ones that contain the sensors, and they communicate with the Border Router. They don't necessarily have to be powered by the tunnel device, but they have to be in the range of the Border Router node.

4.1.2 Z1 Sensor connection

All modules have two integrated sensors; a temperature sensor and an accelerometer. Additionally, up to 4 Phidget sensors can be connected on a module, and their board can be expanded to include more digital and analog pins.

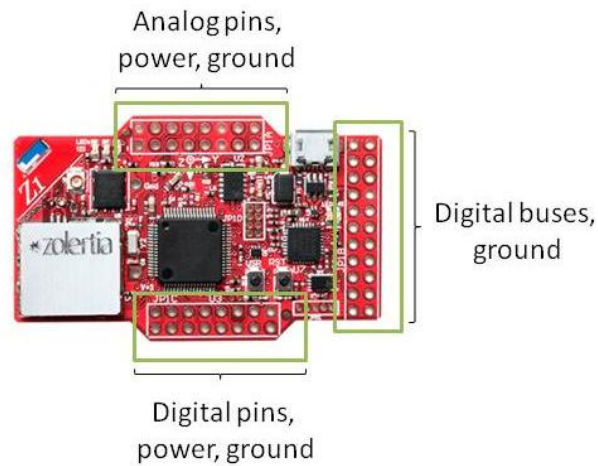


Figure 9: Zolertia Z1 pins

The digital buses may be: I2C, SPI, UARTs, USB.

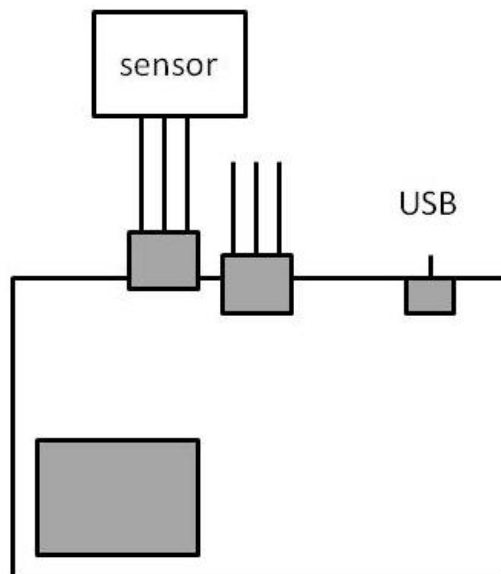


Figure 10: Zolertia Z1 schematic

All modules have 2 phidget ports by default, so these sensors may be connected directly without the requirement of additional pin connection.

4.2 Arduino

Arduino is a single-board microcontroller. It is aimed at the building of digital devices and interactive systems that, with their layout of digital and analog pins, can sense and control physical devices.

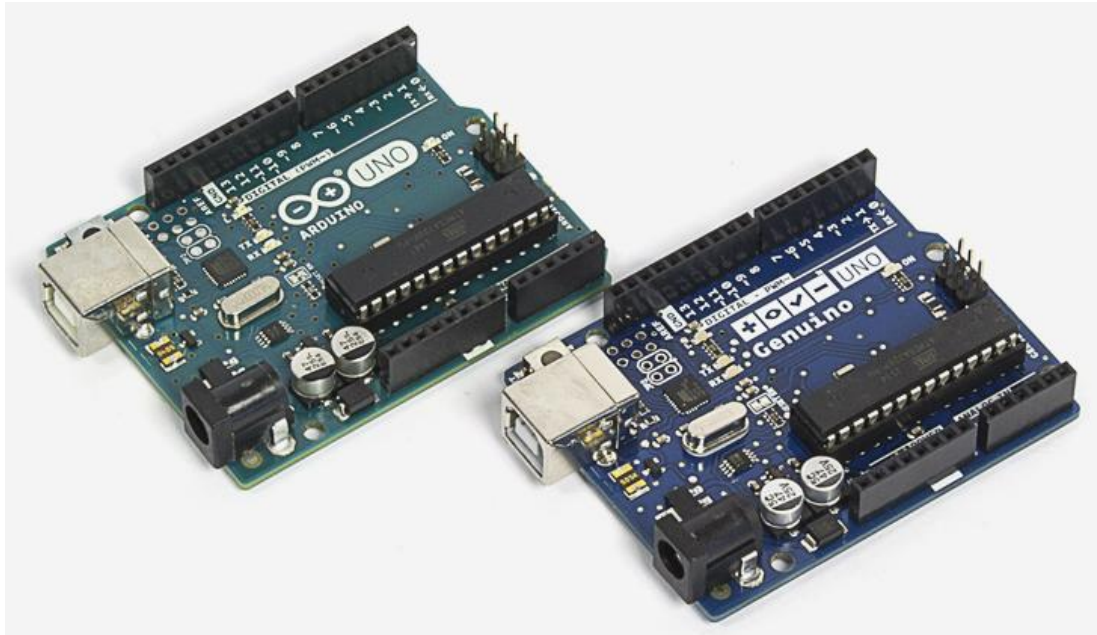


Figure 11: Arduino and Geniuno UNO (from arduino.cc)

There are, however, a number of Arduino boards, and they can be categorized depending on their purpose.

Basic boards	Enhanced boards	IoT oriented boards
Arduino Uno	Arduino Mega	Arduino Yun
Arduino 101	Arduino Zero	
Arduino Pro	Arduino Due	

Table 5: Arduino boards categorized by purpose

The names are pertaining to the products released in the USA, as the ones released on the rest of the world are called Geniuno.

The Arduino Uno is the most basic and commonly used of the Arduino platforms, and it is the best option to get started with the platform. The Arduino 101, in comparison, has lower power microcontroller, Bluetooth capabilities and an accelerometer and gyroscope [4]. The Arduino Pro's do not vary greatly from the previous two boards, but it has the characteristic that it is aimed at more permanent installations.

The Arduino MEGA is oriented at more complex projects, like robotics, and it has more digital and analog pins, and greater memory to run sketches with. The Arduino Zero and Due are both 32-bit and aimed at larger and more complex projects. And the Arduino Yun is oriented at IoT projects as it comes with Wifi and Ethernet built-in capabilities.

Unless those that have been specified like the Arduino Yun, none of the boards have connectivity to the Internet. There are, however, physical board add-ons called ‘Shields’ that provide this feature. Specifically, there are the Arduino Ethernet Shield and the Arduino Wifi Shield. The Arduino comes with the libraries “Ethernet” and “Wifi” respectively to enable their functionality.

There are many Ethernet shields on the market other than the official one, but the Ethernet chip has to be taken into account, as it may need different libraries.

	Ethernet	Ethernet2	EtherCard	UIPEthernet
W5500		x		
W5200	x			
W5100	x			
ENC28J60			x	x

Table 6: Ethernet chips and corresponding libraries

The Ethernet2 is the discontinued library that Arduino Ethernet Shield 2 used, and its interface and functions is practically the same as the Ethernet one. This Ethernet library is the one that Arduino Ethernet Shield 3 uses. However, shields by other manufacturers that have the same chip will also be able to use the official library. Additionally, shields that have ENC28J60 chips or similar are supported by the EtherCard external library, which has a different function list. That is why there is the alternative UIPEthernet external library, which works with ENC chips while keeping the same function layout as the official libraries.

The shields are mounted on top of the Arduino board, and some of them offer PoE (power over Ethernet), so that only one connection is needed in case of installation.

This project will be using Arduino Uno R3 coupled with an Ethernet Shield with the W5100 Ethernet chip.

4.2.1 Arduino sensor connection

Arduino was developed with these kind of projects in mind. Digital and analog sensors can be directly connected to the digital and analog pins on the board.

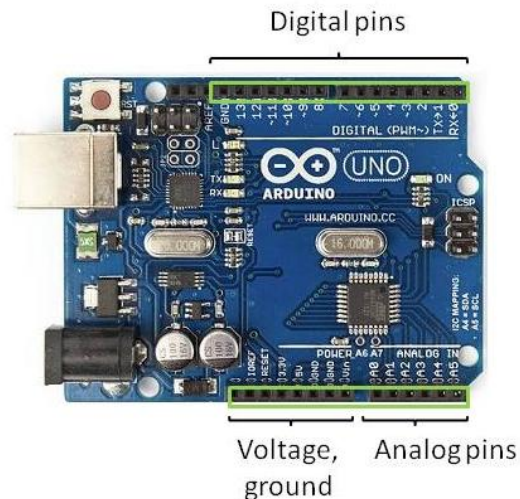


Figure 12: Arduino pins (from arduino.cc)

The Arduino Uno has 6 analog pins (A0-A5) and 14 digital pins (0-13). All analog pins are strictly input pins, and digital pins are, by default, also input pins [5]. They may be configured as output (or specify as input) while programming the Arduino.

The system connection of a single analog sensor follows the schematic.

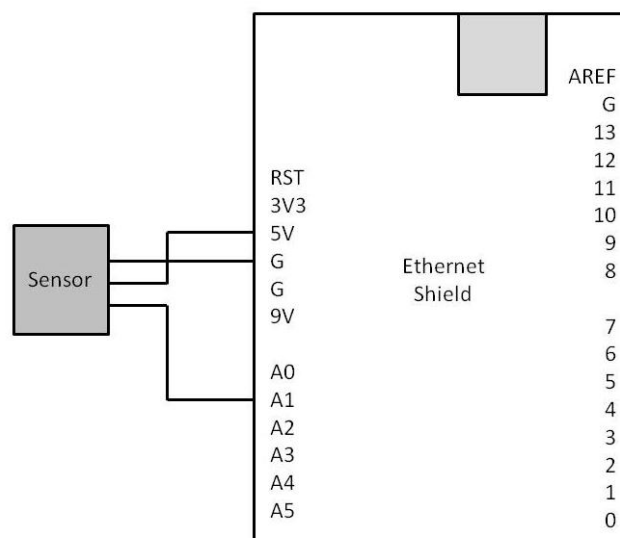


Figure 13: Arduino schematic

Since the Ethernet Shield sits atop the Arduino shield.

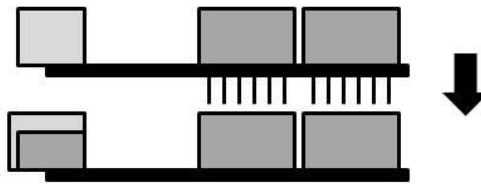


Figure 14: Arduino and Ethernet Shield schematic

The connection for a digital sensor is exactly the same but for the connection of the data, since it would go to one of the digital pins on the right.

4.3 Intel Edison

Intel Edison is a computer-on-module whose dual-core CPU and single core micro-controller support complex data collection that does not need a lot of power. By having these two main features, it is aimed at IoT projects.

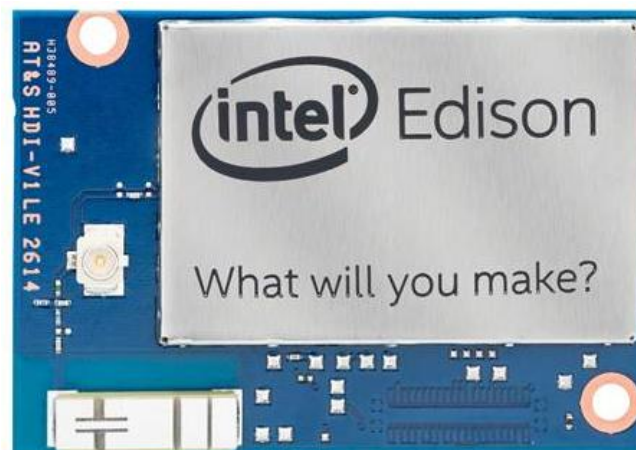


Figure 15: Intel Edison module (from intel.com)

There are a few types of Intel Edison platforms, mainly:

Intel Edison Compute Module: the single microcontroller module, it has all the capabilities and it may be paired with an expansion board to offer GPIO interfaces.

Intel Edison Kit for Arduino: the compute module completed with Arduino Uno R3 expansion board. It provides the ability to program the board with Arduino IDE.

Intel Edison Kit with Breakout Board: the compute module with a small breakout board.

In this project, Intel Edison Kit for Arduino will be used. Any further instances of Intel Edison will refer to this particular platform.

Additionally, Intel Edison has Wi-Fi and Bluetooth capabilities. However it has to be noted that for proper communication between the board and the computer when programming, it is required that both be connected to the same Wi-Fi network.

The board runs Linux on default, and it can be accessed as a regular Linux when Intel Edison is connected to a computer through serial communication. To create and upload programs on the board, there are a number of IDEs.

Intel XDK IoT Edition: an IDE that is programmed with Node.js, a runtime environment that uses JavaScript. It has built-in libraries (in this case, called modules) that allow the user to work with sensors easily. Furthermore, it can be used for other web services and has cloud integration.

Arduino: programming environment using Processing, a language based in C++. It can read and write on the I/O pins very easily, and it comes with a variety of program examples to try all the features.

Intel System Studio IoT Edition: plug-in for Eclipse IDE, may be programmed with C/C++ and Java. It comes with libraries to easily work with sensors.

Python: in this case it is not an IDE, but the board can be programmed in Python.

This project will use the Eclipse IoT, programming in C/C++.

4.3.1 Intel Edison sensor connection

As the Intel Edison Kit for Arduino will be used, any analog and digital pins can be directly connected to the analog and digital pins on the expansion board.

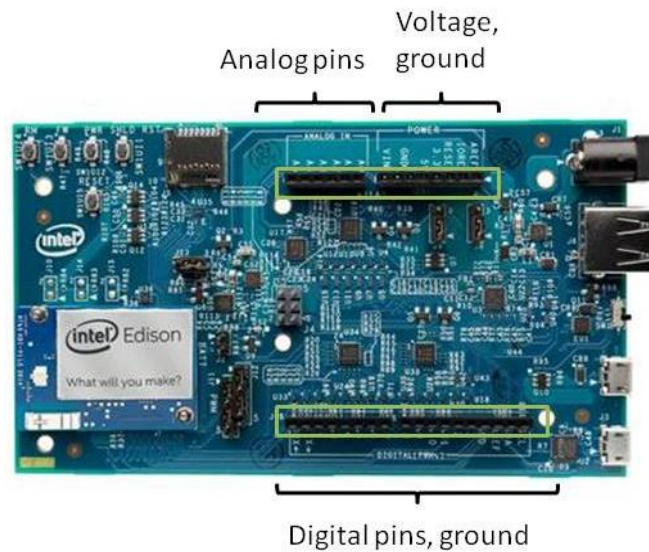


Figure 16: Intel Edison Arduino pins (from intel.com)

The Intel Edison board has 6 analog pins (A0-A5) and 14 digital pins (0-13). All analog pins are strictly input pins, and digital pins may be configured as input or output pins.

The system connection of a single analog pin follows the schematic.

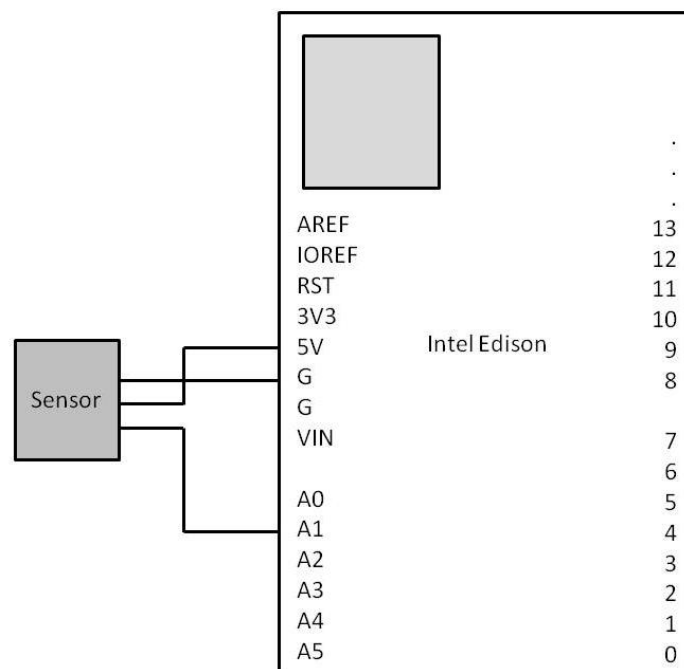


Figure 17: Intel Edison schematic

The connection for a digital sensor is exactly the same but for the connection of the data, since it would go to one of the digital pins on the right.

4.4 Raspberry Pi

The Raspberry Pi is a single-board computer that is developed with the purpose of facilitating the teaching of computer science in schools and developing countries. This becomes evident with its low price, low power need and the ability to connect a monitor through HDMI, and keyboard and mouse with USB. In addition, it can connect to Internet through Ethernet cable (the Raspberry Pi 3 has WiFi capabilities).



Figure 18: Raspberry Pi 2 Model B (from raspberrypi.org)

There are three models of Raspberry Pi (three generations). This project will be centered on the use of the Raspberry Pi 2 (which follows the 'Model B' layout). Further detailing of the device will be about the Raspberry Pi 2.

The CPU is a 900MHz quad-core ARM Cortex-A7, and has 1GB of RAM. It has 4 USB ports, 40 GPIO pins (all pins are digital) [6]. The operating system may vary, and it has to be installed on a microSD card and later inserted on the board.

The best way to install an operating system is through the NOOBS installer. This installer lets the user decide which operating system to install. Some of the available operating systems are:

- Raspbian
- Ubuntu Mate
- Windows 10 IOT Core

The one that has been used for this project is Ubuntu Mate.

4.4.1 Raspberry Pi sensor connection

Analog sensors

As stated above, all the pins from the Raspberry Pi are digital, so analog sensors cannot be connected directly. In this case, an Analog-Digital Converter (ADC) is needed. Some basic modules that may work on the platform are:

- MCP3002: 10-bit data, 2 inputs [7].
- MCP3008: 10-bit data, 8 inputs.
- Gertboard: GPIO expansion board with A/D and D/A converters, and an Atmel ATmega328 AVR microcontroller [8]. It is plugged directly onto the pin lines of the Raspberry Pi, and the microcontroller allows the running of external programs on the board, as well as programming with Arduino IDE.

