Faculty of Sciences and Technology
Department of Informatics Engineering

IoT Student Advisor and Best Lifestyle Analyzer (ISABELA)

IoT Management

Inês Pereira de Azevedo Mota

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Professor Jorge Sá Silva and Professor Fernando Boavida and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

January 2019

This work was developed in colaboration with:

Center of Informatics and Systems

Department of Informatics Engineering
University of Coimbra

Agradecimentos

Começo por agradecer ao Prof. Dr. Jorge Sá Silva e ao Prof. Dr. Fernando Boavida pela orientação, compreensão e disponibilidade demonstradas ao longo destes meses de trabalho.

Aos meus pais, pelo amor, preocupação e ajuda. Por me acompanharem sempre e me darem colo quando preciso.

À minha irmã, pela companhia e apoio. Por me dar a mão e prescindir de si para primeiro garantir que estou bem.

À minha avó, por rezar sempre pelo meu sucesso. Aos meus tios, primos e padrinhos pelo interesse e apoio que sempre me demonstram. Ao meu afilhado, pela sua chegada que alegrou este processo. Aos meus avós e tio, que, apesar de já não estarem, estão sempre comigo.

À Carolina, à Cris, à Joana, à Maria, à Melanie e à Paula, por compreenderem a minha ausência e por me receberem de volta com palavras encorajadoras e carinho.

Ao André, ao António e ao Carlos, por todas as horas de trabalho e de lazer que passámos juntos. Levo-vos comigo para a vida.

Às deusas do "Concílio" e ao Wilson, pela amizade.

Finalmente, ao Armando, ao Duarte, ao Jorge, ao Marcelo, ao Oswaldo e à Soraya por me terem recebido e acolhido e por despenderem do seu tempo para se preocuparem por mim e me ajudarem sempre que precisei.

Abstract

This report is focused on the work made throughout the internship. The work carried consisted on the study of the technologies and protocols underlying the project, the study of IoT Student Advisor and Best Lifestyle Analyzer (ISABELA) and some similar case studies, the implementation of a proximity module through bluetooth operations and the study of Network Management Protocols and integration of the Lightweight machine-to-machine (LWM2M) protocol, that manages the various devices that are part of the project.

The objective of this thesis is to continue and expand the work done on ISABELA so far. Therefore, one of the focal points of this thesis is the development of a proximity module developed to infer about sociability. The module is based on bluetooth operations to discover nearby devices, at all times, and know the amount of time a participant spends with the same people, amongst random people or alone. The platform was tested in a real-life context, with students at Department of Engineering Informatics (DEI), Instituto Superior de Contabilidade e Administração de Coimbra (ISCAC) and Escuela Politécnica Nacional (EPN), in Ecuador. One of the main objectives was achieved, as the system informs the participant when it detects "bad behavior", through messages from the *ChatBot*. This module, also, served to better understand the platform and learn to work with its components. The other focal point of this thesis is the incorporation of a network management protocol to manage all the heterogeneous devices that complete ISABELA. The chosen protocol was LWM2M, a protocol design by Open Mobile Alliance SpecWorks for managing sensor networks and remote machine-to-machine devices. The integration of the protocol was made through two approaches: 1) Eclipse's *Leshan*, where the management protocol is parallel to the ISABELA system and 2) FIWARE's *LWM2M IoT Agent*, where the server is integrated on the project's Internet of Things (IoT) middleware (FIWARE).

Performances tests were made to evaluate the impact that the addition of the management protocol could provoke on the smartphone's battery use. The results were favorable, culminating in that the battery use is not significantly higher for the application running ISABELA and the LWM2M client compared to the application only running ISABELA.

Keywords

Internet of Things, Cyber-physical Systems, Human-in-the-Loop, Context Information, Lightweight machine-to-machine, Constrained Application Protocol (CoAP), IoT Agent.

This page is intentionally left blank.

Resumo

O presente relatório é focado no trabalho feito durante o estágio. Esse trabalho consistiu no estudo das tecnologias e protocolos subjacentes ao projeto, no estudo do ISABELA e alguns casos de estudo similares, na implementação de um módulo de proximidade através de operações de *bluetooth* e o estudo de alguns Protocolos de Gestão de Redes e a integração do protocolo LWM2M, que gere os diversos dispositivos que fazem parte do projeto.

O objetivo desta tese é continuar e expandir o trabalho feito, até agora, no ISABELA. Por isso, um dos pontos focais desta tese é um módulo de proximidade desenvolvido para inferir sobre sociabilidade. O módulo foi desenvolvido com base em operações de *bluetooth*, de modo a descobrir dispositivos nas proximidades e ficar a saber a quantidade de tempo que um estudante passa acompanhado das mesmas pessoas, no meio de multidões que não conhece ou sozinho. Esta plataforma foi testada num contexto real, por alunos do DEI, do ISCAC e de EPN, uma escola do Equador. Um dos principais objetivos foi alcançado, uma vez que o sistema avisa o estudante quando deteta comportamentos prejudicias, através de mensagens do *ChatBot*. Este módulo, também, serviu para conhecer melhor a plataforma e aprender a trabalhar com os seus componentes. O outro ponto focal desta tese é a integração de um protocolo de gestão de redes para gerir os dispositivos heterogéneos que compõem o sistema. O protocolo escolhido foi LWM2M, desenhado pela companhia *Open Mobile Alliance SpecWorks* para gerir redes de sensores e dispositivos remotos. Essa integração foi feita segundo duas abordagens: 1) através do projecto *Leshan* do Eclipse, na qual o protocolo de gestão é paralelo ao sistema ISABELA e 2) através do *Agente IoT para LWM2M* do FIWARE, na qual o servidor é integrado diretamente num módulo do FIWARE do nosso projeto.

Foram realizados testes de desempenho para avaliar o impacto que teria a adição do protocolo de gestão no consumo de bateria do *smartphone*. Os resultados foram favoráveis, culminando em que o consumo de bateria não é significativamente mais elevado para a aplicação que corre o ISABELA e o cliente LWM2M comparada com a aplicação que só corre o ISABELA.

Palavras-Chave

Internet das Coisas, Sistemas ciber-físicos, Human-in-the-Loop, Informação do Contexto, Lightweight machine-to-machine, CoAP, Agente IoT.

This page is intentionally left blank.

Contents

Acronyms

API Application Programming Interface.

CoAP Constrained Application Protocol.

CPS Cyber-physical Systems.

DEI Department of Engineering Informatics.

DTLS Datagram Transport Layer Security.

EPN Escuela Politécnica Nacional.

FI-PPP Future Internet Public Private Partnership.

GE Generic Enablers.

HiTL Human-in-the-Loop.

IETF Internet Engineering Task Force.

IoT Internet of Things.

ISABELA IoT Student Advisor and Best Lifestyle Analyzer.

ISCAC Instituto Superior de Contabilidade e Administração de Coimbra.

LWM2M Lightweight machine-to-machine.

NETCONF Network Configuration Protocol.

OS Operating System.

REST Representational State Transfer.

RFID Radio-Frequency Identification.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

YANG Yet Another Next Generation.

This page is intentionally left blank.

List of Figures

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In this chapter, the context of this Master thesis and its objectives are provided; some important concepts and some of the technologies that enabled the realization of the project are detailed and a brief explanation of the work planning and the methodology are presented.

1.1 Context

The term Internet of Things (IoT) emerged in 1999 and was proposed by Kevin Ashton. At first, it was about monitoring commercial products using Radio-Frequency Identification (RFID) tags, with minimal human intervention [1]. Nowadays, the core is the same but the technology evolved.

In 2018, there are, approximately, 23.14 billion connected devices and this number is expected to grow to, approximately, 75.44 billions, by 2025 [2]. We are surrounded by "smart" things that are, constantly, being equipped with new capabilities. They are connected to the Internet, are capable of sensing and can communicate between them and share the sensing data [3].

However, raw data is not enough to impact the way of life. We need to understand it. There is where Cyber-physical Systems (CPS) enter. CPS are systems that rely on working together in order to control the physical environment around them. They include sensors and actuators and when their finality is to be useful to the users, by considering their actions and emotions, we enter the realm of Human-in-the-Loop (HiTL) CPS. This kind of CPS are used to contribute for the betterment of the quality of life, as they acquire context information, process it and give advice on how to proceed [4].

In 2016, Portugal registered the 4th highest rate of college dropouts in the European Union, at 14%, among students between the ages of 18 and 24 [5]. Two of the most common causes for this rate are the difficulty of having to balance work and school and stress and depression [6]. Students are feeling overwhelmed by schoolwork and other commitments, therefore, are studying more and socializing less.

With the help of IoT and HiTL CPS, we can create more innovative teaching methods centered in understanding students realities and giving them feedback, by using their smartphones as a mean to collect information about their day-to-day activities and giving them informed advice on how to act.