The one that is used in this project is the MCP3002, with the schematic:

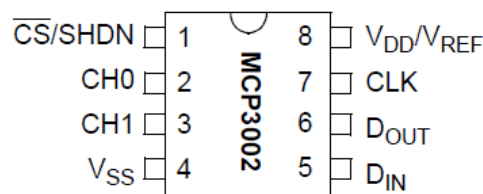


Figure 19: MCP3002 schematic (from microchip.com)

In this image, CS is the Chip Select, which indicates which one of the channel (CH0 or CH1) to use. CH0 and CH1 are the two available channels, where any analog input may be connected to. V_{SS} is the ground. Din and Dout are the input and output channels of the data in digital. CLK is the clock signal from the platform. And V_{DD} is the voltage, which in this case will be 3.3V.

The reading of analog data input also implies the use of the SPI (Serial Peripheral Interface), which is a master/slave bus whose communication is full-duplex. The Raspberry allows up to 2 SPI [9].

It is important to note that the SPI is automatically disabled on this platform. To enable it:

1. Input command `sudo nano /boot/config.txt`
2. Add this line at the end: `dtoverlay=spi=on`
3. Close and save the file
4. Reboot Raspberry Pi

Digital sensors

The digital sensors may be connected directly to the GPIO of the Raspberry Pi.

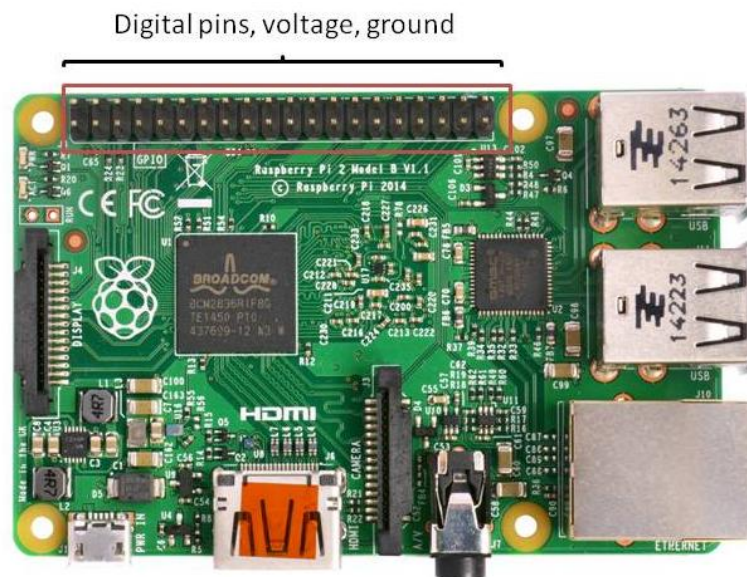


Figure 20: Raspberry Pi pins (from dev.px4.io)

For analog sensors, first we need to connect the Raspberry Pi to the MCP3002 and the sensor. The connection can be done following the schematic.

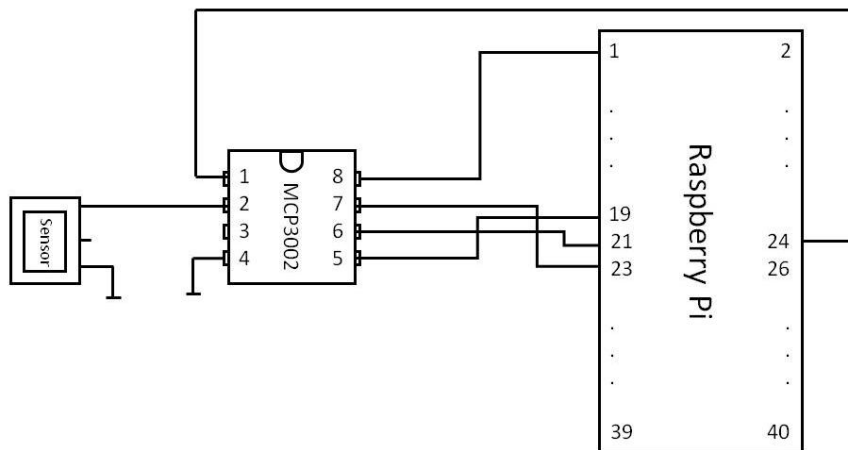


Figure 21: Raspberry Pi connection schematic

The pins' correspondence is as follows.

Raspberry Pi 2 pins	MCP3002 pins	Sensor wires
24 or 26	1	
	2	Data
	3	

Ground	4	Ground
19	5	
21	6	
23	7	
1	8	

Table 7: Raspberry Pi, MCP3002 and sensor pin connection

The first pin of the MCP3002 is the Chip Select, which can either be CS0 (pin 24) or CS1 (pin 26). In the event that there is only one input (so only one of the pins 2 and 3 from MCP3002 is used), the other may be left either unconnected or grounded. In any case, the unused CS pin on the Raspberry will not be available for normal use (digital write or read) while SPI is in function [10]. The third wire from the sensor (input voltage) has to be connected depending on the sensor specifications, as some sensors require 3.3V and others require 5V.

In this particular case the sensor output has been connected to CS0.

There is no specified Ground pin on the table for Raspberry Pi as there are many Ground pins on the platform: 6, 9, 14, 20, 25, 39, 30, 34.

See this page ([11]) for a detailed pinout graphic guide.

5. Sentilo server

One of the requirements of this project is to use Sentilo as the server. Sentilo is an open source sensor and actuator platform developed by the Barcelona City Council through the Municipal Institute of Informatics that began running in November 2012. It follows the idea of open data, so anyone can see and access the information, and it is developed so that users can easily deploy their own server. This Sentilo server offers an architecture composed of isolated layers; providers, server, and applications. This completely fits the system components that this project uses.

As explained in previous sections, providers are the devices that are connected to the Internet and that publish data on the server. The Sentilo server manages all access request petitions to read or publish data, and stores the information in real-time. The applications mainly utilize the data in the server to offer a service, although they are also able publish information. These layers are isolated so that they can work based in, but independent of, the other. All requests from and to the server are done through HTTP protocol.

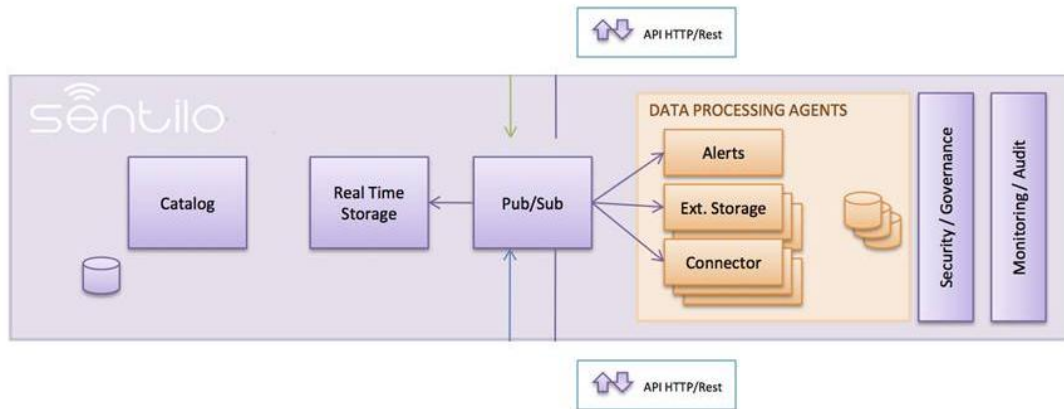


Figure 22: Sentilo architecture (from sentilo.io)

Sentilo offers an open source API based on REST, a resource-based architectural style. That is so Sentilo's elements or resources, which are explained in the following section, can be accessed through simple HTTP methods.

5.1 Sentilo resources and services

All the components on the above described architecture are identified by certain names in the REST architecture.

Resources: elements of the system.

Sensor: software or hardware device that can generate data about an environment observation. All sensors are to be identified by sensor type, data type they produce, and the measurement units.

Component: element that is composed by one or more sensors, mainly identified by its location.

Provider: entity representing a group of components and sensors and is in charge of the communication with Sentilo.

Client application: entity that utilizes the data.

Identifying these elements in this project's structure, the components would be the platform boards, and the providers would be the set of platforms with sensors.

Sentilo offers a number of services.

Alarm: service to record and retrieve alarms associated with an alert.

Alert: events triggered by a defined logic function of a sensor or component value.

Catalog: service to add, update, retrieve or remove any component or sensor.

Data: service to publish, read and delete observations from a sensor.

Order: service to publish (send) or retrieve orders to a sensor or actuator.

Subscription: service that allows the platform clients (applications or modules) to the data, order, and alarm services. Subscriptions can be created and cancelled through this service.

This project will be centered mainly on the Data and Catalog services.

5.2 Sentilo interaction through HTTP REST API

Information can be sent bidirectionally between the server, providers and client applications. All communication is done through HTTP. In the 'Data' service, the actions that can be performed are, following the methods:

GET: Request information.

POST: Send new data.

PUT: Update existing data.

DELETE: Erase data.

The sensor data that can be published on Sentilo is called 'observation', and its value can be numerical, boolean or text in nature. More than one observation can be sent at the same time, and they have to follow a template in JSON structure.

```
{ "observations": [
  { "value": "AAA", "timestamp": "dd/mm/yyyyThh:mm:ss", "location": "" },
  { "value": "AAA" },
  { "value": "AAA" }
]}
```

All observations can contain the fields value, timestamp, and location, but only the value field is mandatory. If the timestamp field is not supplied, the server will store it with the timestamp of which the data was received. Additionally, if the location is not supplied, Sentilo will use the location that has been stated on the sensor's provider information.

To send this message through HTTP, the observations have to be set with the HTTP header together with the PUT method.

URL	<code>http://server-ip:8081/data/providerId/sensorId</code>
Method	PUT
Content-Type	application/json
IDENTITY_KEY	Authorization token
	<pre>{ "observations": [{ "value": "AAA", "timestamp": "dd/mm/yyyThh:mm:ss", "location": "" }, { "value": "AAA" }, { "value": "AAA" }] }</pre>

Table 8: PUT method for Sentilo

To delete the latest observation of a sensor,

URL	<code>http://server-ip:8081/data/providerId/sensorId</code>
Method	DELETE
Content-Type	application/json
IDENTITY_KEY	Authorization token

Table 9: DELETE method for Sentilo

And to read an observation of a sensor,

URL	<code>http://server-ip:8081/data/providerId/sensorId?complement</code>
Method	GET
Content-Type	application/json
IDENTITY_KEY	Authorization token

Table 10: GET method for Sentilo

Where complements are a parameters that can be used to refine the data request, like the number of observations or time constraints.

All of these actions have to include an authorization token (the IDENTITY_KEY field of the HTTP messages). An authorization token is a string of letters and numbers, and it is unique to every provider. Once the message has been sent, the system checks if the Provider has authorization to perform it. It is important to notice that even if a server has enabled different providers, only the provider specified for a sensor will be able to send or retrieve its data.

After a request has been sent, the server will reply with a message containing a response code, which will indicate the status of the request.

Error Code	HTTP	Description
200	Success	Request accepted and processed correctly
4xx	Client Error	Error in request (Wrong format, forbidden mandatory parameters, ...)
401	Unauthorized	Unauthorized request: empty or invalid credential
403	Forbidden	Not authorized for the requested action
5xx	Server Error	Error processing the request

Table 11: Sentilo response codes (from <http://www.sentilo.io/xwiki/bin/view/APIDocs/Overview>)

All communications are secured by an authorization system. The Providers have a unique identity key that is to be used in all request orders.

5.3 Sentilo Catalog Web

Sentilo also offers a web application, the Sentilo Catalog Web. Normal users can monitor the sensors on the Universal viewer, a map that is based on Google Maps and that shows the location of the providers and their sensor information. Admin users or registered users that have permitted access can access the Administration menu of the application.

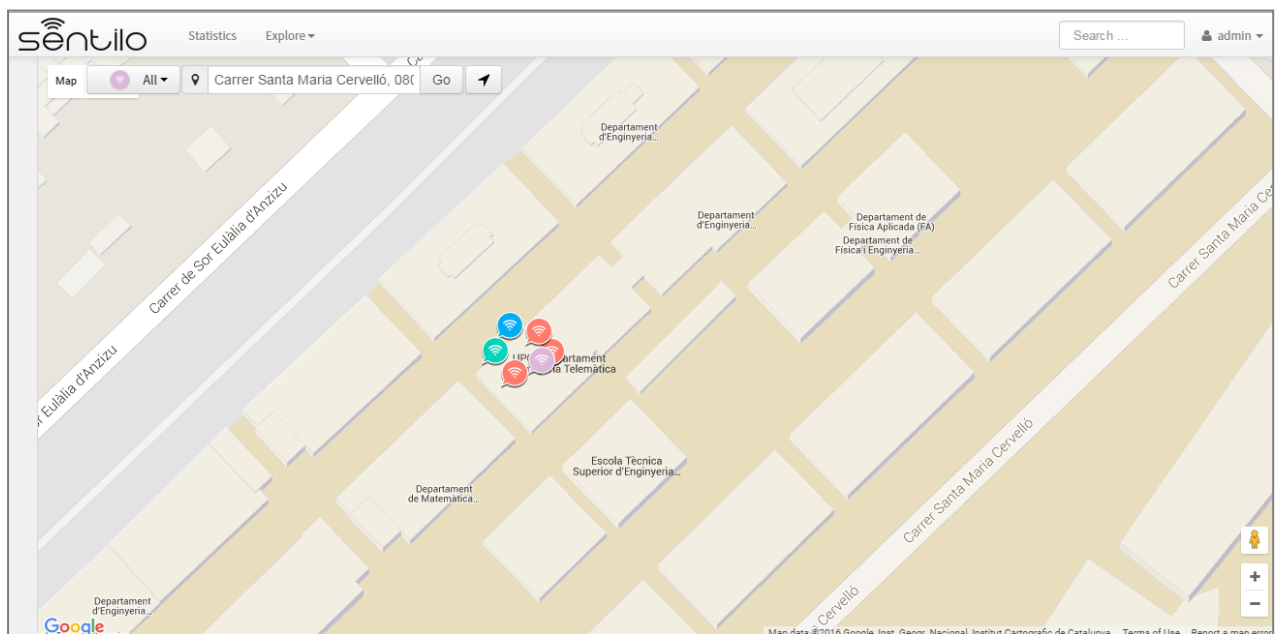


Figure 23: Sentilo Catalog Web's Universal viewer

In the Administration menu, these categories can be managed:

Users: different users can be defined along with their information: identifier, password, description, e-mail and role.

Applications: they are entities that can have permissions over providers (like reading or administrating permissions), and that can subscribe to events.

Providers: they can be created with name, description, identifier and contact information.

Components: they have to pertain to component type, and to a provider. They can be public or private, which will determine which users will see their data on the Universal viewer. They can also be static or mobile, and their location may be specified.

Sensors/Actuators: they have to be related to a component and to a provider, and can also be set as public or private. Optional information fields are the sensor type, the data type (numerical, boolean or text), and the measurement unit.

Alerts: this service can be managed in this category, with a specific typology, provider and client application.

Types of sensors/actuators: for this project, the types force sensor, magnetic switch, precision light sensor, presence sensor and temperature have been defined. Only the name and optional description have to be configured.

Types of components: for this project, the components are the platforms Arduino, Intel Edison, Raspberry Pi and Zolertia Z1. Similarly, only the name and description have to be configured, with the addition of a picture.

This project has used Sentilo Catalog Web as a way to confirm and validate if a connector is operating properly. Since it is updated in real time, once a platform connector is running it promptly displays the results.

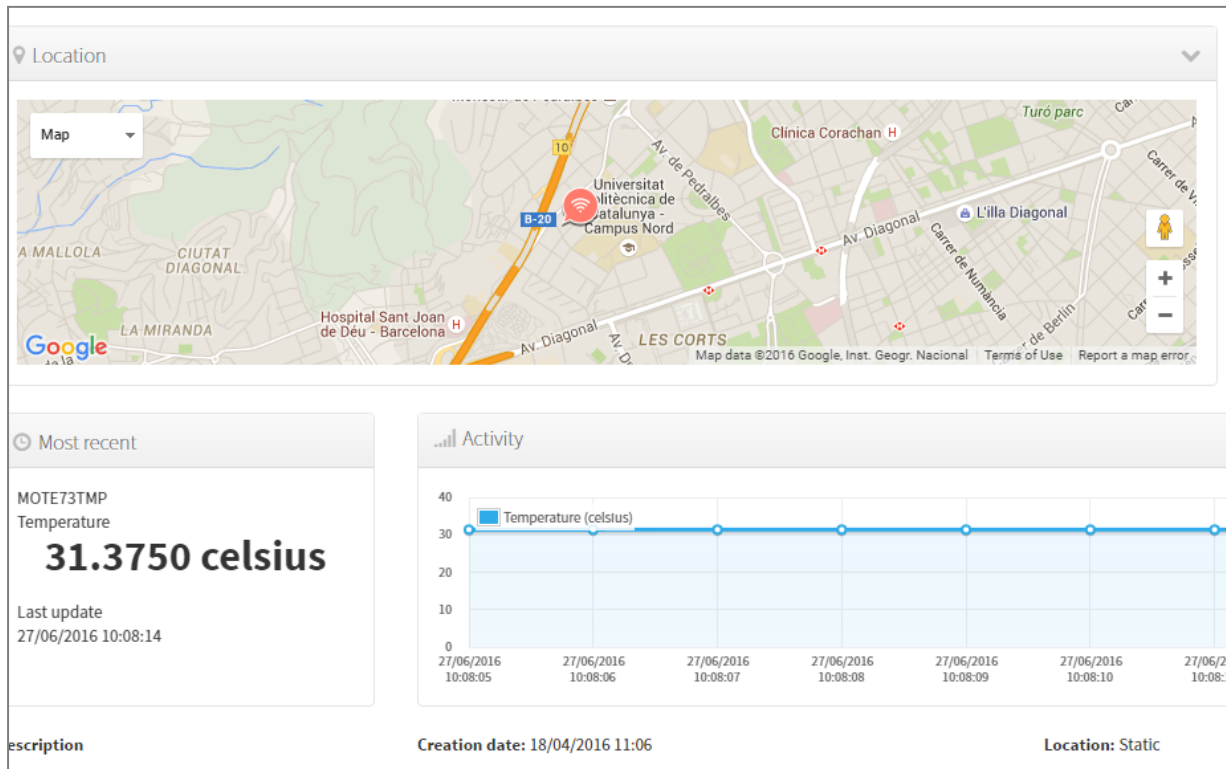


Figure 24: Sentilo Catalog Web view of sensor observation

6. Global connector

As defined in the project scope, one of the goals is to create a connector from the providers to Sentilo that could be used in any of the studied platforms. This idea is motivated by the idea that while the platform set up and data retrieving may be different in each platform, the part of the connector which then sends this data to Sentilo may be the same.

This applies to all platforms except Zolertia Z1. The Z1 connector was developed by student José Ignacio Mimbrero in his Master's thesis, and it is very versatile; it only needs a host computer that runs on Linux (or has a Virtual Machine of the OS) and that has at least one USB connection. The Z1 is also aimed at the creation of Z1 networks, which is a different approach than the rest of the platforms.

For Arduino, Intel Edison and Raspberry Pi, the most fitting IDEs and libraries to work on this system implementation have been studied, and before proceeding with the global connector it is necessary to review their compatibility across platforms.

6.1 Programming language compatibility

Platform	Programming environment			
	C/C++ (MRAA, UPM)	Node.js	Arduino	Python
Arduino			X	
Intel Edison	X	X	X	X
Raspberry Pi	X	X		X

Table 12: Programming compatibility across platforms

As seen in this table, if the connector was approached with C/C++ programming plus sensor libraries (like MRAA/UPM), the Arduino would be left out since the coding language structure is quite different from the other two platforms. On the other hand, if the connector was approached with Arduino programming, it would work for the Intel Edison but not for the Raspberry Pi. This conclusion complicates the initial idea of the connector.

Focusing in the Arduino, there would be roundabout way, such as connecting the Arduino to another computer. The PC, using C/C++, would make the platform access and read the data from the sensor, and then the computer would manipulate it and send it to Sentilo. This option, though, makes the presence of an external PC a requirement, method that defeats the initial purpose of using the Arduino in the first case.

Having reached this conclusion it is necessary to review this section of the project outcome, which is a general connector to work on all platforms. Envisioning it, that would be like running the same executable on all platforms. However, each platform has its way of compiling or uploading the program, so there needs to be an alternative. The focus then changes and becomes not the executable, but the code. In the previous paragraphs it has been established that only one code may not be the best solution, but if we look at this from a higher layer, there is the possibility of a program that could generate the desired program for any platform. This program would interact exclusively with the computer console, and the user would indicate the platform, sensors, and other necessary details for the set up.

The scope of the project would change like this:

Previous scope

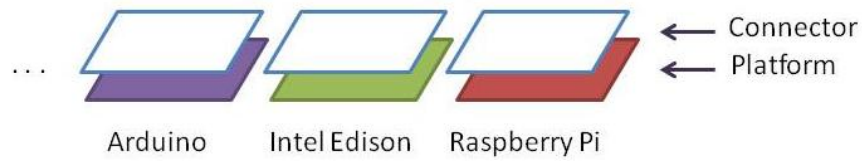


Figure 25: Previous project scope

Current scope

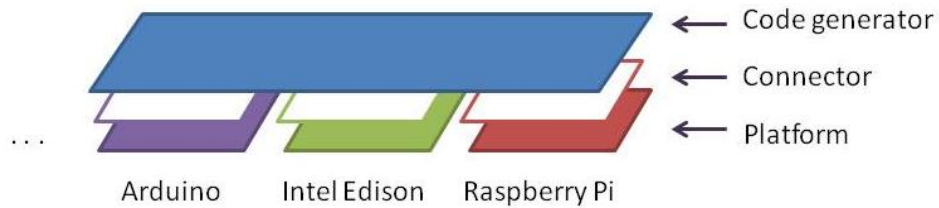


Figure 26: Current project scope

6.2 Individual connector flowchart and structure

Even though they will be written differently, the connectors for the different platforms are set to follow the same program structure to gain parallelism across platforms, and a better understanding of the overall functionality.

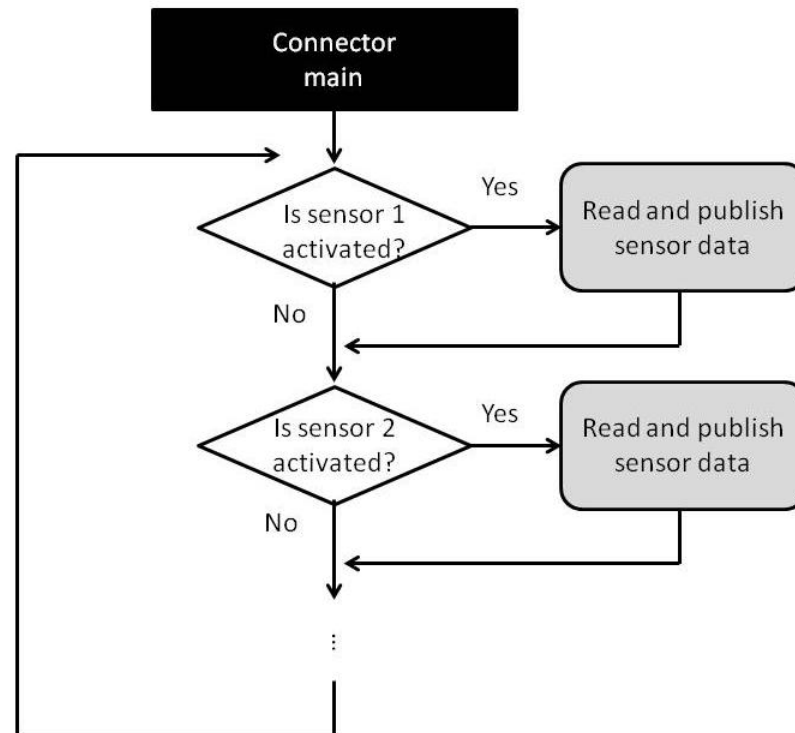


Figure 27: Connector main flowchart

The main of the connector consists in a loop that goes on until it is interrupted. In the loop, it reads and sends the data from the sensors.

The connector has been implemented so that, even if a platform may not be actively using a particular sensor, the function to retrieve and manipulate its data is still there. This was implemented to ease the task of debugging the program. All sensors can be activated and deactivated at any time in the properties section of the code.

```

/*Active sensors*/
const bool light_active = true;
const bool gas_active = false;
const bool temperature_active = false;
const bool humidity_active = false;
const bool magnetic_active = true;
const bool presence_active = false;
    
```

Inside the loop, there is an 'if' instance for every sensor checking whether it is active before proceeding. If a sensor is active, the program carries out three tasks; 1) read the data of the

sensor and modify if necessary, 2) format the data so that it is ready to be sent, and 3) send the formatted data to Sentilo.

When retrieving the sensor observations, it has to be differentiated between analog and digital sensors.

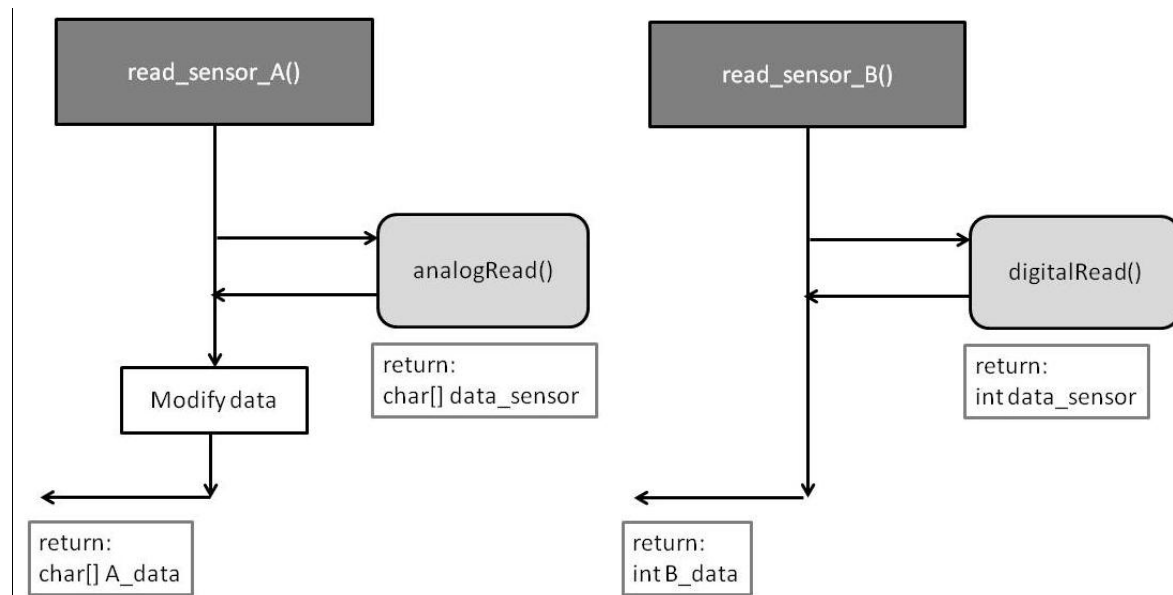


Figure 28: Sensor reading flowchart

For analog sensors, the sensor data is accordingly read as analog with the pin number, and a 10-bit value is returned. This value is then modified to the range of units of each sensor, as specified in section 3. The modified data is returned to the main.

For digital sensors, the sensor data is accordingly read as digital with the pin number, and a '0' or '1' is returned as integer number.

Before publishing this information to Sentilo, it needs to be structured in JSON format and encapsulated with an HTTP header.

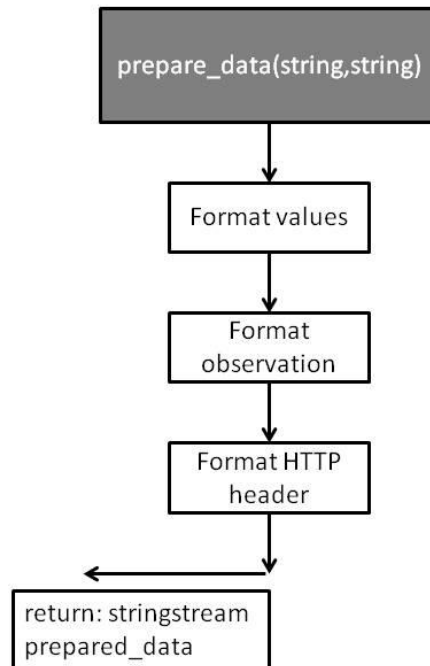


Figure 29: Prepare data flowchart

The name of the sensor and the data (as String) are given as parameters. First, it creates the JSON pair of key and values with the data.

```
{"value": "11.5"}
```

It then encapsulates this within the 'observations' in JSON. This is what will be sent with the HTTP PUT method.

```
{"observations":[
  {"value": "11.5"}
]}
```

And finally, it sets the HTTP header, and encapsulates the message. The resulting HTTP message follows this table:

URL	<code>http://server-ip:8081/data/providerId/sensorId</code>
Method	PUT

Host	Server-ip
IDENTITY_KEY	Authorization token
Content-Length	Size
Content-Type	application/json
	<pre> {"observations":[{"value": "11.5"}]} </pre>

Table 13: HTTP message with observation to send

The last function is to send this to Sentilo.

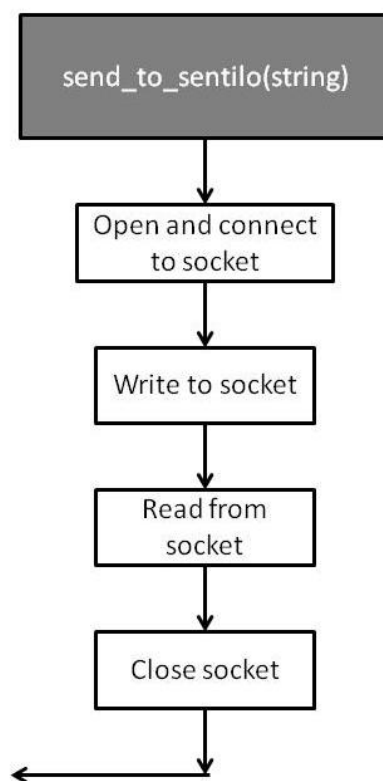


Figure 30: Send to Sentilo flowchart

A socket connection is opened and connected using the server address. The data that has been prepared to send in the previous function is provided as parameter, and written on the socket along with its size. Finally, the socket is read to check the response and whether the exchange has been done correctly, and then closes it.

6.3 Global connector

The global connector is a program written in C/C++ that interacts with the user through the console and whose main goal is to print out a file or files that fit the user's needs about the platform and its sensors.

The printed file will be ready to be compiled and to run on the designated platform to connect the sensors and platform to the Sentilo server.

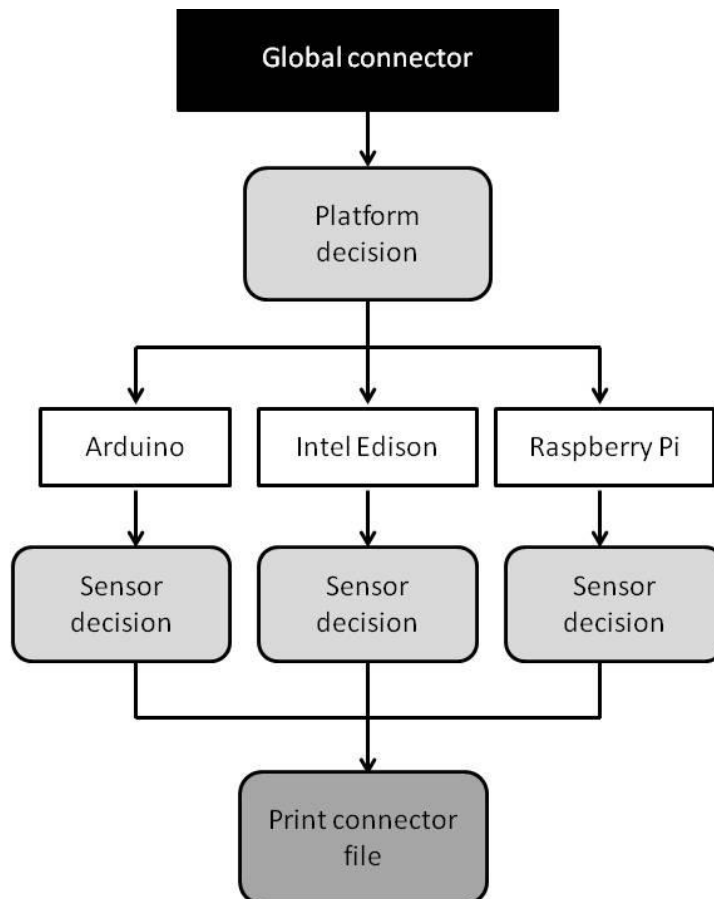


Figure 31: Global connector flowchart

The platform decision is made with a menu that is shown on the console.

The connector can be generated for the following platforms

1. Arduino
2. Intel Edison
3. Raspberry Pi

Please select an option:

As the decision option is saved, it is then used on a switch/case selection statement. On all platform options the following sensor decision menu appears the same.

```
Please select the wanted sensors, one at a time.  
1. Temperature sensor      [OFF]  
2. Magnetic switch         [OFF]  
3. Precision light sensor   [OFF]  
4. Presence detector        [OFF]  
5. Force sensor             [OFF]  
6. Finish and print the code.
```

This will appear in a loop where any listed sensor can be activated until the printing option is selected. The printing of the connector file is done with the 'fopen' function.

```
FILE *out;  
out << fopen("connector_platform_A.c", "w");  
fprintf(out, "/*Platform A connector*/");
```

Using this function, what can be printed is actually of char * nature. To avoid an unnecessary lengthy connector code and to aid with printing large blocks of code, the function 'fopen' and 'fread' will also be used. They will assist in opening a file (with those blocks of code written on it) and reading it, respectively. Those pieces of code will be stored in a folder of the connector, grabbed, and printed onto the new printed connector.

Once the global connector has run, the resulting file will be found on the same folder the global vonnector has run on. If the file ('connector_plarform_A.c') does not exist, it will be created, and if it does exist, the global connector will overwrite it.

6.4 Individual platform specifications

The main code structure is the same, but all platforms have specific IDEs, libraries and methods to accomplish those means.

6.4.1 Arduino specifications

Arduino IDE

The programming and general communication of the platform is done through the Arduino IDE, while the board is connected to the computer by USB. In Arduino, the programs are called “sketches”, and they are programmed on the IDE, compiled, and finally uploaded on the platform.

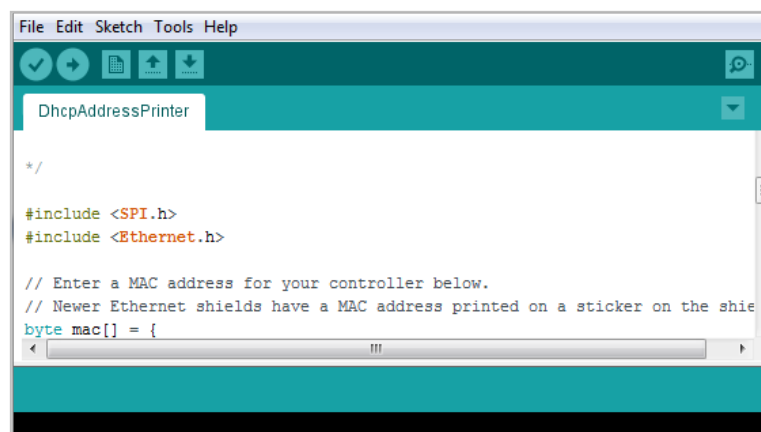


Figure 32: Arduino IDE

The IDE uses a programming language based on Wiring and implemented in C/C++. [12] While its structure retains a semblance to the formers, there are some that are unique to Arduino.