1.2 Objectives

This thesis is inserted in a *Fundação para a Ciência e a Tecnologia* [7] project, which started in July of 2016. The project, IoT Student Advisor and Best Lifestyle Analyzer (ISABELA), is a case study and aims to infer about the impact on the academic performance of its participants and give advice to prevent negative outcomes, based on data collected through sensors. In order to accomplish that, a system able to collect data from the students habits, actions and emotions, process it and give feedback based on the results was created.

ISABELA's objectives were the creation of an IoT platform to communicate with different devices, store the information collected and protect that information, and the development of an Android application to retrieve data through the smartphone's sensors and send it to the aforementioned platform. It is through this application that we can provide relevant information and advice to the user.

The objective of this thesis is to continue and expand the work done on ISABELA so far. For that, this thesis is focused on:

1. The development of a sociability module based on proximity. Sociability is considered to have a great impact on students' lives and their academic performances. This new module was developed through bluetooth operations that enable the discovery of other devices nearby and allow us to infer on the sociability of the participant, because through it we can discover the amount of time the participant spends alone, with the same people or amongst random people. The module also served to better understand the platform and learn to work with its components.

2. The integration of a network management protocol that manages the heterogeneous devices that are part of the ISABELA system. A study of various protocols was made and the chosen one was Lightweight machine-to-machine (LWM2M), a protocol design by Open Mobile Alliance SpecWorks for managing sensor networks and remote machine-to-machine devices. Its integration was made through two approaches: 1) Eclipse's *Leshan*, where the management protocol is parallel to the ISABELA system and 2) FIWARE's *LWM2M IoT Agent*, where the server is integrated on the IoT middleware (FIWARE).

These two components, its development and specific functions will be explained in greater detail in chapter 4.

1.3 Report Structure

This report is divided in 6 major chapters: 1 Introduction, 2 Network Management Protocols, 3 ISABELA, 4 Development, 5 Performance Tests and Results and 6 Conclusions and Future Work.

The first chapter serves as an introduction to the project, where the context and objectives are explained. In this chapter, there is also a brief explanation of the work planning, the methodology and the technologies used throughout the project.

In the second chapter, there is a presentation and evaluation of the various network management protocols considered to manage the several devices that form part of the system.

In the third chapter, there is an overview of the ISABELA project, in which this thesis is inserted. A brief description is presented and some of its components are addressed in more depth because they are vital to the work of this thesis.

The fourth chapter presents the process of development of the components that make it possible to fulfill the two objectives of this thesis.

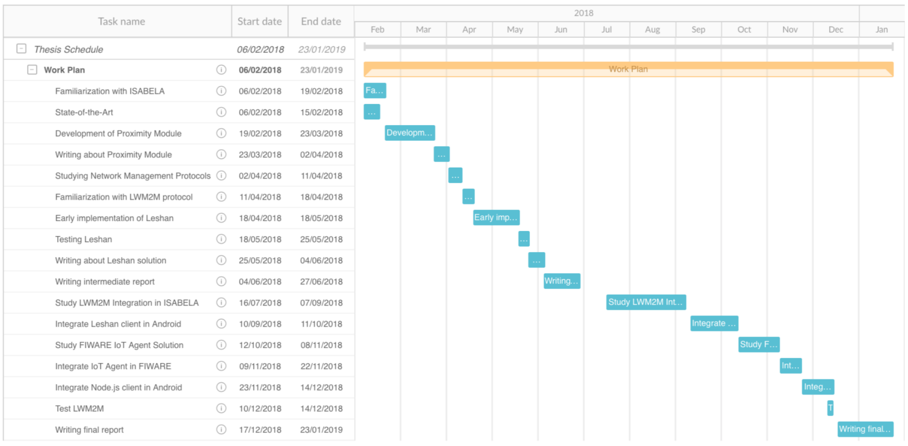
The fifth chapter is focused on the tests made to the application and consequent results.

The sixth chapter is dedicated to the conclusions taken from the development and the challenges that were found along the way. The potential future work is also presented in this chapter.

At the end of the document, there are Appendices where some specific configurations are explained and a scientific paper that I was a co-author of is presented.

1.4 Work Planning and Methodology

In this section, a timeline for the work developed and the followed methodology are presented.



Task name	Start date	End date
Thesis Schedule	*06/02/2018*	*23/01/2019*
Work Plan	**06/02/2018**	23/01/2019
Familiarization with ISABELA	06/02/2018	19/02/2018
State-of-the-Art	06/02/2018	15/02/2018
Development of Proximity Module	19/02/2018	23/03/2018
Writing about Proximity Module	23/03/2018	02/04/2018
Studying Network Management Protocols	02/04/2018	11/04/2018
Familiarization with LWM2M protocol	11/04/2018	18/04/2018
Early implementation of Leshan	18/04/2018	18/05/2018
Testing Leshan	18/05/2018	25/05/2018
Writing about Leshan solution	25/05/2018	04/06/2018
Writing intermediate report	04/06/2018	27/06/2018
Study LWM2M Integration in ISABELA	16/07/2018	07/09/2018
Integrate Leshan client in Android	10/09/2018	11/10/2018
Study FIWARE IoT Agent Solution	12/10/2018	08/11/2018
Integrate IoT Agent in FIWARE	09/11/2018	22/11/2018
Integrate Node.js client in Android	23/11/2018	14/12/2018
Test LWM2M	10/12/2018	14/12/2018
Writing final report	17/12/2018	23/01/2019

Figure 1.1: Work Plan.

As we can see, in Fig. 1.1, the first week was reserved for a familiarization with the ISABELA system that had already been developed and for a review of the similar projects, that will be presented in chapter 3.

The following month was spent developing an Android module for ISABELA. The Proximity Module helps measure sociability. It does it with the help of bluetooth, that starts a discovery every 5 minutes and by comparing the MAC address of all the devices discovered we can infer about the amount of time the participant spends alone, with the same group of people or surrounded by random people.

The following week served to document the module and all the procedure of development. The development of the proximity module is explained in chapter 4.

Then, it was two weeks to learn about the LWM2M protocol and other network management protocols. This study makes up the state-of-the-art, presented in chapter 2 and was followed by a month for an early implementation of the Leshan project, an Eclipse project that helps implement the LWM2M protocol.

The rest of the intermediate report was written during the month of June.

Most of July and August were occupied studying ways in which we could integrate the LWM2M protocol in ISABELA.

In September, the integration of the Leshan client in the Android application started. That took about a month and after that came the study of a new approach for integrating the LWM2M protocol in the system - the IoT Agent solution. Its documentation appears in chapter 4.

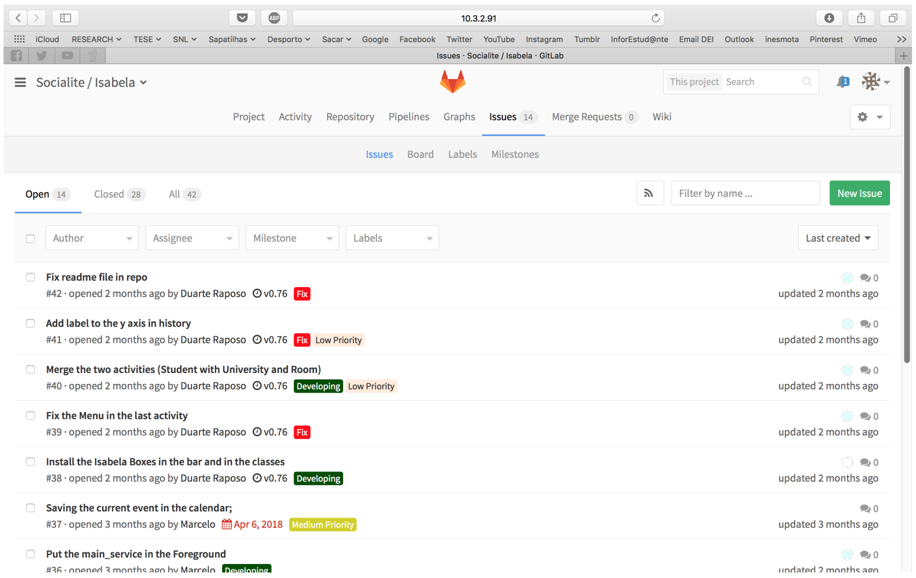
Then, it was two weeks for the integration of the agent with FIWARE and three weeks after that for the integration of the Node.js client in the ISABELA Android application.

Finally, from the 15th of December through January of this year occurred the writing and revising of this document.