All sketches have the same layout: headers (libraries and global variables or constants), setup, loop, and additional functions.

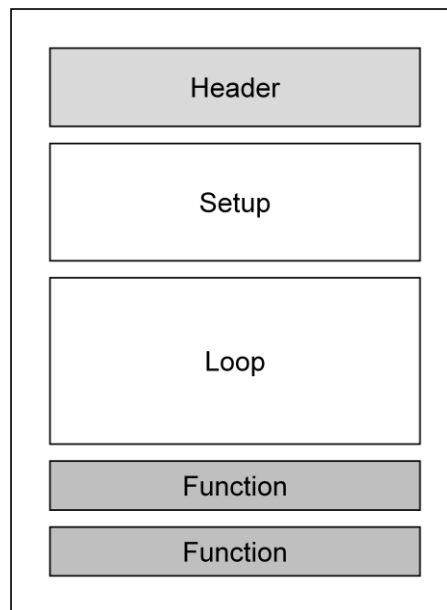


Figure 33: Arduino sketch layout

The setup part of the code is only run once, at the beginning as the Arduino is powered up. The loop is equivalent to the ‘main’ of other programs, but repeated indeterminately. The functions can be called during the setup and the loop.

In Arduino, the console is called ‘Serial monitor’. It has to be initialized during the setup, and the baud rate (the rate of the communication information) has to be indicated. The baud rate can be any of this range: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200 [13].

```
Serial.begin(9600);
```

To use it, there is the function `Serial.print()`.

```
Serial.print("Reading data: ");  
Serial.println(sensor_data);
```

Different types of data output have to be written in different instances, and to change to a new line in the end the function changes to ‘println’.

To read digital and analog data from the pins,

```
int light_data = analogRead(pin);  
int presence_data = digitalRead(pin);
```

The Arduino can read an input up to 10 bits, so the read analog values may vary from 0 to 1024. In the case of digital pins, the value read can be '0' or '1'.

Sentilo library

Sentilo has provided an external library equipped with functions that perform the actions of connecting to Internet as well as managing the requests and responses to and from the server.

`SentiloClient`: constructor, sets the host, port, and content type of the communication.

`dhcp`: configures the MAC and sets up the Ethernet connection with the `begin()` function.

`begin`: calls the `begin()` function from the Ethernet library, which initializes the Ethernet library and configures the network settings with the MAC and IP.

`getCatalog`: retrieves data from the catalog by setting the HTTP message header with the provider identity key and the GET method.

`publishObservation`: sets the HTTP fields of header, path, and observation messages given as parameters of the function and returns with the PUT method.

`registerSensor`: sets the HTTP fields of header, path and message and returns with the POST method.

`createObservationInputMessage`: sets the data given as parameters in the standard JSON format to send.

`createSensorInputMessage`: creates the message with the necessary fields (sensor name, description, type, data type,...) in the JSON format to publish the sensor (as needed in `registerSensor` function).

`get`: returns a request with the HTTP GET method, path, and body of the message.

`post`: returns a request with the HTTP POST method, path, and body of the message.

`put`: returns a request with the HTTP PUT method, path, and body of the message.

`request`: sets up a connection to Sentilo server with the host and port and proceeds to write the HTTP request with the header, path, and message body, and checks the response code to see if it has been done correctly.

`readResponse`: if the response is not null, it reads it and returns the response code.

write: print information on the client.

6.4.2 Intel Edison specifications

Intel System Studio IoT Edition

Intel System Studio IoT Edition is an Eclipse plug-in that allows the connection from the computer to the board, and to write and upload the programs onto the board. It provides the libraries MRAA and UPM. MRAA is a low-level library that offers access to the input and output information to the pins. UPM is a library that is based on MRAA and offers a defined set of sensors (light, temperature, humidity...) so that there is no need to further manipulate the data, as it is with MRAA.

First, the sensors need to be declared with their correspondent pin number. Analog pins are declared as 'Aio' and digital pins are declared as 'Gpio' of the MRAA class. The pin number is the same as the physical pin number.

```
mraa::Aio* sensor_light_pin = new mraa::Aio(1);  
mraa::Gpio* sensor_magnetic_switch = new mraa::Gpio(3);
```

As the digital pins may be input or output pins, after the declaration it is needed to specify the direction.

```
mraa::Result response = sensor_magnetic_switch->dir(mraa::DIR_IN);
```

To read the analog sensor data, Intel Edison reads analog-to-digital data in 12 bits [14], that's why the data is placed in a 16-bit data type (the immediate smaller is 8-bit, too small in this case). The `read()` function captures a sample from the analog input pins.

```
uint16_t data_sensor = sensor_light_pin->read();
```

Alternatively, to read digital sensor data, Intel Edison returns an integer that can be either '1' or '0'.

```
int switch_state = sensor_magnetic_switch->read();
```

6.4.3 Raspberry Pi specifications

Libraries

To program the connector, since the Raspberry has an Ubuntu Mate operating system, it is possible to write the code on a regular text editor and then compile and run it on the terminal, and there is no need for an IDE or other software to program the platform. However, unlike Arduino and Intel Edison, Raspberry does not come with built in libraries that enable the use of GPIO easily. However, there are public libraries to work with in a wide variety of programming languages like C, C++, Ruby, Python, Node.js, Java, and others [15].

The first library that was studied was WiringPi because of it provided an easy to use interface. There was, however, a bug in the part of the library that controlled the SPI from MCP3002, so it was discarded. Instead, the connector is based on the library bcm2835, which is written by Mike McCauley, and is in C (and C++ compatible). It contains functions to read and write on digital pins, as well as to use SPI and I2C.

All programs that use this library have to be run as root.

There are several functions that will be necessary. First, to initialize the library and get the pointers to the internal memory for the registers. It returns 1 if it initializes correctly, and 0 if it fails.

```
bcm2835_init();
```

And to close it, at the end of any program.

```
bcm2835_close();
```

Before reading and writing on any digital pins, they have to be declared as such with Function Select, the pin number and the constant of either input or output.

```
bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);  
bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);
```

And, if there is any need, the pin can be set in pull-up mode.

```
bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);
```

To read the digital input, there is the Level function. It may return either HIGH or LOW.

```
uint8_t value = bcm2835_gpio_lev(PIN);
```

And to write a digital output, there is the Write function. It may be set to HIGH or LOW.

```
bcm2835_gpio_write(PIN, HIGH);  
  
bcm2835_gpio_write(PIN, LOW);
```

For analog sensors, the program needs to initialize the SPI after initializing the library. It returns 1 when run successfully, and 0 if it fails. After that, the communication through SPI needs to be configured. Below are the default settings. Note that this is where the Chip Select is specified.

```
bcm2835_spi_begin()  
bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);  
bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);  
bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_65536);  
bcm2835_spi_chipSelect(BCM2835_SPI_CS0);  
bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);
```

Finally, to read an analog input, the function Transfer is used.

```
bcm2835_spi_transfern(buf, sizeof(buf));
```

6.4.4 Zolertia Z1 specifications

The connector of the Z1 platforms, since it is intended to run on Linux OS, has a GUI (Graphical User Interface).

SentiloAPI

This java class implements the requests to the Sentilo server. The functions that are described are getData, postData, and putData, which correspond to the HTTP methods of GET, POST, and PUT. All functions open a connection with the URL of the Sentilo server and send a request.

Sensor

Constructor
Sensor(URI uri, WebLink resource,String location)
Functions
public synchronized void doObserve()
public synchronized void cancelObserve()
public void registerSensor()
public synchronized String getType()
public synchronized String getTitle()
public synchronized String getId()
public synchronized String getComponent()
public synchronized boolean isObserved()
public boolean isSentiloRegistered()

Table 14: Sensor class functions

The constructor takes the sensor URI, COAP Client resource, and location as parameters. The resource is a Weblink object that contains the information of the sensor that has been retrieved by using the GET method on the COAP Client. It then sets all the class attributes by extracting information of the resource.

`doObserve()`: if the sensor is registered and it is not observing, it creates a COAP Client and sets it so that it starts observing. Afterwards, it sends the data to Sentilo and checks if everything has gone correctly with the response code it gets in return.

`cancelObserve()`: cancels the observation for the sensor.

`registerSensor()`: first it checks the boolean class attribute 'registered'. But, to confirm that it is indeed not registered, it calls on the function `isSentiloRegistered()`. If it is indeed not registered, it sets the data to send to Sentilo in a POST method and sends it.

`isObserved()`: it checks if the COAP Server relation which has been retrieved from the observation, and which manages the Observe connection, is null.

`isSentiloRegistered()`: this function makes a GET request to Sentilo with the name of the sensor component. If the response it receives is not null, it then creates a JSON array. The function then checks every element of the array to see if any providers and sensors match the one it is comparing from the class. It returns true or false depending on these comprobations.

Highlighted functions of the table return the class parameters as they are set to private.

Mote

Constructor
public Mote(String id, String ip,String prefix)
Functions
public String getIp()
public String getPrefix()
public String getId
public boolean discover()
public Set<WebLink> getResources()
public ArrayList<Sensor> getSensors()
public void registerSensors()
public void stopObserve()

Table 15: Mote class functions

The constructor takes the Mote id, Mote IPv6 address and its prefix as parameters, and sets them as the class properties. It also creates a new sensor array (without defining any element), and sets the Mote location by creating a COAP Client with the URI and reading the “location” field of the JSON received as response.

`discover()`: this function creates a new COAP Client with the URI of the mote. It checks if there are resources. If there are, for every resource it creates a new sensor with the URI, sensor and location, and adds the sensor to the list of sensors of the class. It returns ‘true’ or ‘false’ depending if resources were found or not.

`registerSensors()`: for every sensor on the list, it registers them with the `registerSensor()` function, and sets them on observation with the `doObserve()` function.

`stopObserve()`: calls the `cancelObserve()` function for every sensor of the list.

Highlighted functions return the class parameters as they are set to private.

SensorNetwork

Constructor
public SensorNetwork(String url)
Functions
public synchronized void findMotes()
public synchronized Mote getMote(String id)
public synchronized HashMap<String,Mote> getAllMotes()
private synchronized String getXML()
public void registerMotes()
public void deleteAllMotes()

Table 16: SensorNetwork class functions

The constructor gets the URL as parameter, and it creates a new map of Motes, and then calls the functions findMotes() and registerMotes();

findMotes(): this function reads the XML that contains the information about the motes. Once read, it creates a NodeList with new elements for every instance of 'node' in the XML. Then, for every item in the list, it checks that the class' Motes map already has that particular node included. If it does not, it sends a request and if the response is not null, it adds the Mote to the map.

getXML(): this function creates a BufferedReader and uses it to read the response to a connection with the class URL, and saves the information in a string.

registerMotes(): for every mote on the map it calls the registerSensors() function.

deleteAllMotes(): for every mote on the map it calls the stopObserve() function. At the end, it clears the map.

Highlighted functions return the class parameters as they are set to private.

MainApp

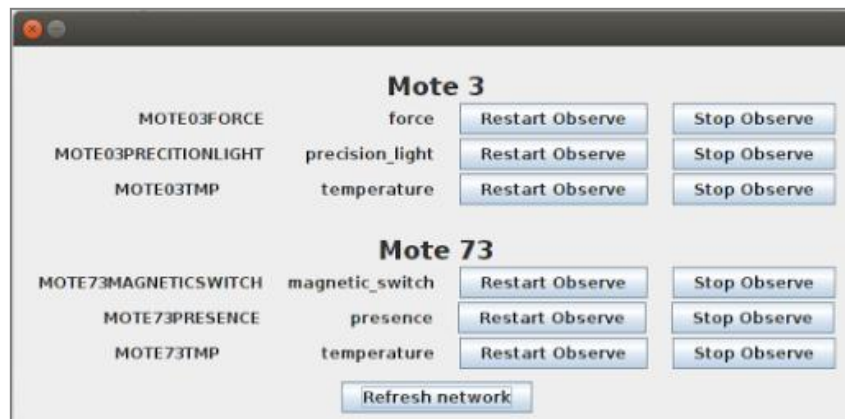


Figure 34: Z1 Connector MainApp GUI

On start up, the window with a single 'Refresh Network' button appears. When this button is clicked, the connector runs the 'getAllMotes()' function on the Sensor Network variable. Both will be explained later, with their class. After that, the program generates a list of sensors for each mote that has been found, and sets both Mote and sensors on the GUI display. All sensors will have a 'Restart Observe' and 'Stop Observe' buttons, which call the functions with the same name of the sensors.

7. Client application

On the other end of the system, there are the applications. They utilize the data that is generated by the sensors and sent to the Sentilo server. The client application that has been designed is a web page.

The client application will be in the form of a web page, since it complies with the requirement of being accessible for a wide variety of hardware devices and operating systems.

However, a browser by itself cannot perform these HTTP exchanges as it is not permitted by computers' firewalls. This leaves the application design in two clear sections; one to communicate to Sentilo, and the other to show the results on the page.

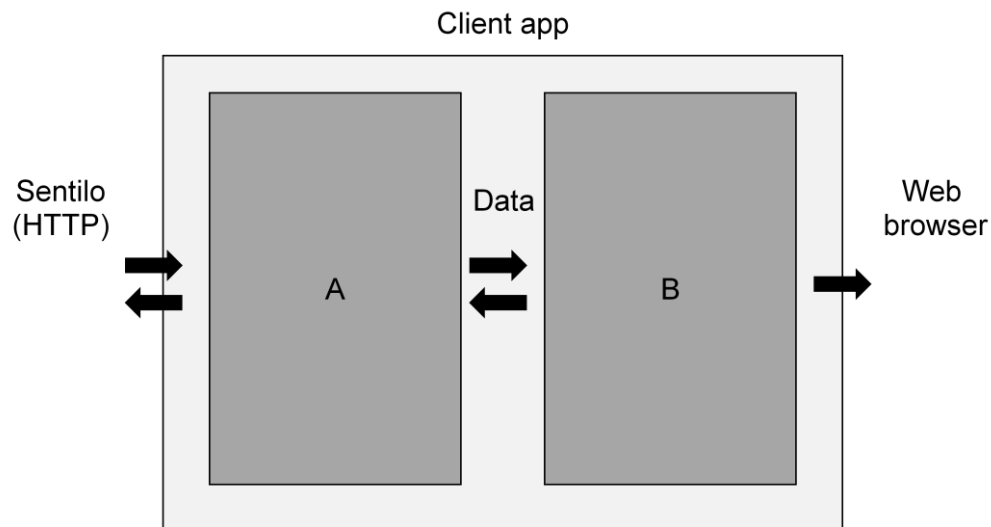


Figure 35: Client application design

The ‘B’ section of the application is clearly the web page itself, and it is backed by the ‘A’ section. The former one can be implemented in a number of ways. As the communication through HTTP could easily be done in Java, the more plausible options may be the Java Servlet technology and the JavaServer Pages technology. The first offers the option of customizing the handling the GET and POST requests, as well as being able to handle a lot of code in Java. The second also allows Java programming, although it is intended for more static pages that do not have many coding in Java.

7.1 HTTP requests and responses

The data from Sentilo can be accessed through HTTP requests based on their REST API. The request must indicate the URL of the sensor resource, as well as the method.

URL	<code>http://server-ip:8081/data/providerId/sensorId?limit=5</code>
Method	GET
Content-Type	application/json
IDENTITY_KEY	Authorization token

Table 17: Client application HTTP request

The provider id as well as its sensor id have to be provided, together with the authorization token to be able to perform the request. The parameters after the sensor id may be used to specify different requirements.

From: indicates the starting time the observations have to start with.

Limit: indicates the total number of observations to retrieve.

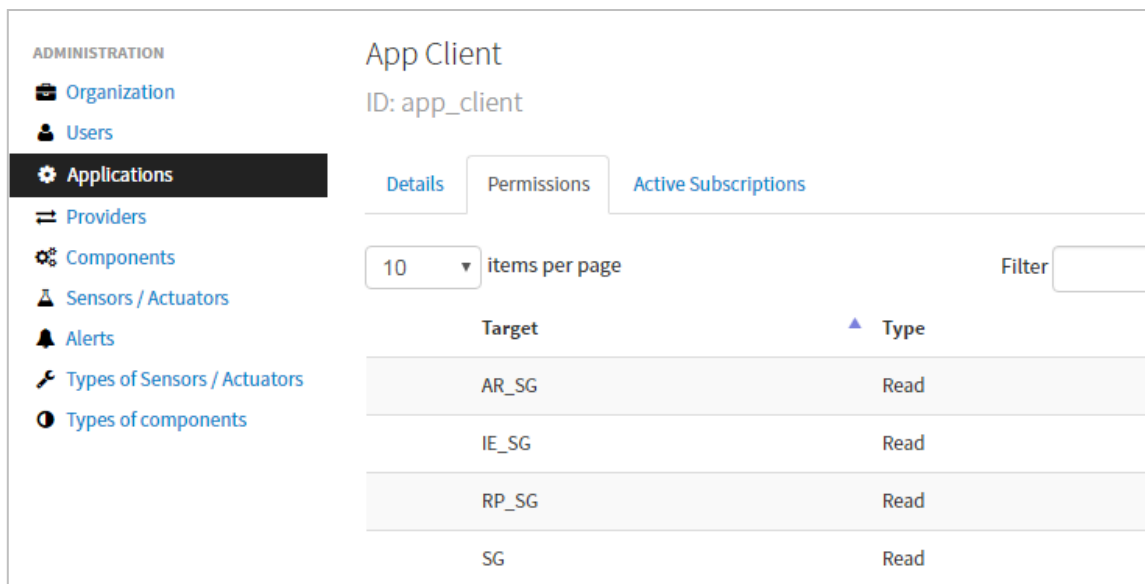
To: indicates the ending time the observations have to stop with.