In this project, we used Scrum methodology [8]. Scrum is an agile methodology, which means that it works on the basis of small increments that, eventually, make up a final product fully functional and up-to-date. This makes us work with Sprints. Sprints are iterations of the project and have a duration of 2 to 3 weeks. For example, the development of the Proximity Module was a sprint, in which we started by planning the requirements for this implementation, then we built the module and tested it and, finally, we reviewed the changes made. At the end of every Sprint, there should be a smaller product, ready for deployment.

Through the development of the project, every week on Thursday, there was an individual meeting with the supervisor in order to decide what needed to be done that week and to expose difficulties and a group meeting in which each member presented briefly the work they had been doing.

The use of GitLab was very useful as a team management tool.


10.3.2.91
Issues · Socialite / Isabela · GitLab
Socialite / Isabela
This project Search
Project Activity Repository Pipelines Graphs Issues 14 Merge Requests 0 Wiki
Issues Board Labels Milestones
Open 14 Closed 28 All 42
Filter by name ...
New Issue
Author Assignee Milestone Labels
Last created

Fix readme file in repo
#42 · opened 2 months ago by Duarte Raposo v0.76 Fix — 0 — updated 2 months ago

Add label to the y axis in history
#41 · opened 2 months ago by Duarte Raposo v0.76 Fix Low Priority — 0 — updated 2 months ago

Merge the two activities (Student with University and Room)
#40 · opened 2 months ago by Duarte Raposo v0.76 Developing Low Priority — 0 — updated 2 months ago

Fix the Menu in the last activity
#39 · opened 2 months ago by Duarte Raposo v0.76 Fix — 0 — updated 2 months ago

Install the Isabela Boxes in the bar and in the classes
#38 · opened 2 months ago by Duarte Raposo v0.76 Developing — 0 — updated 2 months ago

Saving the current event in the calendar;
#37 · opened 3 months ago by Marcelo Apr 6, 2018 Medium Priority — 0 — updated 3 months ago

Put the main_service in the Foreground
#36 · opened 3 months ago by Marcelo Developing — 0 — updated 2 months ago

Figure 1.2: GitLab project's issues.

As we can see, in Fig. 1.2., every member of the team had access to all the projects that the group was working on. This tool allowed us to coordinate work, as every issue could represent work to be done, it could be assigned to a specific user and it could have a deadline to follow.

As a form of communication, we used Slack [9] and Email. However, Slack allowed us to create specific channels for specific problems, thus, obtaining faster answers.

1.5 Concepts

In this section, a brief summary of some concepts that help contextualize the project is presented.

1.5.1 Internet of Things

Generically, the term "Internet of Things" refers to scenarios where everyday devices (not normally considered computers) are able to connect and generate, exchange and consume data, with minimal human intervention [1].

This term was coined by Kevin Ashton, in 1999. Ashton used the term to reference the linkage of RFID tags used in corporate supply chains to the Internet for tracking purposes. However, that concept evolved and other techniques to connect everyday devices emerged. The first IoT applications appeared from health-care, transportation and automotive industries and recently are moving towards smart-cities [3].

Presently, there are billions of connected IoT devices and the tendency is for that number to grow (Fig. 1.3), as new techniques for implementation of IoT applications arise.

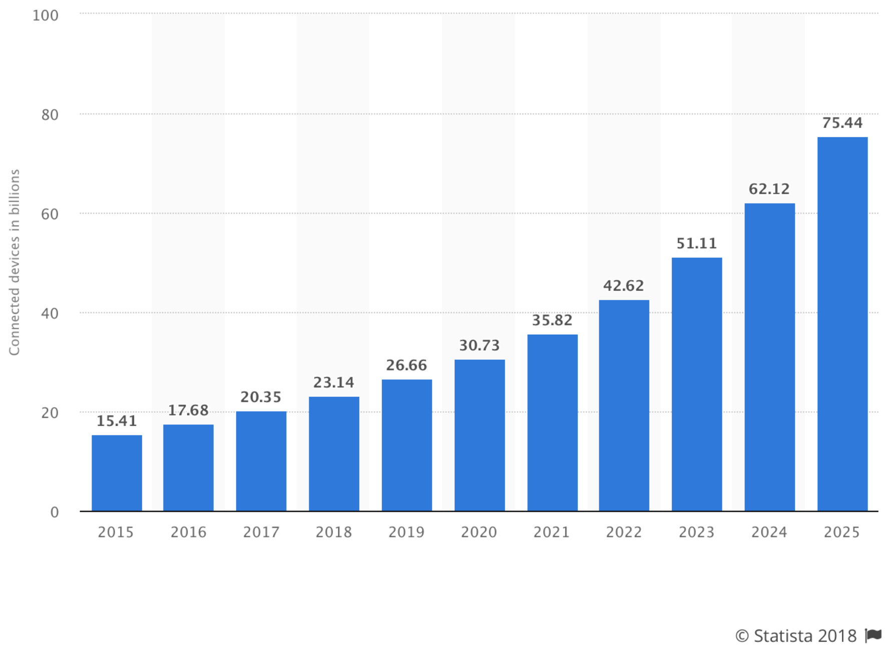


Figure 1.3: Number of connected devices worldwide from 2015 to 2025 [2].

IoT devices are no longer, only, computers or phones. Many more everyday devices are empowered with communication abilities, such as fridges, TVs and fireplaces.

1.5.2 Cyber-physical Systems

CPS aim to determine the way we interact with the physical world around us [10]. The concept of CPS rises from a junction of several elements (which include sensor networks and IoT devices) that work together in order to control and monitor the physical environment around them [4].

CPS are already used in various industries. They are mostly used to help with the betterment of processes in place, by sensing the environment in which that processes are inserted into. These systems must include sensors and actuators and use the computational capabilities of the devices to reach desirable results [4].

If these systems are made to be useful for the users, we enter the realm of HiTL CPS, which take into account human actions and emotions.

1.5.3 Human-in-the-Loop

Most of the CPS are applications centered in humans, however, still consider them to be external actors. In the future, if we want to create technology to better serve humans, these systems will need to take into consideration human dependent factors, such as emotions and actions [4].

The process of HiTL control (Fig. 1.4) has humans in the center of the system and works in four phases:

1. The *data acquisition* is made by gathering information through the available sensors. This phase is made easier by the gadgets that humans carry with them on a daily basis, which are full of sensors.

2. The data acquired is processed and is used to *infer the user's state.* This step helps improve the performance and accuracy of the control's loop.

3. Some procedures try to *predict future states* based on information previously gathered and the current state.

4. Finally, the systems use the information gathered, processed and the user's state to determine if a certain action may be performed.

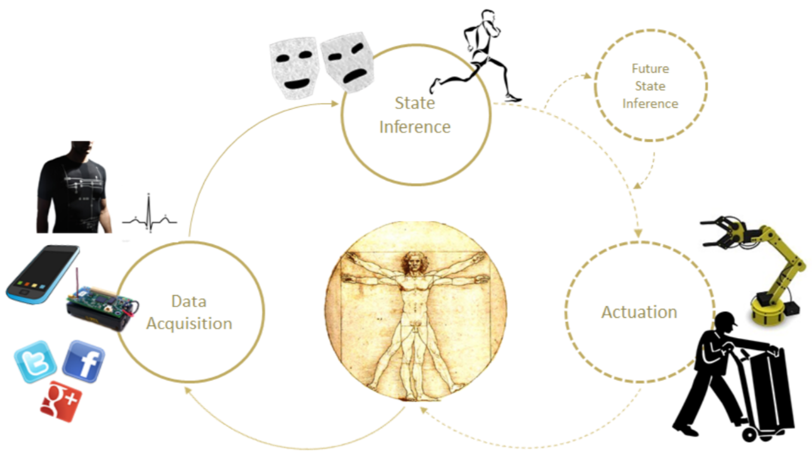


Figure 1.4: The process of HiTL [4].

1.6 Used Technologies

In this section, the technologies used during the first phase of the project are going to be presented.

1.6.1 Android

Android is a mobile Operating System (OS) developed by Google and based on the Linux Kernel. Initially, it was created by Android Inc. for digital cameras, but then Google acquired the company, in order to enter the mobile market [11].

It was primarily designed for smartphones and tablets, but, over the years, Google developed new versions of the OS for smart TVs (Android TV), for cars (Android Auto) and for smartwatches (Android Wear). Other variants may also be found on game consoles and digital cameras.

Android benefits from the fact that several smartphones manufacturers build devices specifically designed for the Android system. This greatly helps Google in gaining market share, making them accountable for 87,7% of the shares of the global mobile operating system market [12]. This reflects in 2 billion monthly active users and 3.6 million apps available in the Google Play Store [13].

As the Android OS is an open-source technology, the main language used in the development of Android applications is Java, through the Java Development Kit (JDK), and it is so used and so well documented, it was the chosen platform for our application.