The response from Sentilo is a JSON structure with the 'observations' as main field.

```
{
  "observations": [
    {
      "value": "11.82",
      "timestamp": "12/06/2016T10:05:47",
      "location": ""
    },
    {
      "value": "11.91",
      "timestamp": "12/06/2016T10:05:45",
      "location": ""
    },
    {
      "value": "12.01",
      "timestamp": "12/06/2016T10:05:44",
      "location": ""
    },
    {
      "value": "12.01",
      "timestamp": "12/06/2016T10:05:43",
      "location": ""
    },
    {
      "value": "11.91",
      "timestamp": "12/06/2016T10:05:42",
      "location": ""
    }
  ]
}
```

When the location is not specified, the system uses the static component's defined location.

To be able to request data from Sentilo, the petition needs to be verified with the authorization token. Every provider has a unique authorization token, and with it they can update and retrieve each sensor's observations. However, as an application has to be able to retrieve all sensors' observations, the best option is to create an Application profile on Sentilo Catalog Web, and enable the permissions to 'read' observations from all providers.



The screenshot shows the 'App Client' configuration page for 'app_client'. The left sidebar contains navigation links: ADMINISTRATION, Organization, Users, Applications (selected), Providers, Components, Sensors / Actuators, Alerts, Types of Sensors / Actuators, and Types of components. The main content area has tabs for Details, Permissions, and Active Subscriptions. Under the Permissions tab, there is a dropdown for '10 items per page' and a 'Filter' input. Below this is a table with two columns: 'Target' and 'Type'.

Target	Type
AR_SG	Read
IE_SG	Read
RP_SG	Read
SG	Read

Figure 36: Application permissions on Sentilo Catalog Web

By using only this authorization token, the system can send requests about all providers and sensors.

7.2 Connection with Java class

The actual code to send an HTTP request to Sentilo and receive and manage the response's data will be written in a Java class.

The class only has two methods.

getData

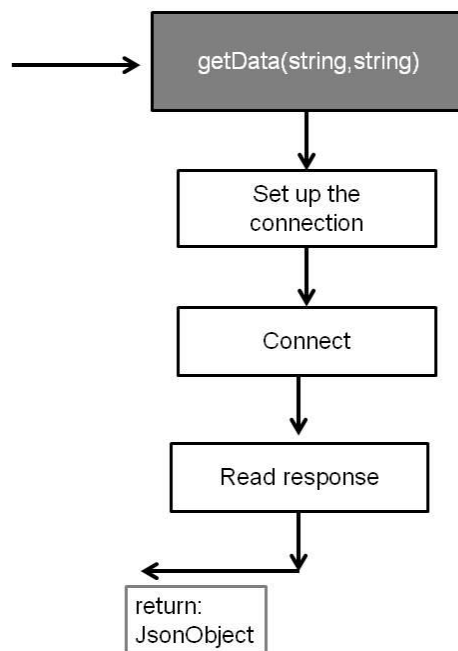


Figure 37: getData flowchart

With the provider id and sensor id given as parameters, it sets the connection with the parameters as specified in the above table.

First, the URL is set.

```
url = new URL("http:// server-ip:8081/data/" + provider_id + "/" +
sensor_id + "?limit=5");
```

And then the connection is created and configured.

```

URLConnection conn = (URLConnection) url.openConnection();

conn.setRequestMethod("GET");
conn.setRequestProperty("Content-Type", "application/json");
conn.setRequestProperty("IDENTITY_KEY", auth_token);
conn.connect();

int responseCode = conn.getResponseCode();
  
```

Once the request has been sent, it then reads the code of the response. It only proceeds further if the response code is '200', which indicates that everything is correct.

Since the response is in JSON, the class utilizes the javax.json library as a JSON parser to be able to handle the data easily. At this point, it reads the whole response message as a JsonObject. A Json Reader has to be created, with the String response as StringReader.

```

JsonReader reader = Json.createReader(new StringReader(response.toString()));
JsonObject jsonObj = reader.readObject();
reader.close();
  
```

getObservation

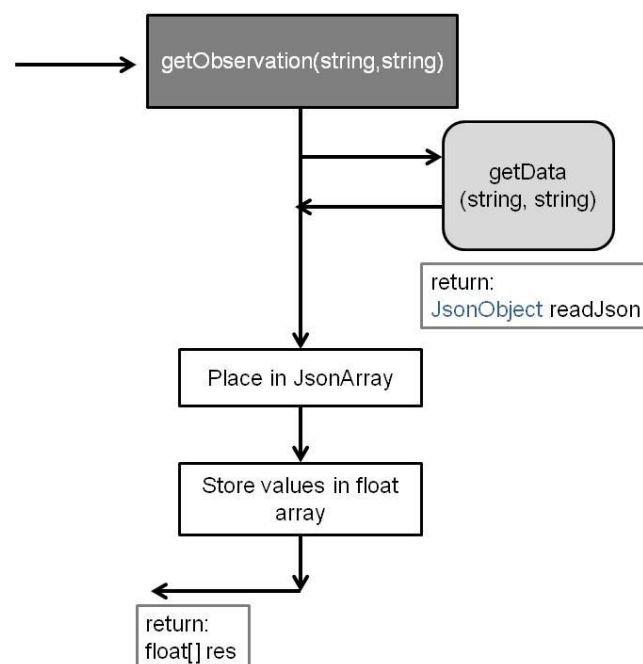


Figure 38: getObservation flowchart

Only the function `getObservations` is called externally, and in turn this function calls the `getData` function. As with `getData`, the provider and sensor ids are given as parameters.

The sensor observation data is placed on a Json Array. And in turn, the Json Array has Json Objects, which have key and value pairs.

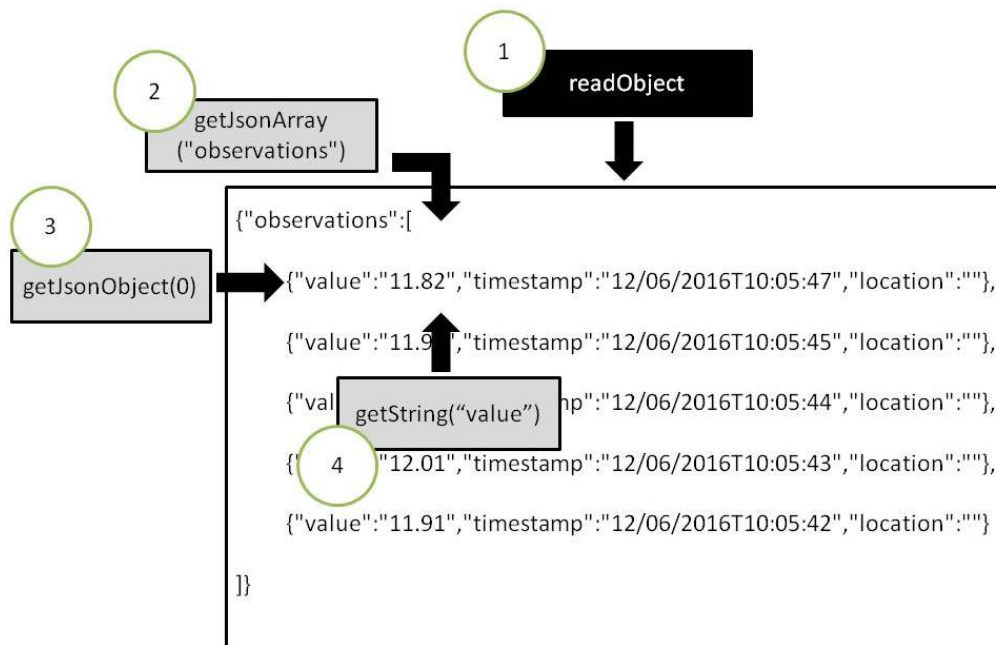


Figure 39: JSON reading order

First, the array corresponding to the key “observations” is read from the whole Json Object returned by `getData`.

```
obsArray = readJson.getJSONArray("observations");
```

Then, with a ‘for’ loop, each array element is accessed as Json Object, and then the value is stored in the variable array that will be returned at the end of this function. The value is read as String, but will be stored as a float.

```
obsValue = obsArray.getJSONObject(i);
valString = obsValue.getString("value");
res[i] = Float.parseFloat(valString);
```

7.3 Web application with JSP

Since the client application will rely heavily on scripting code, the technology used for 'B' section is JavaServer Pages, backed by the Java class on the 'A' section.

The JavaServer Pages (JSP) is a text file that can contain static content, such as web page descriptors like HTML, and dynamic content, such as Java. The dynamic content of the page is compiled on the server, so that when a web browser shows that page, they get strictly the static content with the results of the dynamic content included.

An example of a JSP page with both types of content would be

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

    <%! int sum (int a, int b){
        int res = a + b;
        return res;
    } %>
<!DOCTYPE html>
<html>
    <head><title>Hello World Title</title></head>
    <body>
        <!-- Start of the main page -->
        Hello World<br/>
        <%
            out.println("Your IP address is " +
request.getRemoteAddr());
        %>
        <p>
            Today's date: <%= (new
java.util.Date()).toLocaleString()%>
        </p>
        <p>This is a sum: <%= sum(1,2) %>
        </p>
    </body>
</html>
```

The most important things to note are:

- All JSP dynamic content goes in inside the `<% %>` marks.
- The language in which the dynamic content is written has to be specified, such as `<%@page language="java"%>`.
- Declarations of variables and functions go inside `<%! %>` marks.
- General Java operations may go inside `<% %>`.
- To print out static content for the web page
 - Using the 'out.println' Java function, `<% out.println("Sample text"); %>`
 - Using JSP expressions, as they transform the variable to String and prints it, `<%= sum(1,2) %>`.
- JSP comments are written with the `<%-- --%>` marks. The comment format in HTML can also be used, with the difference that the page source code will contain them, but not the JSP ones.

However, the browser will see this

```
<!DOCTYPE html>
<html>
  <head><title>Hello World/A Comment Test</title></head>
  <body>
    Hello World!!<br/>
    Your IP address is 0:0:0:0:0:0:0:1

    <p>
      Today's date: 12-jun-2016 9:47:45
    </p>
    <p>This is a sum: 3
    </p>
  </body>
</html>
```

As page source code.

With this, we can easily import a Java class into the JSP, so that we can use its functions in order to display one result or another on the web page as static content.

```

<%@ page import="projectServlet.SentiloClass" %>

<%! String disp_AR = sentiloConn.getDisplayLight("AR_SG",
"ARDUINOLIGHT"); %>

<html><body>
<td id="<%=disp_AR%>"><a href="/proj/light_arduino.jsp">A1101</a></td>
</body></html>
  
```

This action prints 'A1101' in a style that is determined by a function from 'SentiloClass'.

This web app will be able to run not only on a desktop computer web browser, but will also be able to be viewed on tablets and on mobile phones.

8. Integration

This section will showcase the results of the integration of the whole system.

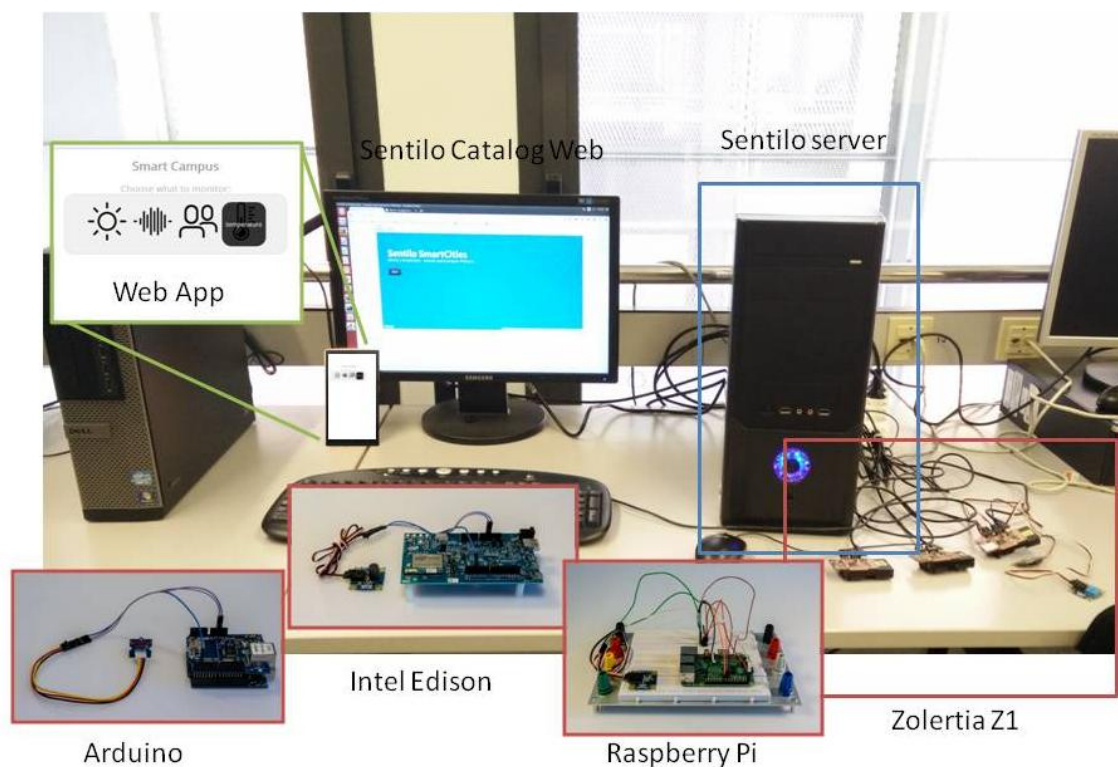


Figure 40: Integration of the complete system

Arduino integration

Integration of the Arduino Uno R3 with an Ethernet Shield and a magnetic switch.

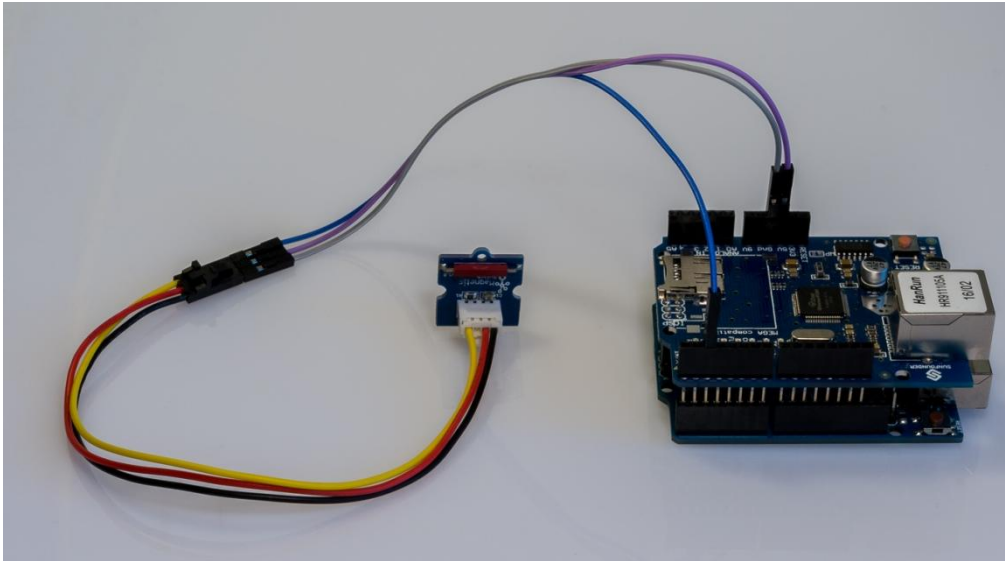


Figure 41: Arduino integration with a magnetic switch sensor

Running the Arduino connector to Sentilo.

```

[AR] 1. Read the data from the presence detector: 1
[AR] 2. Format the data from the sensor: true
[AR] 3. Send the observations to Sentilo.
[AR] 4. Code 200: Data sent successfully.

[AR] 1. Read the data from the light sensor: 567.38
[AR] 2. Format the data from the sensor.
[AR] 3. Send the observations to Sentilo.
[AR] 4. Code 200: Data sent successfully.

[AR] 1. Read the data from the magnetic switch: 1
[AR] 2. Format the data from the sensor: true
[AR] 3. Send the observations to Sentilo.
[AR] 4. Code 200: Data sent successfully.