1.6.2 FIWARE

FIWARE is an IoT platform created in Europe by the Future Internet Public Private Partnership (FI-PPP) with the objectives of [14]:

- accelerating the development and adoption Future Internet technologies in Europe;

- advancing the European market for smart infrastructures;
- increasing the effectiveness of business processes through the Internet.

FIWARE, which means Future Internet Ware, is public, open-source and free of royalty [15] and one of its objectives is helping developing smart applications from various domains. Smart applications need to be constantly gathering information that may be important for the application (context information), then that information will be processed and analyzed and the results shown. FIWARE aims to standardize the way to collect, manage and publish context information and to solve the heterogeneity in IoT protocols and translate the information to a common language [16].

FIWARE core platform is based on Generic Enablers (GE), which implement Application Programming Interface (API) to make easier the development of smart applications [17]. GE are divided into seven categories:

- **Data/Context Management:** Easing access, gathering, processing, publication and analysis of context information at large scale.
- **IoT Services Enablement:** Make connected things available, searchable, accessible and usable.
- **Advanced Web-based User Interface:** 3D and AR capabilities for web-based UI.
- **Security:** Make delivery and usage of services trustworthy by meeting security and privacy requirements.
- **Interface to Networks and Devices (I2ND):** Build communication-efficient distributed applications, exploit advanced network capabilities and easily manage robotic devices.
- **Architecture of Applications/Services Ecosystem and Delivery Framework:** Co-create, publish, cross-sell and consume applications/services, addressing all business aspects.
- **Cloud Hosting:** Provides computation, storage and network resources to manage services.

1.6.3 Raspberry Pi and Arduino

Two types of embedded circuits are used in this project: Raspberry Pi [18] and Arduino [19]. Their integration in ISABELA serves as a way to obtain information regarding the environment where the participant is inserted in and provide more founded tips.

Raspberry Pi is a microprocessor, therefore, is a smaller general-purpose computer with an OS and the ability to run various programs at once. Arduino is a micro-controller and for this reason is a simpler computer only able to run one program at a time, repeatedly.

However, both are valuable to ISABELA. As the Raspberry Pi is more powerful, better at multitasking and able to process larger volumes of data, Arduino is better for simple repetitive tasks, such as turning something on and off [20].

Chapter 2

Network Management Protocols

ISABELA uses various sensors, not only physical but also virtual, present in different devices. We were looking for a generic way to deal and manage these devices. There are various Network Management Protocols able to handle IoT devices. We started studying some of the available ones and decided which would be more appropriate for the ISABELA system.

In this chapter, there is an overview of the considered protocols and a comparative analysis of them.

2.1 Protocols

The number of active devices is rising and their complexity is increasing. This resulted in the creation of various tools and technologies specialized in IoT management.

Management functionalities require data exchanges between the manager and the managed systems. The managed systems are often different from each other in terms of computing capabilities, storage and energy consumption.

The heterogeneity of the devices that make up the system must be considered when a way to manage them is being chosen. Several organizations developed standards able to handle heterogeneous devices and some of those were considered for this work and are going to be presented in the next subsections.

2.1.1 Network Configuration Protocol (NETCONF)

Network Configuration Protocol (NETCONF) is a network management protocol proposed by the Internet Engineering Task Force (IETF). It was created so that it was possible to configure networks of devices, implementing functions to install, edit and delete its parameters [21]. It uses XML encoding for the configuration data as well as the protocol messages. The transport protocol is Transmission Control Protocol (TCP), generally. Its operations are performed as remote procedure calls. NETCONF establishes SSH sessions between its server and its client [22] and defines the configuration of data stores and a set of operations - Create, Read, Update and Delete (CRUD) - that can be used to access the aforementioned data stores [23][24].

2.1.2 RESTCONF

RESTCONF is a protocol based on HTML that provides an interface for accessing data defined in Yet Another Next Generation (YANG)[1], using datastores defined in NETCONF. Its operations are the HTTP operations GET, POST, PUT, PATCH and DELETE and the transport protocol is HTTP. It uses HTTP methods as equivalents for NETCONF operations. This protocol was also designed by the IETF [25] and does not intend to replace NETCONF. In fact, they can coexist (Fig. 2.1), so that RESTCONF can offer an additional interface with REST-like functionalities to the NETCONF protocol, or it can be used alone (Fig. 2.2) [21].

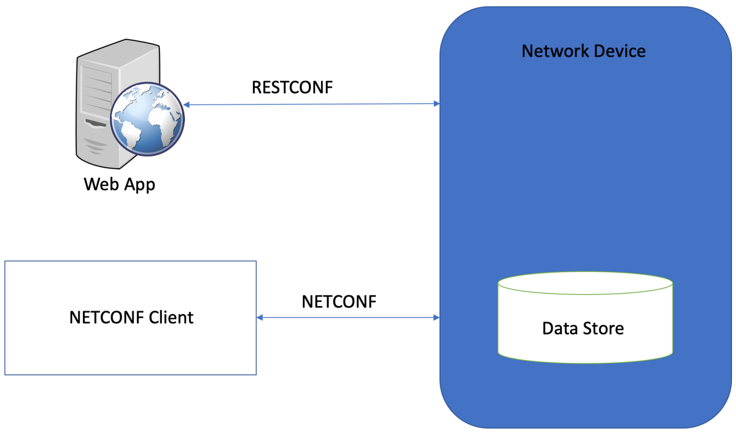


Figure 2.1: RESTCONF with the NETCONF Server [21].

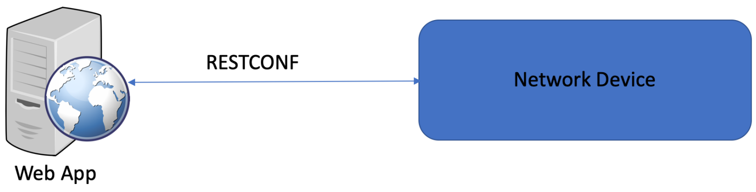


Figure 2.2: RESTCONF without the NETCONF Server [21].

2.1.3 Lightweight machine-to-machine

LWM2M is a protocol for IoT device management from Open Mobile Alliance SpecWorks [26]. It was designed for sensor networks and remote management of machine-to-machine devices. Its architectural design is based on Representational State Transfer (REST), defines an extensible resource and data model and builds on an efficient secure data transfer standard called the Constrained Application Protocol (CoAP)[2]. The target devices are mainly resource constrained, so the protocol is light and compact.

[1]Data modeling language for the definition of data sent over the NETCONF protocol.
[2]CoAP is a specialized Internet Application Protocol for constrained devices, defined in RFC 7252 [29].

2.2 Comparative Analysis

NETCONF was the first real solution for the integration of a network management protocol as one of its strengths is its support for robust configuration change involving multiple devices. These transactions are done for multiple devices and are mostly important when configuring services across network elements [27]. But for networks of constrained devices the operations tested were only *get*, *get-config*, *copy-config*, *lock* and *unlock*. NETCONF does not require necessarily a security mechanism but some authors tested a light version of the protocol with some mechanisms in constrained devices and concluded that it is inefficient in terms of memory usage [28].

RESTCONF is a more recent bet from the IETF and, as said before, does not intend to replace NETCONF. Unlike NETCONF, RESTCONF allows for XML and JSON to be used and does not have the concept of distributed transactions, only device-by-device configuration. Also, a call from this protocol is a transaction by itself as it uses HTTP operations to edit data resources [27] and it requires a security mechanism - Transport Layer Security (TLS).

LWM2M is a client-server model and requires end-to-end IP connectivity. Sometimes this poses as a challenge but some enhancing strategies are being proposed in recent works. Despite this, LWM2M is a widely implemented and used protocol and is considered a very good solution for the management of constrained IoT devices [21].

The characteristics of each protocol analyzed are summarized in table 2.1.

Protocol	NETCONF	RESTCONF	LWM2M
Standard	IETF	IETF	OMA
Definition Language	YANG	YANG	LWM2M Language, YANG Extended
Information Model	YANG Modules	YANG Modules	LWM2M Objects
Instantiated Information/Transfer Syntax (payload)	XML	XML or JSON	Plain Text, JSON or TLV
Transfer Protocol	SSH, SSL, HTTP TLS	HTTP, TLS, HTTPS with X.509v3	CoAP, DTLS, UDP
Security	Yes	Yes	Yes
Used on Constrained Devices	Yes, reducing some operations	Yes, reducing some operations	Yes

Table 2.1: Summarized Protocols [30].

In conclusion, we wanted a generic mapping for heterogeneous devices. For these purposes, the best options are RESTCONF and LWM2M. As LWM2M was designed specifically for sensor networks and the remote management of machine-to-machine IoT devices, we chose to implement this protocol.

This page is intentionally left blank.