[AR] 1. Read the data from the presence detector: 1
[AR] 2. Format the data from the sensor: true
    
```

Figure 42: Arduino connector integration

The system will print out all the steps of the connector with the relevant data, and if there are any errors it will indicate so.

Additional information logs can be printed (like formatted data, response data, etc), although they are commented in this last stage of the integration.

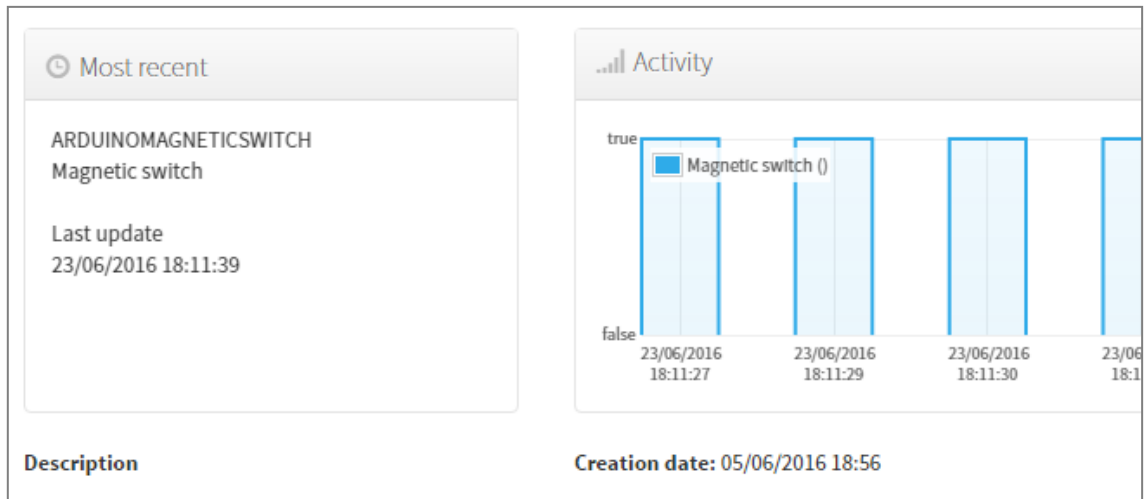


Figure 43: Arduino integration results as shown on SCW

These are the last observations that can be seen on the Sentilo Catalog Web.

Intel Edison integration

Integration of the Intel Edison with Arduino Kit with a digital presence detector sensor.

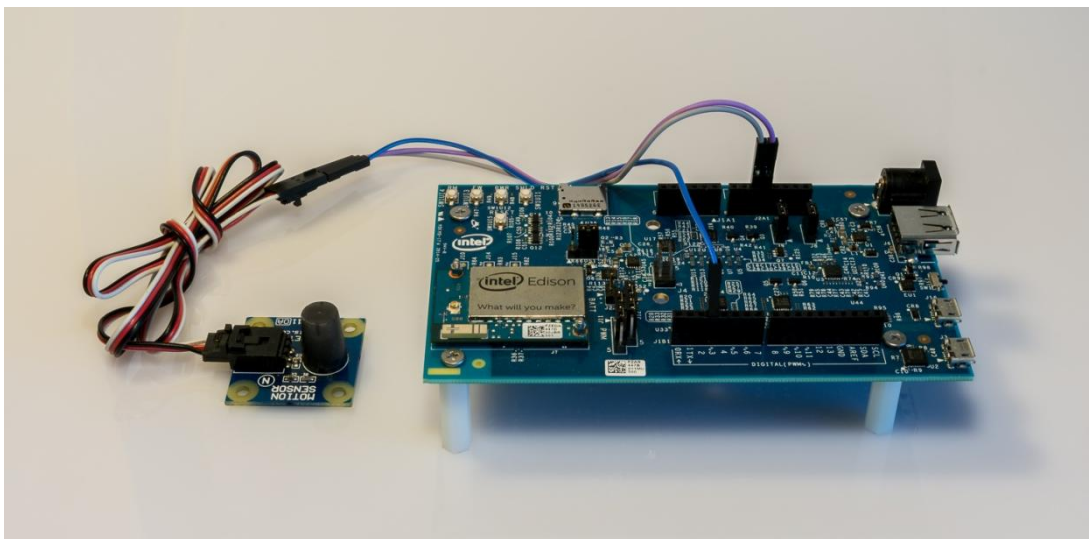
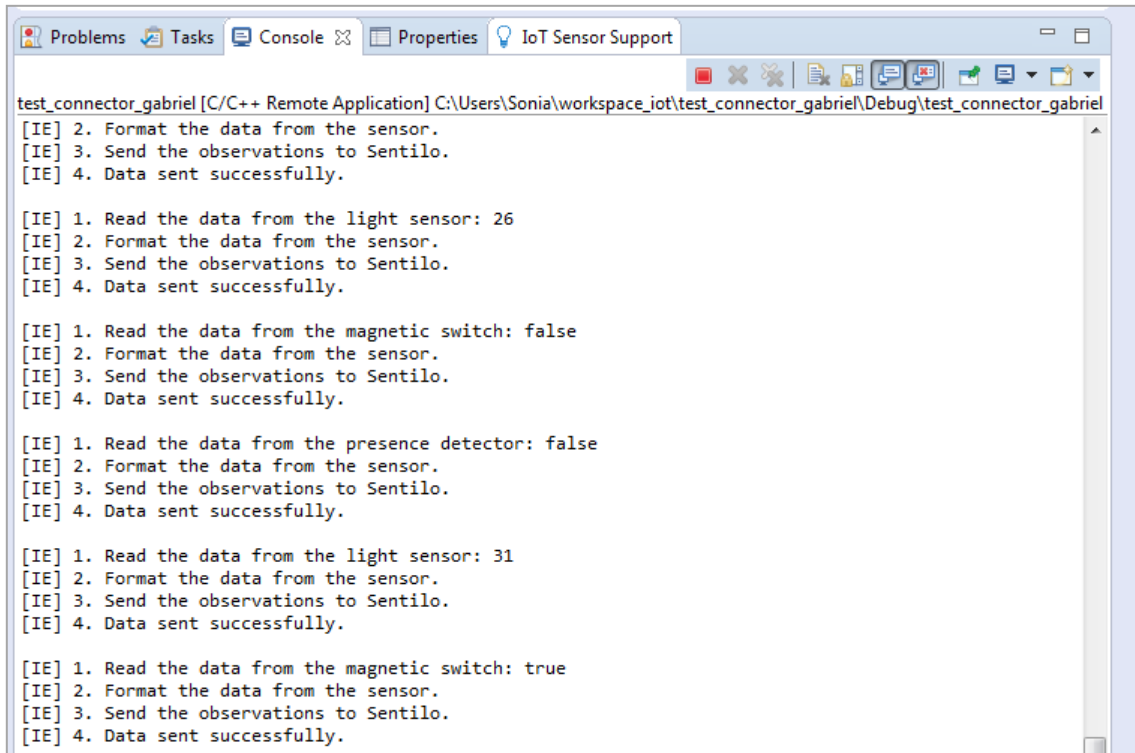


Figure 44: Intel Edison integration with a sensor

Running the Intel Edison connector to Sentilo.



```

test_connector_gabriel [C/C++ Remote Application] C:\Users\Sonia\workspace_iot\test_connector_gabriel\Debug\test_connector_gabriel
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.

[IE] 1. Read the data from the light sensor: 26
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.

[IE] 1. Read the data from the magnetic switch: false
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.

[IE] 1. Read the data from the presence detector: false
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.

[IE] 1. Read the data from the light sensor: 31
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.

[IE] 1. Read the data from the magnetic switch: true
[IE] 2. Format the data from the sensor.
[IE] 3. Send the observations to Sentilo.
[IE] 4. Data sent successfully.
    
```

Figure 45: Intel Edison connector integration

Similarly to the Arduino, the system will print the different steps.

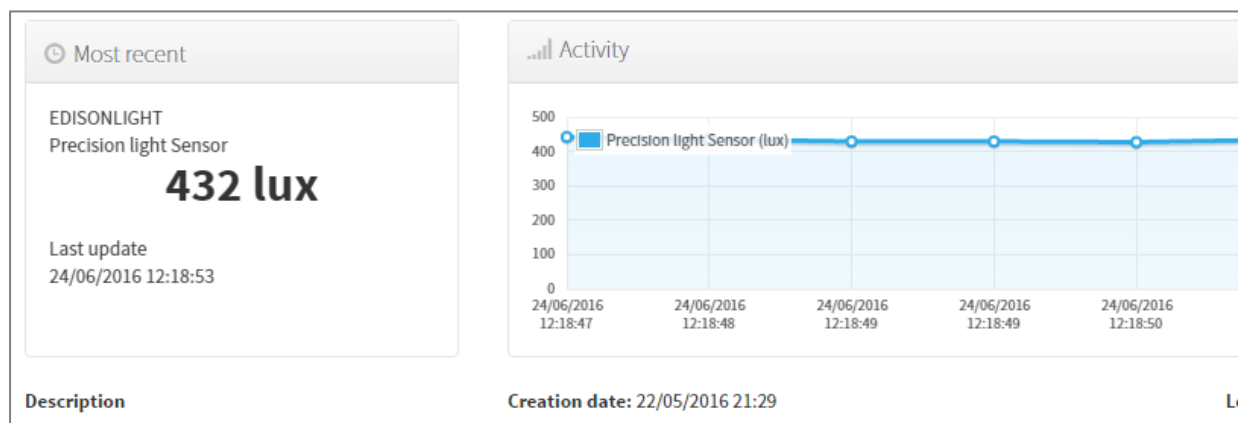


Figure 46: Intel Edison integration results as shown on SCW

Raspberry Pi integration

Integration of the Raspberry Pi 2 with an analog sensor by using the microchip MCP3002.

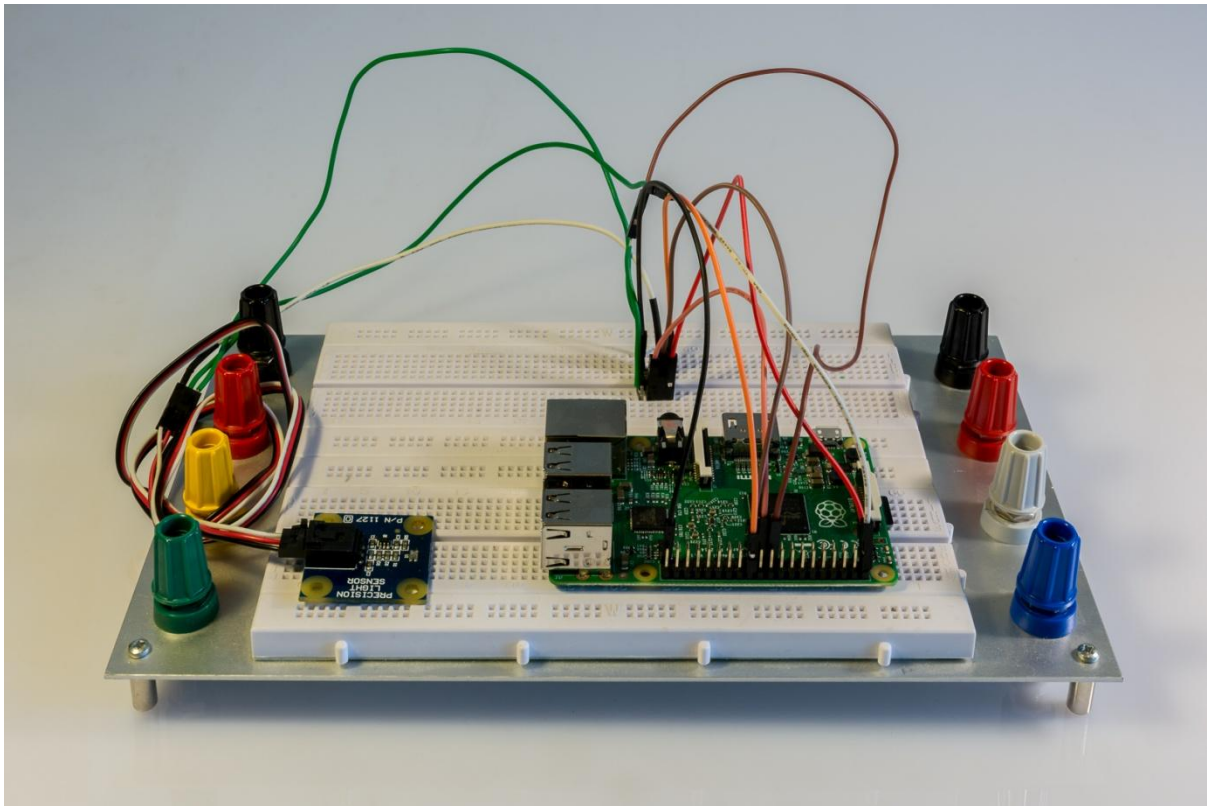


Figure 47: Raspberry Pi integration with a sensor

Running the Raspberry Pi connector to Sentilo.

```
root@wng-raspberry: /home/wng/Documents/ctests
File Edit View Search Terminal Help
root@wng-raspberry: /home/wng/Documents/ctests# ./connector_raspberry
[RP] 1. Read the data from the light sensor: 35
[RP] 2. Format the data from the sensor.
[RP] 3. Send the observations to Sentilo.
[RP] 4. Data sent successfully.

[RP] 1. Read the data from the magnetic switch: true
[RP] 2. Format the data from the sensor.
[RP] 3. Send the observations to Sentilo.
[RP] 4. Data sent successfully.

[RP] 1. Read the data from the light sensor: 36
[RP] 2. Format the data from the sensor.
[RP] 3. Send the observations to Sentilo.
[RP] 4. Data sent successfully.

[RP] 1. Read the data from the magnetic switch: true
[RP] 2. Format the data from the sensor.
[RP] 3. Send the observations to Sentilo.
[RP] 4. Data sent successfully.

[RP] 1. Read the data from the light sensor: 32
[RP] 2. Format the data from the sensor.
```

Figure 48: Raspberry Pi connector integration

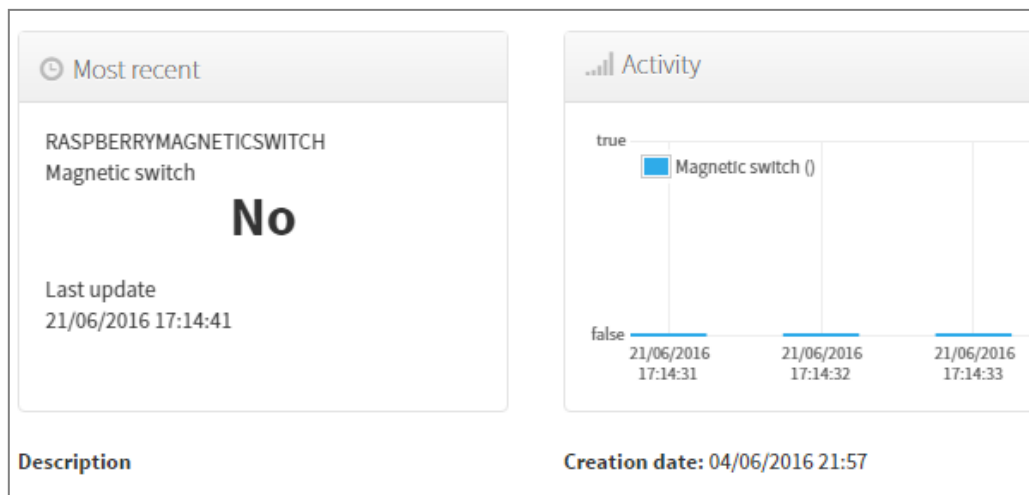


Figure 49: Raspberry Pi integration results as shown on SCW

Global connector testing

```

Problems Tasks Console Properties
<terminated> (exit value: 0) connector_general2.exe [C/C++ Application] C:\Users\Sonia\workspace_c\connector_general2\Debu
Welcome to SmartCampus Global Connector. The connector can be generated for the following platform
1. Arduino
2. Intel Edison
3. Raspberry Pi
Please select an option: 1

You have selected 1

Please select the wanted sensors, one at a time.
1. Temperature sensor [OFF]
2. Magnetic switch [OFF]
3. Precision light sensor [OFF]
4. Presence detector [OFF]
5. Force sensor [OFF]
6. Finish and print the code.
1

Please select the wanted sensors, one at a time.
1. Temperature sensor [ON]
2. Magnetic switch [OFF]
3. Precision light sensor [OFF]
4. Presence detector [OFF]
5. Force sensor [OFF]
6. Finish and print the code.
4

Please select the wanted sensors, one at a time.
1. Temperature sensor [ON]
2. Magnetic switch [OFF]
3. Precision light sensor [OFF]
4. Presence detector [ON]
5. Force sensor [OFF]
6. Finish and print the code.
6

Program finished. You may find the connector source file on the same folder.

```

Figure 50: Console view of the Global Connector

```
connector_arduino: Bloc de notas
Archivo Edición Formato Ver Ayuda
int pin_magnetic_switch = 2;
int pin_sensor_temp = A2;
int pin_presence_detector = 3;
int pin_sensor_force = A3;
/*Active sensors*/
/*May be activated and deactivated when necessary*/
const bool light_active = true;
const bool magnetic_active = false;
const bool temperature_active = true;
const bool presence_active = false;
const bool force_active = false;

/*Declaration of constants related to the connection*/
Sentiloclient sentiloclient = Sentiloclient(ip, port);
```

Figure 51: Results of the Global Connector integration

Zolertia on Linux PC

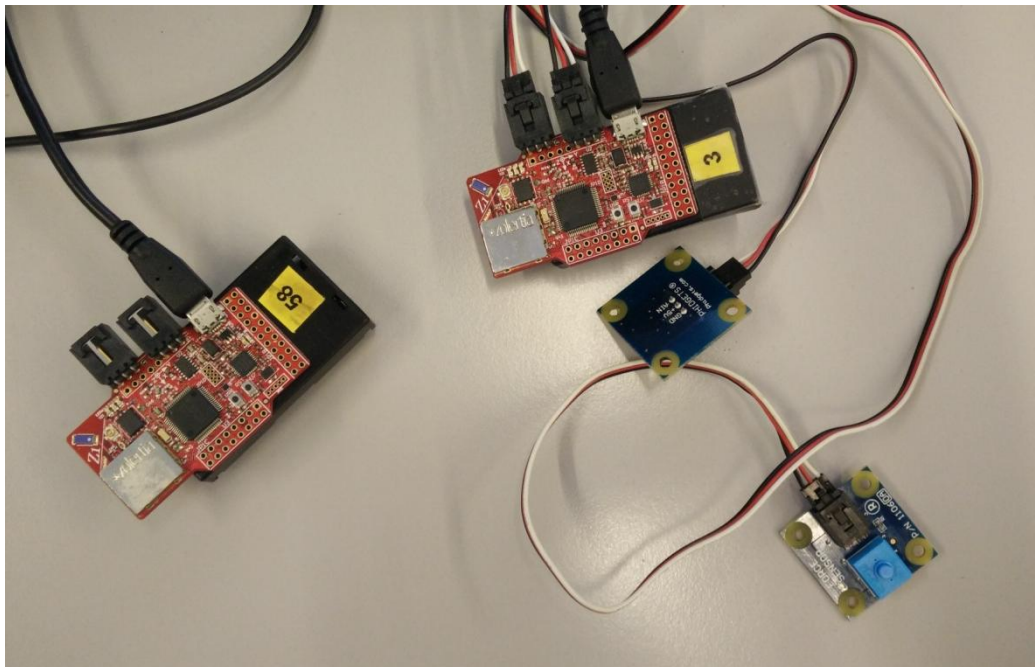


Figure 52: Integration of Zolertia Z1 nodes

```
netstat -nr | awk '{ if ($2 == "tun0") print "route delete -net \"$1; }' | sh

root@wng-lab:/home/wng/z1-apps/coap/sentilo/border-router# make connect-router
using saved target 'z1'
sudo ../../../../contiki-2.7/tools/tunslip6 aaaa::1/64
*****SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet 'hostname' up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Rime started with address 193.12.0.0.0.0.58
MAC c1:0c:00:00:00:00:3a Contiki 2.7 started. Node id is set to 58.
CSMA nullrdc, channel check rate 128 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:003a
Starting 'Border router process' 'Web server'
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
aaaa::c30c:0:0:3a
fe80::c30c:0:0:3a
```

Figure 53: Border Router running log

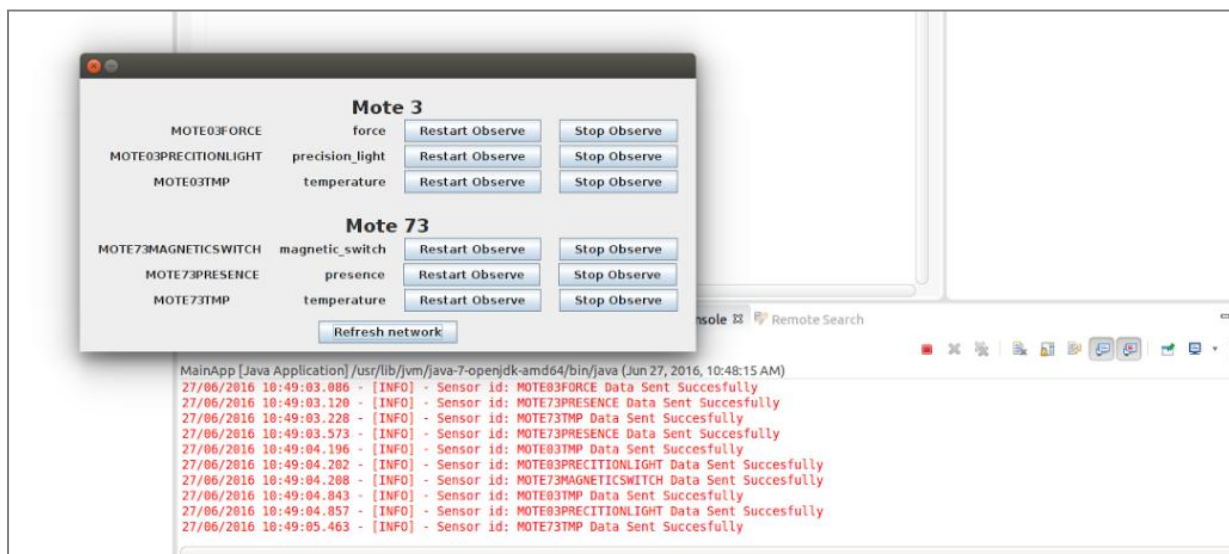


Figure 54: Connector's MainApp GUI

Zolertia Z1 on Raspberry Pi

As all Z1 modules, they are connected to power through USB. The only requirement is that the Border Router be connected to the platform, in this case Raspberry Pi. Other COAP Server modules may be connected to this platform, another one, or regular power supply. As for sensors, any sensor that works with Z1 modules can be connected.

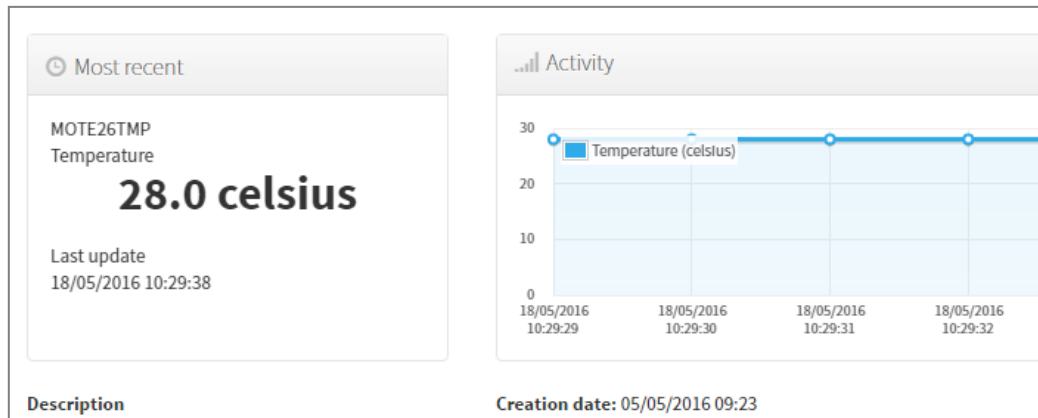


Figure 55: Raspberry Pi integration results as shown on SCW

Sentilo server

```

Border Router x Sentilo x v
2016-06-27 08:51:10,684 [ThreadPool-14] DEBUG org.sentilo.platform.server.handler.AbstractHandler -
Entity source: SG
Service: /data
Resource: path: SG/MOTE73PRESENCE
2016-06-27 08:51:10,684 [ThreadPool-14] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Default charset
2016-06-27 08:51:10,684 [ThreadPool-14] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Entity Content-
2016-06-27 08:51:10,686 [ThreadPool-14] DEBUG org.sentilo.platform.service.impl.DataServiceImpl - Registered in Redis obs
SG
2016-06-27 08:51:10,686 [ThreadPool-14] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - New http respon
2016-06-27 08:51:10,686 [ThreadPool-14] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Status: 200
2016-06-27 08:51:10,688 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Content-Type :
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Parsed Content-
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - New http PUT re
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Content-Type: a
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.auth.impl.AuthorizationServiceImpl - Init check
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.auth.impl.AuthorizationServiceImpl - Finished c
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Looking handler
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.handler.AbstractHandler - Manage PUT request
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.handler.impl.DataHandler - Executing data PUT r
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.handler.AbstractHandler -
Entity source: SG
Service: /data
Resource: path: SG/MOTE73MAGNETICSWITCH
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Default charset
2016-06-27 08:51:10,689 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Entity Content-
2016-06-27 08:51:10,690 [ThreadPool-16] DEBUG org.sentilo.platform.service.impl.DataServiceImpl - Registered in Redis obs
vider SG
2016-06-27 08:51:10,690 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - New http respon
2016-06-27 08:51:10,690 [ThreadPool-16] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Status: 200
2016-06-27 08:51:12,232 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Content-Type :
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Parsed Content-
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - New http PUT re
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Content-Type: a
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - extractHeader:
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.auth.impl.AuthorizationServiceImpl - Init check
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.auth.impl.AuthorizationServiceImpl - Finished c
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Looking handler
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.handler.AbstractHandler - Manage PUT request
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.handler.impl.DataHandler - Executing data PUT r
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.handler.AbstractHandler -
Entity source: SG
Service: /data
Resource: path: SG/MOTE73TMP
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Default charset
2016-06-27 08:51:12,233 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Entity Content-
2016-06-27 08:51:12,235 [ThreadPool-17] DEBUG org.sentilo.platform.service.impl.DataServiceImpl - Registered in Redis obs
2016-06-27 08:51:12,236 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - New http respon
2016-06-27 08:51:12,236 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Status: 200
2016-06-27 08:51:12,236 [ThreadPool-17] DEBUG org.sentilo.platform.server.request.SentiloRequestHandler - Content-Type :

```

Figure 56: Sentilo server running log

Client Application

The client application has a main menu, where a monitoring option may be chosen.

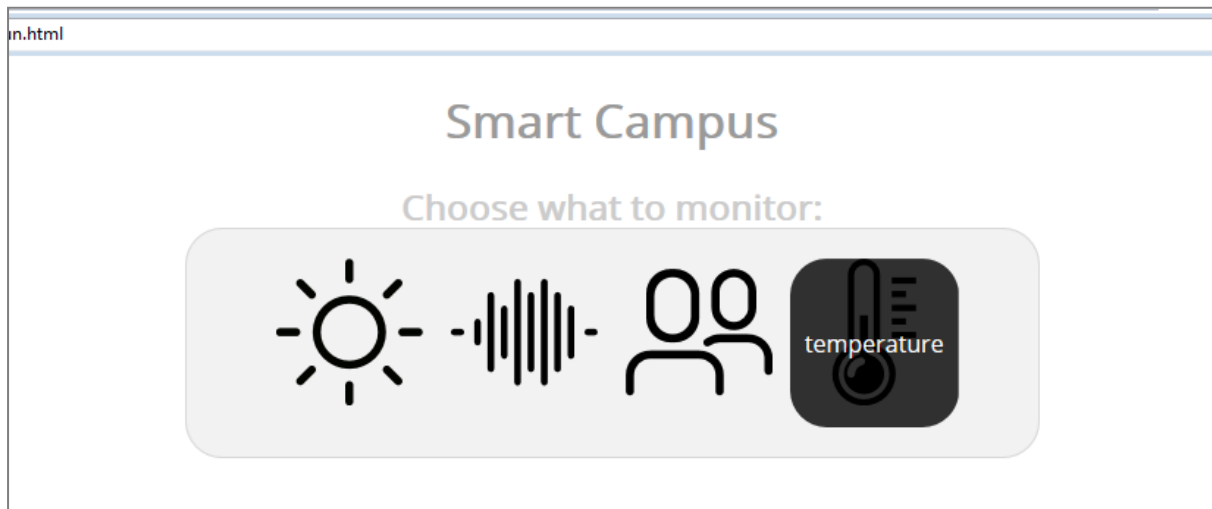


Figure 57: Web app main menu

Then, the desired space may be monitored.

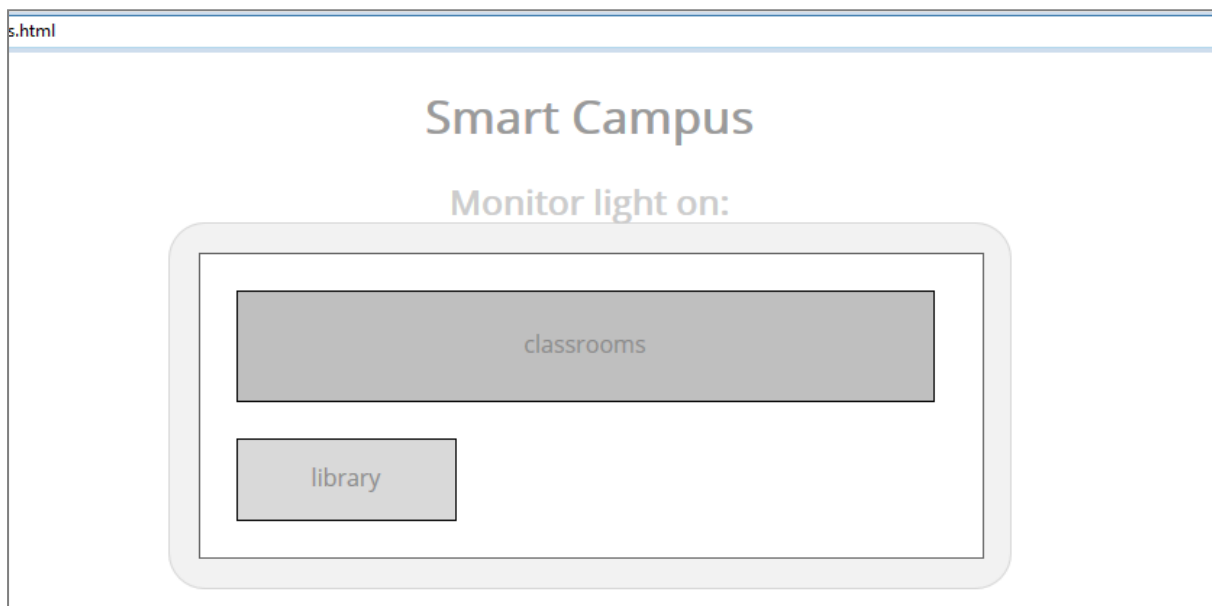


Figure 58: Web app spaces view

Finally, a floor plan with the results displayed as color code.

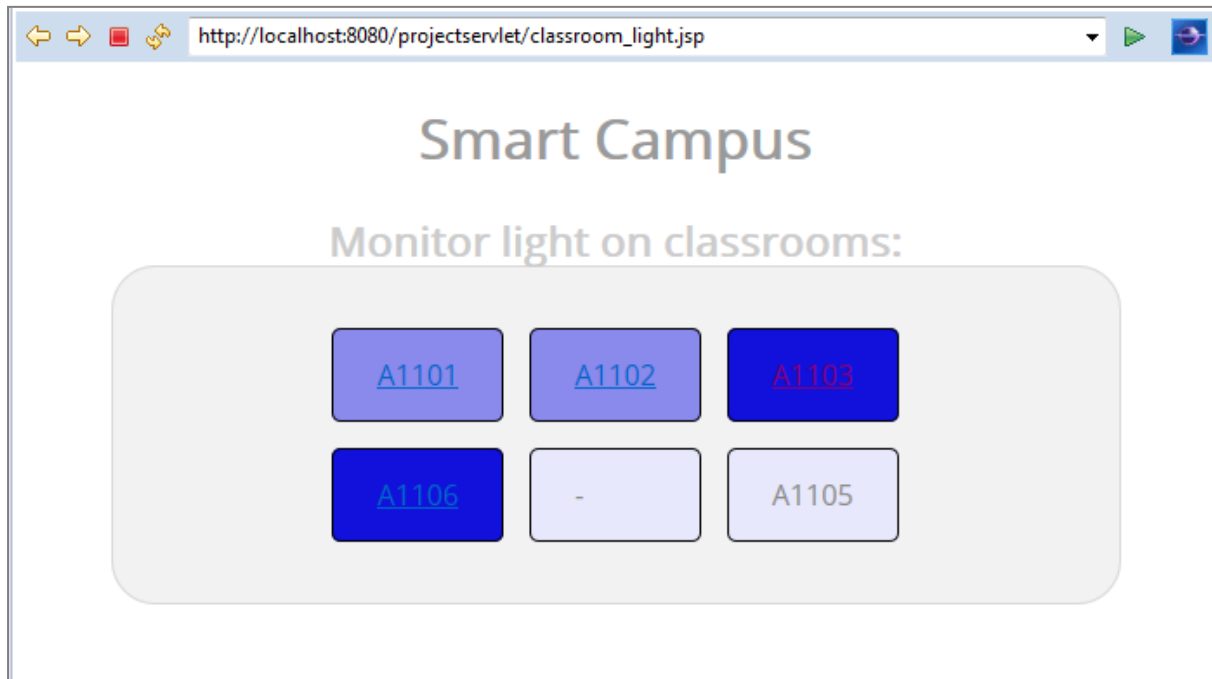


Figure 59: Web app floorplan with results

And when a particular room is chose, a graphic is displayed.

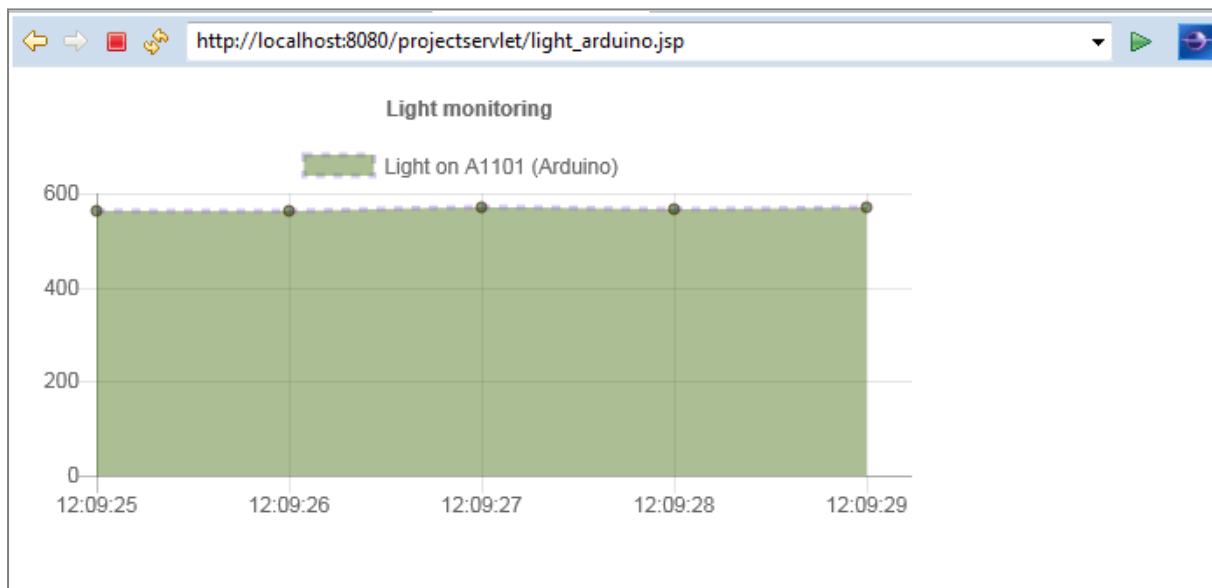


Figure 60: Web app results graph

With the title indicating which of the platforms has been used to simulate the deployment of the system in that space.

9. Budget

This section explains the budget for the project.

First, the overall materials that have been used in this project.

Research Equipment	Units	Cost per unit (€)	Cost (€)
Desktop computer	1	700	700
Desktop monitor	2	100	200
Keyboard	2	15	30
Mouse	2	10	20
Research Materials	Units	Cost per unit (€)	Cost (€)
Arduino Uno R3	1	20	20
Intel Edison Arduino Kit	1	100	100
Raspberry Pi 2	1	42	42
Zolertia Z1	5	95	475
Force sensor	2	10	20
Magnetic switch	1	5	5
Precision light sensor	1	5	5
Motion sensor (presence detector)	1	45	45
ADC microchip MCP3002	1	2	2
Ethernet Shield	1	15	15
Electronic cables			5
Subtotal			1684

Table 18: Budget of materials

Where Research Equipment is the equipment available in the research laboratory, and the Research Materials are those materials particular for this project.

	Hours	€/h	Total (€)
Research hours	420	8	3360
Subtotal			3360

Table 19: Budget of work

Approximating the project duration at 21 weeks.

So the total budget is as follows.

Total (€)	5044
-----------	------

Table 20: Budget total

10. Conclusions

This project has studied a complete IoT system from end to end as an application of the Smart City ideal to a Smart Campus. From sensors, to platform, to server, and to applications. The main objectives were the implementation of the entire system with the platforms Arduino, Intel Edison, Raspberry Pi and Z1; the implementation of the Sentilo server; the development of a multi-platform connector to be used on all platforms; the development of a client application; and the documentation of the framework concerning all platforms.

All the platforms have been able to be integrated with sensors, and implemented as providers successfully.

The server that has been used, Sentilo, has proved to be a very good choice. First, for being open source and allowing any user who wishes to, to create their own server. And also because of its accessible data through HTTP, which was a great opportunity to use with the application.

For the multi-platform and global connector, even though the initial idea of using one connector program for all platforms was found to be not possible because of programming incompatibilities, there was a way to accomplish the same end but from a higher layer, a code generator or printer. Any user would be able to run that program, choose any of the platforms and their sensors, and copy it to their platform to compile and run the connector. Additionally, the code has been designed so that any new platform or sensor may be added at a later time, to include more options.

And as for the application, the best option when it came to accessing the data through HTTP was using the JavaServer Pages technology. These were used to build an application that would not only display or visualize the data that had been generated in the first end of the system, but to be able to use and rely as a service.

The documentation has also been completed, and can be found in this Final Report's technology and Appendices sections.

This project was particularly challenging because there was need to study not only one platform's workings, but four of them.

With this project concluded, this is now a good starting point for the Smart Campus; to scale it into a bigger, more platform diverse deployment.

11. Future lines of work

Some of the future lines of work may be including more platforms in the global connector. Other platforms that have not been studied in this project but that can be used in IoT projects are:

- Pinoccio
- UD00
- PanStamp Minibat
- XinoRF
- BeagleBone Black
- CubieBoard
- Nanode
- WelO
- OpenPicus Flyport WiFi

All of these platforms have I/O pins and have Internet connectivity via Ethernet or WiFi. Additionally, the connectors could be tested with newer or older versions of the platforms to check their compatibility. And, on another hand, the Arduino individual connector could add the ability to connect through WiFi in addition to Ethernet.

For the client application, there could also be a native mobile app for a more specific functionality.

Other future lines of work would be exploring the possible utilization of Dockers in order to have a development environment that is independent of the platform. And, for the server, the utilization of resilient structures based on Cloud in order to guarantee the collecting of information.

Bibliography

- [1] M. Vallianatos, "Uncovering the Early History of "Big Data" and the "Smart City" in Los Angeles," 06 2015. [Online]. Available: <http://www.boomcalifornia.com/2015/06/uncovering-the-early-history-of-big-data-and-the-smart-city-in-la/>. [Accessed 19 June 2016].
- [2] T. Falk, "The origins of smart city technology," ZDNet, 27 02 2012. [Online]. Available: <http://www.zdnet.com/article/the-origins-of-smart-city-technology/>. [Accessed 19 May 2016].
- [3] A. Puri, "What are smart cities?," The Hindu, 15 08 2014. [Online]. Available: <http://www.thehindu.com/features/homes-and-gardens/green-living/what-are-smart-cities/article6321332.ece>. [Accessed 2016 May 19].
- [4] Arduino, "Arduino Board 101," Arduino, [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoard101>. [Accessed 1 May 2016].
- [5] Arduino, "Digital Pins," Arduino, [Online]. Available: <https://www.arduino.cc/en/Tutorial/DigitalPins>. [Accessed 1 May 2016].
- [6] R. Pi, "Raspberry Pi 2 Model B," Raspberry Pi, [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. [Accessed 13 April 2016].
- [7] Microchip, "MCP3002," Microchip, [Online]. Available: <http://www.microchip.com/wwwproducts/en/MCP3002>. [Accessed 15 April 2016].
- [8] G. Henderson, "The Gertboard for the Raspberry Pi," Gordons Projects, [Online]. Available: <https://projects.drogon.net/raspberry-pi/gertboard/>. [Accessed 15 April 2016].
- [9] G. Henderson, "Understanding SPI on the Raspberry Pi," Gordons Projects, 27 August 2012. [Online]. Available: <https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/>. [Accessed 15 April 2016].
- [10] M. McCauley, "C library for Broadcom BCM 2835 as used in Raspberry Pi," bcm2835, [Online]. Available: <http://www.airspayce.com/mikem/bcm2835/index.html>. [Accessed 15 April 2016].

- [11] R. M. Phil Howard, "Pinout," Raspberry Pinout, [Online]. Available: <http://pinout.xyz/>. [Accessed 16 May 2016].
- [12] Arduino, "Arduino/Processing Language Comparison," Arduino, [Online]. Available: <https://www.arduino.cc/en/Reference/Comparison>. [Accessed 24 May 2016].
- [13] Arduino, "begin()," Arduino, [Online]. Available: <https://www.arduino.cc/en/Serial/Begin>. [Accessed 7 June 2016].
- [14] J. T. p. J. T., "INTERNET OF THINGS: USING MRAA TO ABSTRACT PLATFORM I/O CAPABILITIES," Intel Developer Zone, 29 April 2015. [Online]. Available: <https://software.intel.com/en-us/articles/internet-of-things-using-mraa-to-abstract-platform-io-capabilities>. [Accessed 9 June 2016].
- [15] IanH, "RPi GPIO Code Samples," eLinux, 27 March 2016. [Online]. Available: http://elinux.org/RPi_GPIO_Code_Samples. [Accessed 20 April 2016].
- [16] J. T. p. J. T., "Intel Developer Zone," Intel Edison, 29 04 2015. [Online]. Available: https://software.intel.com/en-us/articles/internet-of-things-using-mraa-to-abstract-platform-io-capabilities#_Toc2.2. [Accessed 04 06 2016].
- [17] arfoll, "intel-iot-devkit Repository," 24 03 2016. [Online]. Available: <https://github.com/intel-iot-devkit/mraa/blob/master/docs/edison.md>. [Accessed 04 06 2016].

Appendices

I Sentilo

To run the Sentilo server, the system needs to install the following tools and configure them appropriately.

- Git
- Maven2
- Redis
- MySQL
- Tomcat

First, install them.