Chapter 3

ISABELA

The objectives of my work go through, as exposed in 1.2, the implementation of a functionality for the ISABELA application and the incorporation of a network management protocol that will make use of some of the modules that the system already had. This chapter is used to give a general overview of ISABELA and briefly explain its most relevant components for this thesis.

3.1 General Overview

ISABELA mobile application was developed to collect data from students and evaluate it in order to assess the state of the participants and help them, by providing feedback and advices to try to prevent poor academic results.

Depression, caused by stress and feelings of overwhelm, is one of the principal causes of dropout amongst students between the ages of 18 and 24. If we can monitor the day-by-day of the students, like the amount of sleep they take, the amount of exercise and their sociability, maybe we can give advice on how to prevent these factors from interfering with their work-life and achieve better outcomes.

3.1.1 Description

ISABELA is a HiTL CPS application that uses smartphones' sensors, other physical sensors and some virtual ones too to collect information of the participants' day-to-day life, in order to detect behavior that is potentially bad for good academic results. This application is meant to be used by college students.

The application monitors the students continuously. The sensors retrieve information that is sent to FIWARE through an Internet connection. Then, the data is retrieved from FIWARE, it is processed and shown to the user. When behavior that is not deemed helpful to achieve academic success is detected, a ChatBot sends a notification and a message and if the user wants to know more about how to prevent that type of behavior, they can ask the ChatBot and it gives back a well-founded answer.

To assess the students' behavior, ISABELA collects information about their activity, sleep, sociability, social media presence and location.

3.1.2 Similar Projects

StudentLife and Smart Kindergarten are two projects with similar objectives to ISABELA. They are based on HiTL CPS and centered on smartphones applications.

3.1.2.1 StudentLife

StudentLife is a case study developed by a research team from Dartmouth College. They developed an Android application in order to assess the impact of workload on day-to-day activities [31].

The data was collected through the accelerometer, microphone, light sensor, GPS and Bluetooth sensors of the mobile phones to assess the user activity, conversation patterns, sleep cycles and location. Then, they added information obtained through some surveys to gauge the levels of stress, the mood, the sociability and the behavior of the participants. The outcomes of the study emerge of the correlation between the gathered data and the students' mental health, their academic performances and the Dartmouth term lifecycle.

This study was conducted over a period of 10 weeks, with 48 participants. The results show a significant correlation between the state of mind of a student and its own academic success. The students start the term with healthy levels of activity, sociability and sleep and as the term progresses and the number of assignments increases these levels drop.

3.1.2.2 Smart Kindergarten

Smart Kindergarten is a case study developed by the Networked & Embedded Systems Laboratory at the University of California Los Angeles. The intention of the study was to provide tools to the educators for them to comprehend a students learning process. This tools are obtained by evaluating the students progress, considering where the student spends their time, and sociability, by evaluating the time a student spends alone. In order to do that, it needs to collect and interpret information that come from sensors and present it in a logical and easy-to-understand way.

In this study, the location of sensor nodes is calculated using an algorithm that utilizes known locations and distance measurements over multiple hops, by making use of the sensors to gauge the distance between them and the next nodes and interchange that information to estimate their location. They suggest the term *collaborative multilateration* (or *n-hop multilateration primitive*) which consists of the process explained before and locations are obtained with high accuracy [32].

3.1.2.3 ISABELA vs Similar Projects

There are several differences between these two case studies and ISABELA. The main difference is the HiTL element that our project has and the others do not. Smart Kindergarten's intention is to provide tools of evaluation to educators, it does not include giving advice to students based on their detected behavior. StudentLife, although closer to ISABELA, does not give feedback to the students to ease their stress and help them perform better.

In ISABELA, we close the loop by gathering behavioral information through the sensors on the participants' phones, environment information through the Arduino and Raspberry

Pi and emotional information through virtual sensors (for example, social media presence). This information is then processed through a variety of analytical algorithms and transformed into quantitative data. With this data, we provide feedback and advice to the user, through ChatBot messages.

Another difference is that in Smart Kindergarten they created an algorithm that uses known locations and, then, calculations of distance measurements to estimate precise locations for sensor nodes, in order to know the location of a student. In ISABELA, we use the SSID of the Wi-Fi hotspots to know whether the user is at Department of Engineering Informatics (DEI) (if yes, the specific location within the building) or at home.

The sociability part of the Smart Kindergarten study gave us the idea for the proximity module on ISABELA, in order to know if the students have a group of friends and if they spend time with them. But, again, the method is different. In Smart Kindergarten, they obtain this information based on location and, in ISABELA, we use bluetooth to discover nearby devices and see if the participant spends considerable time with the same "devices".

We, also, did not implement a simple server-application architecture as it is common in projects with mobile applications. We implemented an IoT architecture with an IoT middleware platform, that handles the communication protocols needed to establish connections with different kinds of embedded devices, making it easier if we want to add other types of devices, in the future.

3.1.3 User Interface

Usability is a very important quality of a piece of software because it is the most visible to the user and, nowadays, they are easily influenced by what they see (and like).

In this section, some screens of the ISABELA mobile application are going to be presented.

The first screen that the user sees, after clicking on the application icon, is a loading image that has the name of the application (ISABELA), with an animation that says "Made by University of Coimbra".

After the introductory screen, comes the "Login" screen (Fig. 3.1). Login is made through Facebook and only after that does the user have access to the application.

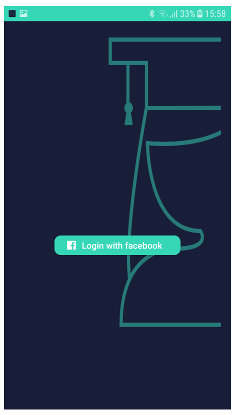


Figure 3.1: Login Screen.

The information screens are the most important. In them, the user can see their behavior and receive messages if the application thinks that they need to change.

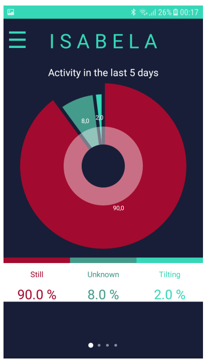


Figure 3.2: Activity Screen.

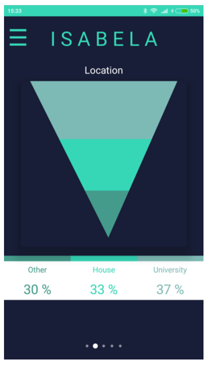
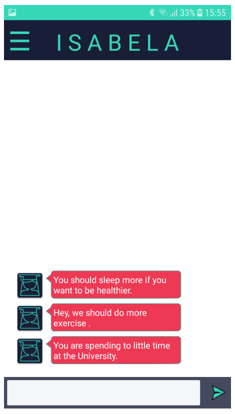


Figure 3.3: Location Screen.

In Fig. 3.2, the user sees the percentage of their physical activity and, in Fig. 3.3, the user sees their location. If the application believes that some percentages are very high and that that behavior is prejudicial the color on the charts changes to red and it sends a message through the ChatBot (Fig. 3.4).


ISABELA

You should sleep more if you want to be healthier.

Hey, we should do more exercise .

You are spending to little time at the University.

Figure 3.4: ChatBot.

The application has many other functionalities, such as a *Sleep Form*, which each user fills with information about the quality and quantity of their sleep or a *History* in which the user can see the evolution of the values of activity, location, sociability and amount of sleep.

3.2 System Architecture and Components

In this section, we aim to expose the architecture of the system and explain with greater detail some of its components. For that, the system architecture and the FIWARE architecture and modules used in the system are going to be presented.

3.2.1 Overall System

ISABELA is an HiTL CPS, whose goal is to prevent bad academic performances. As said before, this type of systems work in four phases: data acquisition, inference, future inference and actuation. With that in mind, the architecture, represented in Fig. 3.5, emerged.

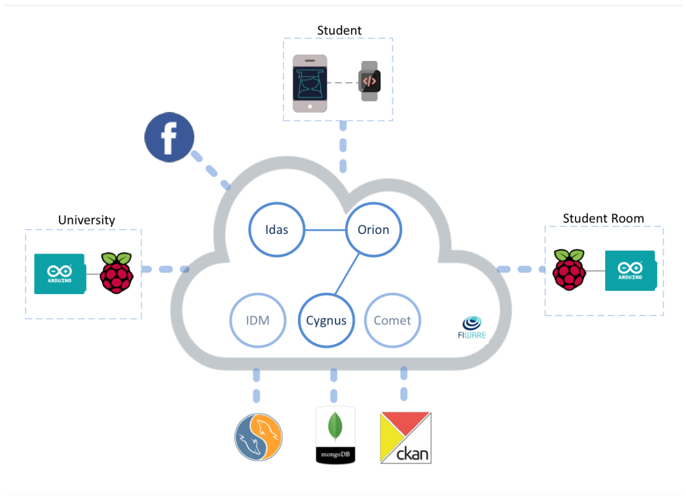


Figure 3.5: System architecture.

We use the smartphone and all the sensors to acquire the data. FIWARE provides the storage capabilities and allows the communication between the devices. State inference is made in the smartphone and the actuation is performed through messages and notifications, based on the previous inferences.

3.2.2 FIWARE Architecture and Modules

FIWARE has the objective of standardize the way we collect, manage and publish context information and to solve the heterogeneity in IoT protocols. Therefore, it is used as the backend of the ISABELA system. In Fig. 3.6, its architecture is represented.

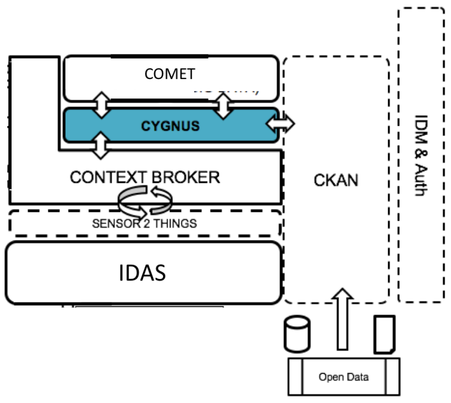


Figure 3.6: FIWARE architecture.

As we can see, the ISABELA system uses many FIWARE modules. These modules are going to be explained in the following sections.

3.2.2.1 ORION

ORION is the context broker. This broker is capable of representing several IoT contexts using a new representation standard – FIWARE NGSIv2 API. This API implements a REST API, therefore, is capable of performing updates, queries or react to changes. This is a must have capability, as we want to create connections between the sensors and the applications that consume the information.

ORION only holds information about the last instance of an entity and/or an attribute. To save the context history, created by the evolution of context information overtime, we need to accompany ORION with COMET and CYGNUS [33].

3.2.2.2 COMET

COMET or FIWARE Short Time Historic is a component capable of storing and retrieving historical context information [34]. As said before, it communicates with ORION and lets external clients, like ISABELA, query the stored information, through REST API.

COMET is very useful because it allows us to query specific time intervals and aggregate information by time, sums and occurrences.

COMET is a very important component of this system, as it is from it that it gets most of the data in the application.

3.2.2.3 IDAS

IDAS is the module that handles the communications with the devices that make up the system, as it offers a wide range of IoT agents [35].

This module is needed to connect objects to gather data. While the smartphone connects directly with ORION, the embedded devices connect to IDAS. Its IoT agents translate IoT-specific protocols into FIWARE standard data exchange model. By using an IoT agent, the devices can subscribe entities and can query and be updated if a value of that entity is changed [36].

3.2.2.4 Other Modules

- **CYGNUS**

  CYGNUS is a connector in charge of coordinating the data, creating a historical view of such data. We can introduce this data into third-party storage systems, such as MongoDB or MySQL. It also connects the ORION to many FIWARE storages, like CKAN and COMET [37].

- **CKAN**

  CKAN is not a FIWARE module. It is an open data platform that makes it easy to publish, share and work with data. This platform allows us to store the retrieved data, with the advantage that it has a rich front-end and visualization tools, that we can use to see the data [38].

CKAN provides a powerful way for cataloging and accessing datasets. If the data retrieved during this project is available on this platform, it can be used by other investigators.

In conclusion, in order to fulfill the objectives of my work, I had to work with FIWARE, especially in close detail with the ORION and IDAS modules, and the ISABELA Android application. It is important for this chapter to be present in order to contextualize the scope of the work I have done.

This page is intentionally left blank.

Chapter 4

Development

In this chapter, the development of the work done is exposed. First, the requirements for each of the objectives presented in 1.2 are stated, then all the steps taken to produce the Proximity Module are detailed and finally the incorporation of the LWM2M protocol in the ISABELA system is explained.

4.1 Requirements

In this section, the requirements for the two components developed throughout this thesis are presented.

4.1.1 Proximity Module

- **Functional Requirements**

Requirement	Bluetooth Discovery.
Priority	Must be implemented.
Description	The application must have a bluetooth sensing function in order to gather information.
Actors	The system.
Pre-conditions	The device must have a bluetooth sensor and the user must give permission for the application to use bluetooth.
Events Flow	The phone starts sensing right after the application is turned on.
Expected Outcome	If the sensing is successful, the application must show the results to the user.

Table 4.1: Bluetooth Discovery Requirement.

Requirement	Friend's Top Screen.
Priority	Must be implemented.
Description	The application must have a module that tells the user the amount of time he spends alone, with friends or amongst random people.
Actors	The user.
Pre-conditions	The application must have access to an Internet connection.
Events Flow	From the navigation drawer, the user presses the "Friends" button, then he can see the amount of time he spent alone and a list with the names of the devices and the amount of time he spent with those "devices".
Expected Outcome	When the button is pressed, the screen changes and the list should be populated with the list of devices and the amount of time, in a few seconds. If there are no devices, the only thing that appears is the alone time.

Table 4.2: Friend's Top Screen Requirement.

- **Other**

  Privacy and Security are not focal points of this thesis. Nonetheless, they are covered by the encryption of the Entity User's ID, which was the only thing that could breach the participants privacy, and the secure channel that the information passes by, implemented by my colleagues. Even so, all the participants signed a term in which they consented the use of their information for academic purposes.

4.1.2 Device Management

- **Functional Requirements**

Requirement	LWM2M Protocol.
Priority	Must be implemented.
Description	The system must have a protocol to help manage all the heterogeneous devices.
Actors	The system.
Pre-conditions	The system must have access to an Internet connection.
Events Flow	A device connects to the server and it appears on its page. If we press the name of the device, we can see more detailed information
Expected Outcome	When a new device tries to connect to the server of the ISABELA system and is successful, it appears on the server page and we can see its detailed information. If not, the device does not appear.

Table 4.3: LWM2M Protocol Requirement.

- **Other**

  Security is covered by the LWM2M protocol which has its own protocol to ensure it, as it is going to be explained in a later section.

4.2 Proximity Module

In the StudentLife study [31], sociability was considered to be affected by the students' workload and, consequently, to have an impact on the student's life and academic performance. So, in that study, in order to determine the "amount" of sociability, conversations were recorded, through the smartphone's microphone, and the measure was considered to be the frequency and duration of conversations around a student.

In ISABELA, although the impact of sociability is also considered, that was not the used approach. In a first approach, 5 factors were defined to calculate the sociability: SMSs' frequency, Calls' frequency, SMSs' ratio, Calls' ratio and the diversity [39]. Now, the factors considered are the number of SMSs' and Calls' frequency and with the addition of a virtual sensor - the bluetooth - with which the proximity module was implemented.

The **proximity module** aims to bring to light the amount of time a participant spends with the same people, amongst random people or alone. This provides data that will make it possible to infer about the sociability of the participants, if they have a group of friends he likes to spend time with or if we are dealing with a lonely person.

The objective of this module is only to gather data about the amount of time the participants spends with the same "devices". In this work we do not propose a way to infer about sociability based on the gathered data.

Nevertheless, in order to measure this data, the ISABELA application uses bluetooth functionalities provided by the Android application framework. In the next subsections, it is going to be explained how the data is acquired, stored, managed and displayed.

4.2.1 Data Acquisition

The acquisition of data, for this module, is made through the smartphone's bluetooth sensors. As it was said before, the Android application framework provides access to Bluetooth functionalities through the Android Bluetooth API [40]. This API allow wireless connections between applications and other Bluetooth devices.

By using the Bluetooth API, an Android application can perform various activities, such as:

- Scan for other Bluetooth devices;
- Query the local Bluetooth adapter for paired Bluetooth devices;
- Establish RFCOMM channels;
- Connect to other devices through service discovery;
- Transfer data to and from other devices;
- Manage multiple connections.

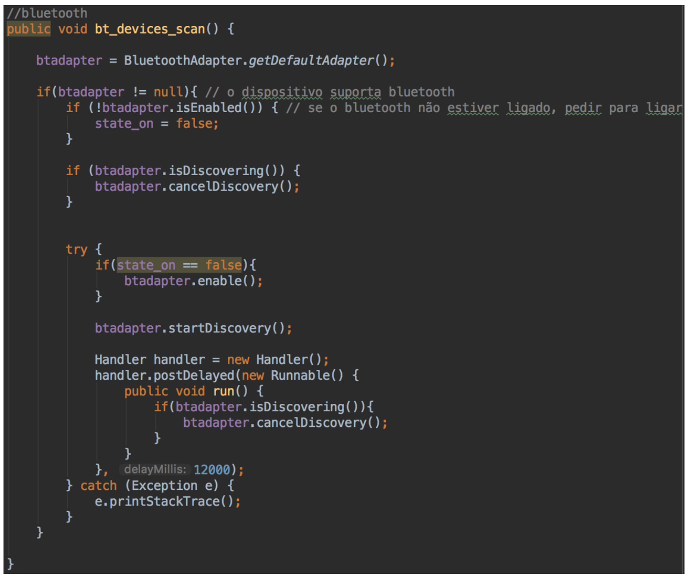
Our focus will be on the first activity listed. Each device will be performing scans, throughout the day, and collecting the name and MAC address of the discoverable devices around the participant. In order for that to happen, the ISABELA application has to have a service that is always running so that the scans are performed automatically.