```
sudo apt-get install git maven2 redis-server mongodb mysql-server tomcat7
```

If MySQL is being installed for the first time, a new username and password when prompted.

The source code for the server is obtained from its original repository.

```
git clone https://github.com/sentilo/sentilo.git sentilo
```

And built using maven in order to create the executables.

```
cd sentilo  
  
mvn clean install  
  
mvn eclipse:clean eclipse:eclipse
```

Configure Redis

Configure Redis to enable and edit the password.

```
sudo gedit /etc/redis/redis.conf
```

And modify the line

```
#requirepass pw → requirepass new_pw
```

With the desired password.

Configure MongoDB

Configure MongoDB to enable the authentication.

```
sudo gedit /etc/mongodb.conf
```

And modify the line

```
#Turn on/off security. Off is currently the default
#noauth = true
#auth = true
```

to

```
#Turn on/off security. Off is currently the default
#noauth = true
auth = true
```

Save and close the edited file. Afterwards, create the sentilo database, a user and password to access it.

```
mongo
use sentilo
db.addUser("sentilo","sentilo")
exit
```

After the database has been created, we can add the data from the Sentilo server into it.

```
cd /home/user_name/sentilo/scripts/mongodb
mongo -u sentilo -p sentilo sentilo init_test_data.js
```

Configure MySQL

To configure MySQL, create the database 'sentilo' and the user with its password.

```
mysql -u root -p
[insert password]
CREATE USER 'sentilo_user'@'localhost' IDENTIFIED BY 'sentilo_pwd';
CREATE DATABASE sentilo;
GRANT ALL ON sentilo.* TO 'sentilo_user'@'localhost';
flush privileges;
exit
```

In the event that MySQL was already installed

Create the tables that are needed for Sentilo in the database. There is already a file with the necessary queries to create them.

```
mysql --user=sentilo_user --password=sentilo_pwd sentilo < sentilo-agent-  
relational/src/main/resources/bd/agent_mysql.sql
```

Configure Tomcat7

To configure Tomcat7, copy the .war file in Sentilo's Catalog Web.

```
sudo cp ~/sentilo/sentilo-catalog-web/target/sentilo-catalog-web.war  
/var/lib/tomcat7/webapps  
  
sudo service tomcat7 restart
```

Sentilo server launch

To start sentilo, the system has to launch 4 binaries.

First, create the folders.

```
mkdir /opt/sentilo-server  
  
mkdir /opt/sentilo-agent-alert  
  
mkdir /opt/sentilo-agent-relational  
  
mkdir /opt/sentilo-agent-location-updater
```

If it is the newer version of Sentilo, the executables have to be created by manually building the project.

```
cd sentilo/scripts  
  
sudo chmod +x buildSentilo.sh  
  
./buildSentilo.sh
```

Copy all the files into the folders.

```
mv sentilo-platform/sentilo-platform-server/target/appassembler/*  
/opt/sentilo-server  
  
mv sentilo-agent-alert/target/appassembler/* /opt/sentilo-agent-alert
```

```
mv sentilo-agent-relational/target/appassembler/* /opt/sentilo-agent-  
relational
```

```
mv sentilo-agent-location-updater/target/appassembler/* /opt/sentilo-agent-  
location-updater
```

II Z1: Contiki OS 2.7

First, install the Contiki OS for the Zolertia nodes by downloading from:

<https://codeload.github.com/contiki-os/contiki/zip/2.7>

Unzip the file and place the folder in the working directory. This directory needs to be accessed when compiling, uploading, or running a program on a node.

There are additional building tools that need to be installed in order to compile the programs.

- msp430 toolchain, to compile and upload to the Z1 motes.
- Java Development Kit (1.7 recommended for this project).
- libncurses5, necessary library for the native examples of Contiki.
- Ant, to use the cooja simulator.

Install all the tools with their required dependencies.

```
sudo apt-get install build-essential binutils-msp430 gcc-msp430 msp430-libc  
msp430mcu  
  
mspdebug openjdk-7-jdk libncurses5-dev ant
```

There is, however, a bug in the msp430-gcc 4.6.3 regarding the serial communication that affects the dump prints from the mote. Since this is the version that is downloaded automatically from the repository, it is necessary to manually download the newer version (4.7) from:

<http://sourceforge.net/projects/zolertia/files/Toolchain/msp430-47.tar.gz>

Unzip it, and move it to the /opt directory.

```
sudo cp -r msp430-47 /opt
```

Add the folder to the environment variable PATH for the local user.

```
echo "PATH=/opt/msp430-47/bin:$PATH" >> ~/.bashrc
```

Finally, modify it for the root user.

```
sudo visudo
```

Modify the following line:

```
secure_path="/usr/local/sbin:/usr...
```

```
secure_path="/opt/msp430-47/bin:usr/local/sbin...
```

Close the editor and save the modified file.

Now the OS and its building tools are installed, but x64 systems require an additional step because the compiler is designed to compile in 32 bits.

Finally, if the system is x64, in order to compile correctly.

```
sudo -i

cd /etc/apt/sources.list.d

echo "deb http://old-releases.ubuntu.com/ubuntu/ raring main restricted
universe multiverse" > ia32-libs-raring.list

apt-get update

apt-get install ia32-libs
```

This final step may take some time.

III Z1: COAP elements

First of all, the platform has to be connected on the computer through USB, and this computer has to have the Contiki OS folders installed.

To compile a program, the 'make' order is used. This order has to be given from the folder where the app is located.

```
app-location$ make TARGET=z1 appName
```

To compile and upload a program,

```
app-location$ make TARGET=z1 appName.upload
```

Please take notice that this order can only be given as is if the Z1 connection to the computer corresponds to /dev/ttyUSB0. If it does not, it has to be specified.

```
app-location$ make TARGET=z1 MOTES=/dev/ttyUSB1 appName.upload
```

After the application has been uploaded, the console can run and check the results with the order

```
app-location$ make login
```

Border Router

To program a Z1 to act as Border Router for the Z1 network, first go to the border-router folder.

```
border-router$ make TARGET=z1 border-router.upload
```

Because of the tunslip settings, the Border Router has to be uploaded and launched as /ttyUSB0.

COAP Servers

The COAP Servers are the Z1 nodes that will have the sensors connected and will transmit the data to the Border Router. For every COAP Server, they have to have specified which sensors are attached to them.

The configuration file is the 'general-server.c' file on the coap-servers folder. Both internal sensors and external (phidget) sensors can be configured.

The header file contains

```
[...]
#include "sys/node-id.h"

/*****Z1 internal*****/
/*****Sensors resources*****/

/* Temperature sensor */
#define RESOURCE_TEMPERATURE_SENSOR      1
#define TEMP_READ_INTERVAL               CLOCK_SECOND

/* Accelerometer sensor */
#define RESOURCE_ACCELEROMETER_SENSOR    0
#define ACCL_READ_INTERVAL               CLOCK_SECOND

/*****
//Define if an external Phidget is
connected.

*****/

/*****Z1 external*****/
/*****Sensors resources*****/
#define RESOURCE_LIGHT_SENSOR            0
#define LIGHT_SENSOR_PORT                5 // 3 or 5 if
connected to 6.7(3V) or 6.0(5V)
#define LIGHT_READ_INTERVAL              CLOCK_SECOND
```

```
#define RESOURCE_PRECISION_LIGHT_SENSOR 1
#define PRECISION_LIGHT_SENSOR_PORT 5 // 3 or 5 if
connected to 6.7(3V) or 6.0(5V)
#define PRECISION_LIGHT_READ_INTERVAL CLOCK_SECOND

#define RESOURCE_FORCE_SENSOR 1
#define FORCE_SENSOR_PORT 3 // 3 or 5 if
connected to 6.7(3V) or 6.0(5V)
#define FORCE_READ_INTERVAL CLOCK_SECOND

#define RESOURCE_PIR_SENSOR 0
#define PIR_SENSOR_PORT 5 // 3 or 5 if
connected to 6.7(3V) or 6.0(5V)
#define PIR_THRESHOLD 1000
#define PIR_READ_INTERVAL CLOCK_SECOND

#define RESOURCE_MAGNETIC_SWITCH 0
#define MAGNETIC_SWITCH_PORT 3 // 3 or 5 if
connected to 6.7(3V) or 6.0(5V)
#define MAGNETIC_SWITCH_THRESHOLD 1000
#define MAGNETIC_SWITCH_READ_INTERVAL CLOCK_SECOND

#define RESOURCE_DISTANCE 0
#define DISTANCE_READ_INTERVAL CLOCK_SECOND
```

All sensors can be activated or deactivated by setting the value of “#define RESOURCE_TEMPERATURE_SENSOR” to 1 or 0, respectively. The voltage or port they will be connected to also has to be specified.

When this is set to 1, when uploading this server, the file containing the program for that sensor will also be uploaded, as seen in:

```
#if RESOURCE_TEMPERATURE_SENSOR
    #include "resources/temp.c"
#endif
```

All the sensor resources, which are the code to modify and handle the data, are located on the 'resources' folder inside the COAP Server folder.

To compile and upload,

```
coap-server$ make TARGET=z1 MOTES=/dev/ttyUSB1 general-server.upload
```

Running the Z1 network

1. Upload the programs to the Border Router and the COAP Servers.
2. Connect the Border Router to the host computer, making sure it is /ttyUSB0.
3. Connect the COAP Servers to a power source. May be the host computer or another power source.
4. Go to the Border Router folder and input the order `make connect-router`.
 - a. Sometimes the BR needs to be reset, press the RST button on the platform if necessary.
5. On the folder where the connector is located, configure the 'app.properties' file.
6. On the same folder, run the connector with the command `java -jar wsn.jar`.

IV Raspberry Pi

Enabling SPI

SPI (Serial Peripheral Interface) is disabled on the Raspberry Pi 2. To enable it in order to work with analog input, follow these steps:

1. Input command `sudo nano /boot/config.txt`
2. Add this line at the end: `dtoverlay=spi=on`
3. Close and save the file
4. Reboot Raspberry Pi

Installing bcm2835 library

Download the latest version of the bcm2835 library (in .tar.gz format):

<http://www.airspayce.com/mikem/bcm2835/>

Place it on a folder of your choice and on the console,

```
tar zxvf bcm2835-1.50.tar.gz
cd bcm2835-1.xx
./configure
make
sudo make check
sudo make install
```

Replace the library name depending on the release as needed.

Compiling

The connector has been programmed with C++, so a C/C++ compiler is needed. The following order can easily be commanded to see if there is said compiler installed on the platform.

```
gcc -v
```

To compile with the flag as required by the bcm2835 library,

```
gcc -o program program.c -l bcm2835
```

Or as `g++` and `‘.cpp’` if it is C++.

V Intel Edison and Arduino

Intel Edison

The IDE may be installed and the platform configured with the official [Getting Started Guide](#).

To run the connector, open the Eclipse IDE (IoT version). To create a new project, go to File – New - Create new Intel IoT project as C/C++.

Copy the connector code onto the .cpp file. And then connect the board to the computer, and run it with Run – Run as – Intel System Studio IoT Edition. This will automatically upload the program onto the board and start running it.

Arduino

The Arduino IDE may be downloaded in any OS.

Simply connect the Arduino to the computer, copy the code onto a new Sketch, and upload it into the platform.

Open a Serial Monitor window to view the debugging traces.

Glossary

Board: physical body of the device (single-board computer) that integrates the sensors and manipulates the data.

COAP server: in this project, a Z1 node which has the sensors and thus generates data and sends them to the Border Router.

GPIO: general purpose input/output, the software view of physical pins.

GUI: graphical user interface.

IoT: Internet of things, which follows the idea of everyday objects being able to connect to wifi to share their data.

JSON: a format for structuring data which includes key and value fields.

Mote: node of a Z1 network.

Pin: physical metal piece that connects to a circuit board.

Provider: the set of sensor and board (or component) as identified by Sentilo.

REST resource: any kind of information that can be named, which is used in REST.

Sensor: hardware device that generates digital or analogic data based on environment observations.

Server: usually a computer or program that can provide a service to an outer element, the Client.

Smart City: ide of developing cities by using information and communication technologies.

URI: uniform resource identifier, used to identify a Sentilo element's location.