The service was implemented in a class named "Main_Service.java". Four parts make up this service:

- A function where the discovery of nearby devices is enabled (and disabled) through a *Bluetooth Adapter* [41];
- A *Broadcast Receiver* that receives and handles broadcast intents [42];
- A timer that controls the amount of scans that are performed;
- Storage of the information.

First, in the function "bt_devices_scan" (Fig. 4.1), a *Bluetooth Adapter* is declared to get a handle of the default local Bluetooth adapter. This returns the default local adapter or *null*, if the device does not support Bluetooth [43].

Then, through the adapter, we enable the Bluetooth sensor on the device (in case it was not enabled already) and start the discovery. After 12 seconds[1], if a discovery is taking place, we cancel it. Otherwise, it would be discovering throughout the entire time the service is running and that is too battery consuming.



```
//bluetooth
public void bt_devices_scan() {

    btadapter = BluetoothAdapter.getDefaultAdapter();

    if(btadapter != null){ // o dispositivo suporta bluetooth
        if (!btadapter.isEnabled()) { // se o bluetooth não estiver ligado, pedir para ligar
            state_on = false;
        }

        if (btadapter.isDiscovering()) {
            btadapter.cancelDiscovery();
        }


        try {
            if(state_on == false){
                btadapter.enable();
            }

            btadapter.startDiscovery();

            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                public void run() {
                    if(btadapter.isDiscovering()){
                        btadapter.cancelDiscovery();
                    }
                }
            },  delayMillis: 12000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Figure 4.1: Funtion "bt_devices_scan".

The second part of this service, the *Broadcast Receiver* (Fig. 4.2), handles two actions:

- *BluetoothDevice.ACTION_FOUND*;
- *BluetoothAdapter.ACTION_DISCOVERY_FINISHED.*

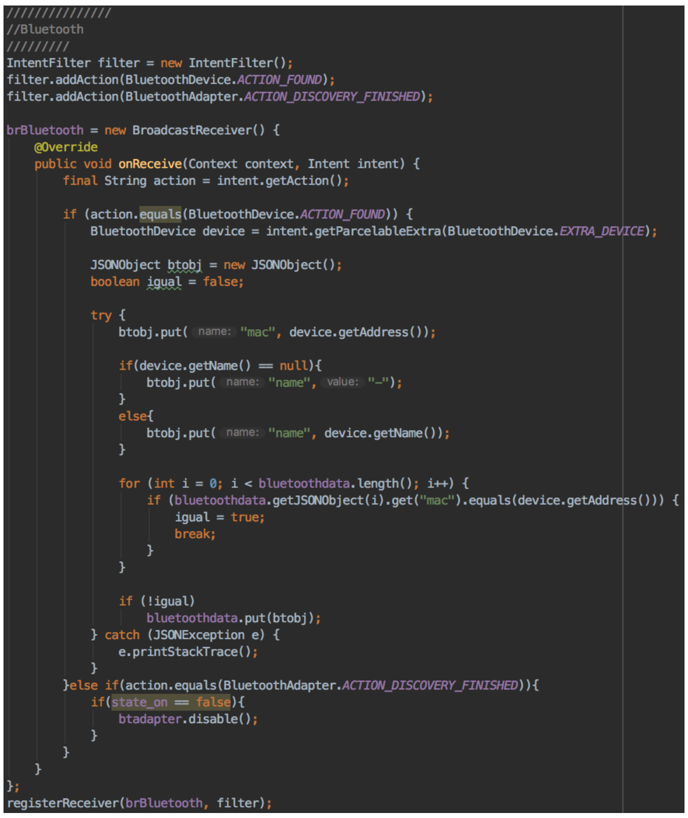
[1]The amount of time that a discovery of nearby devices takes, as per the Android Bluetooth API [44].

These two actions are added to the *Intent Filter*, through the method *addAction*. This means that these two actions are the intent values to be matched, that will be received by the bluetooth broadcast receiver [45]. With the help of the Intent method *getAction*, the received action can be retrieved to perform operations, accordingly [46].

If the intent action matches with *BluetoothDevice.ACTION_FOUND*[2], then, with the help of the Intent method *getParcelableExtra* and the Bluetooth Device constant *EXTRA_DEVICE*, the device found can be obtained. Once we have the device, we retrieve its name and MAC address through the methods *getName* and *getAddress*, respectively, and put it into a *JSON Object*. After that, we search the array of JSON objects for objects with equal MAC addresses. If there is none, we add the current device to the array.

If the intent action matches with *BluetoothDevice.ACTION_DISCOVERY_FINISHED*[3], we verify if the bluetooth sensor was turned off before the start of the discovery process and disable it, if it were.

Finally, we have to register the Broadcast Receiver for it to be run in the main activity thread. The receiver will be called with any broadcast Intent that matches the Intent filter, in the main application thread [49]. In order to do this, we use the method *registerReceiver*, which receives the broadcast receiver and the intent filter, as parameters.

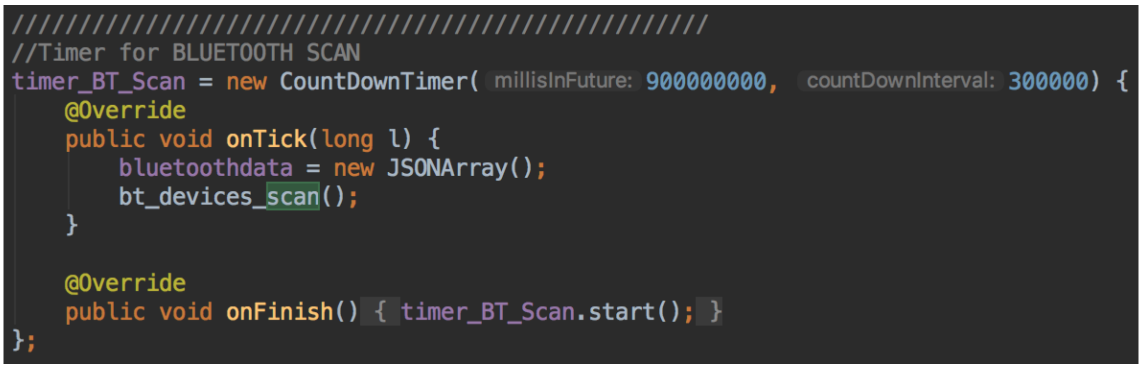


```java
///////////////
//Bluetooth
/////////
IntentFilter filter = new IntentFilter();
filter.addAction(BluetoothDevice.ACTION_FOUND);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);

brBluetooth = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();

        if (action.equals(BluetoothDevice.ACTION_FOUND)) {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

            JSONObject btobj = new JSONObject();
            boolean igual = false;

            try {
                btobj.put( name: "mac", device.getAddress());

                if(device.getName() == null){
                    btobj.put( name: "name", value: "-");
                }
                else{
                    btobj.put( name: "name", device.getName());
                }

                for (int i = 0; i < bluetoothdata.length(); i++) {
                    if (bluetoothdata.getJSONObject(i).get("mac").equals(device.getAddress())) {
                        igual = true;
                        break;
                    }
                }

                if (!igual)
                    bluetoothdata.put(btobj);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }else if(action.equals(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)){
            if(state_on == false){
                btadapter.disable();
            }
        }
    }
};
registerReceiver(brBluetooth, filter);
```

Figure 4.2: Bluetooth Broadcast Receiver.

Then, there is a timer (Fig. 4.3) that is set with an interval of 5 minutes. In this timer, the array of JSON objects is initialized and the function "bt_devices_scan" (Fig. 4.1), that handles the start and finish of the discovery process, as previously mentioned, is called.

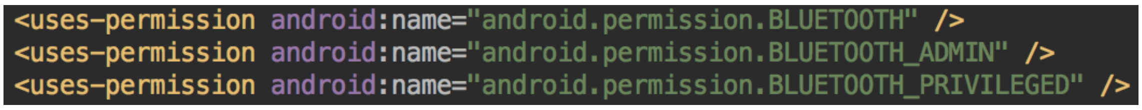
[2]This means that a remote device was found during the discovery process [47].
[3]This means that the discovery process of adjacent devices has finished [48].



```
////////////////////////////////////////////////////
//Timer for BLUETOOTH SCAN
timer_BT_Scan = new CountDownTimer( millisInFuture: 900000000,  countDownInterval: 300000) {
    @Override
    public void onTick(long l) {
        bluetoothdata = new JSONArray();
        bt_devices_scan();
    }

    @Override
    public void onFinish() { timer_BT_Scan.start(); }
};
```

Figure 4.3: Timer for Bluetooth scan.

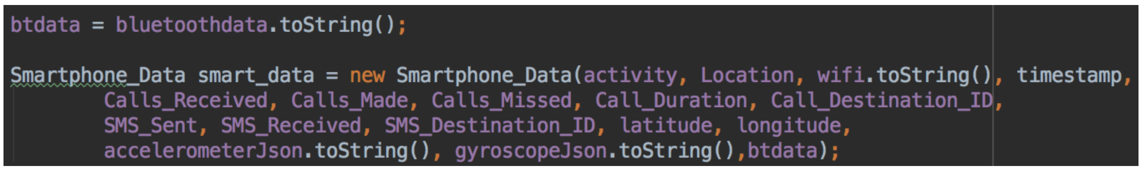
NOTE: So that we can use bluetooth and its methods, the application has to have permissions. So, the following lines (Fig. 4.4) were added to the *AndroidManifest.xml* file.



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

Figure 4.4: Bluetooth Permissions on AndroidManifest.xml.

Lastly, the data acquired from the bluetooth sensors is a field of a *Smartphone_Data*[4] object. This field is added in the form of a String and obtained from the JSONArray, which contains the JSON objects composed by the devices discovered, as it is shown in Fig. 4.5. This object is then stored into the database, in the "Main_service.java" method named "sendBroadcast".



```
btdata = bluetoothdata.toString();

Smartphone_Data smart_data = new Smartphone_Data(activity, Location, wifi.toString(), timestamp,
        Calls_Received, Calls_Made, Calls_Missed, Call_Duration, Call_Destination_ID,
        SMS_Sent, SMS_Received, SMS_Destination_ID, latitude, longitude,
        accelerometerJson.toString(), gyroscopeJson.toString(),btdata);
```

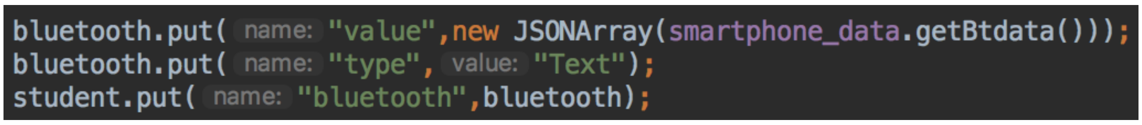
Figure 4.5: Smartphone_Data object with the data acquired from the bluetooth sensors.

To be stored in FIWARE [50], the Smartphone_Data objects have to be transformed into JSON objects. So, a class named "Entity_To_JSON" was created.

In Fig. 4.6, part of an Entity_To_JSON method named *getJSON* is represented. A bluetooth field is being composed by the JSON array with the information gathered from the sensors and being put into a JSON object named *student*.

The information contained in the JSON object obtained in this method is made available in FIWARE, in the form of an Entity named "Student", so we can request its attributes in the way we want to display them on the screen.

[4]Smartphone_Data is a database entity.



```
bluetooth.put( name: "value",new JSONArray(smartphone_data.getBtdata()));
bluetooth.put( name: "type", value: "Text");
student.put( name: "bluetooth",bluetooth);
```

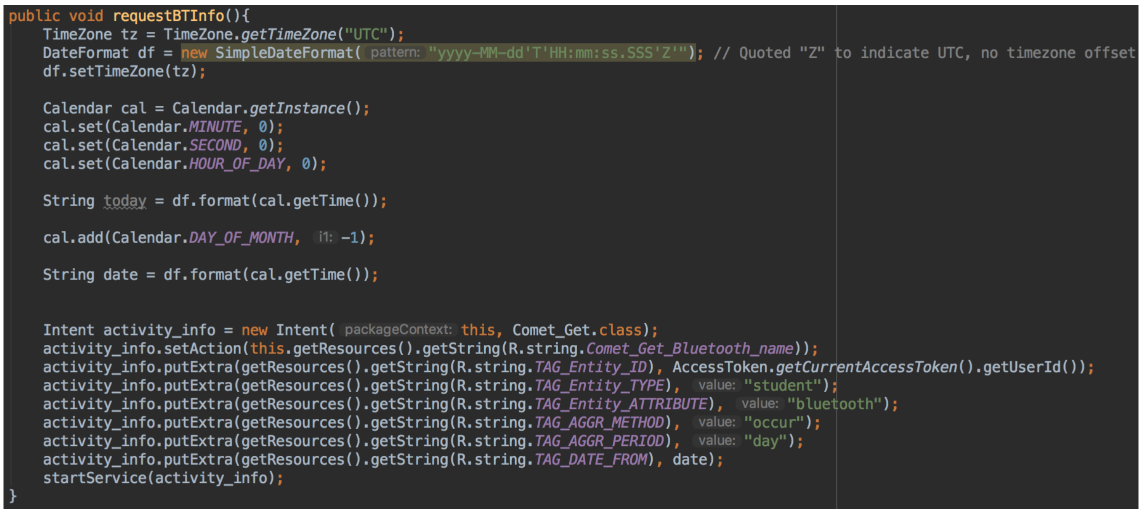
Figure 4.6: Bluetooth field of the JSON object student.

4.2.2 Communication

The communication for this module is made through COMET, presented in 3.2.2.2. As explained in the previous section, the information we want to display is already in FI-WARE. So, now, we want to retrieve it and, in order to do that, we have to query the COMET module.

The query is made in the "TopBTActivity.java" method named "requestBTInfo" (Fig. 4.7), with the aid of an intent that is used to start a service: "Comet_Get.java".

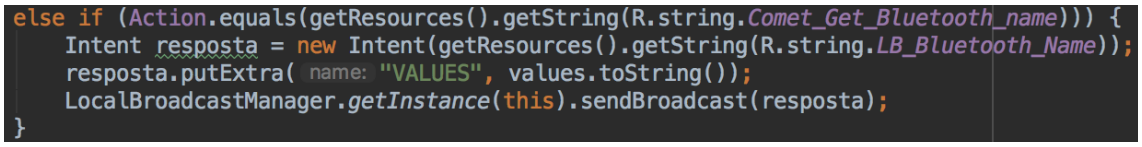
As what is wanted is the bluetooth information of the participant that has requested it through this Activity, in the intent, we put an *action*, that will identify what type of response we want from the "Comet_Get" service, and specific strings, that will tell COMET where to get this information from. In this case, we want the values of the *occur method*, of the *bluetooth attribute*, of the *student entity* with *id* equal to the *Access Token*, from a period of a *day* stated by the variable *date*.



```
public void requestBTInfo(){
    TimeZone tz = TimeZone.getTimeZone("UTC");
    DateFormat df = new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"); // Quoted "Z" to indicate UTC, no timezone offset
    df.setTimeZone(tz);

    Calendar cal = Calendar.getInstance();
    cal.set(Calendar.MINUTE, 0);
    cal.set(Calendar.SECOND, 0);
    cal.set(Calendar.HOUR_OF_DAY, 0);

    String today = df.format(cal.getTime());

    cal.add(Calendar.DAY_OF_MONTH,  i1: -1);

    String date = df.format(cal.getTime());


    Intent activity_info = new Intent( packageContext: this, Comet_Get.class);
    activity_info.setAction(this.getResources().getString(R.string.Comet_Get_Bluetooth_name));
    activity_info.putExtra(getResources().getString(R.string.TAG_Entity_ID), AccessToken.getCurrentAccessToken().getUserId());
    activity_info.putExtra(getResources().getString(R.string.TAG_Entity_TYPE),  value: "student");
    activity_info.putExtra(getResources().getString(R.string.TAG_Entity_ATTRIBUTE),  value: "bluetooth");
    activity_info.putExtra(getResources().getString(R.string.TAG_AGGR_METHOD),  value: "occur");
    activity_info.putExtra(getResources().getString(R.string.TAG_AGGR_PERIOD),  value: "day");
    activity_info.putExtra(getResources().getString(R.string.TAG_DATE_FROM), date);
    startService(activity_info);
}
```

Figure 4.7: Request bluetooth information.

After the request, in the "Comet_Get.java" service side, an url is constructed with the specific strings that came in the intent from "TopBTActivity" and the connection to COMET is made on a method called "onHandleIntent". After that, through a verification of the action (also sent in the intent) (Fig. 4.8), a new intent is created with a specific string and the values obtained from the request. The new intent is sent to a Broadcast Receiver, registered with a string equal to the specific string contained in the intent, with the help of the *LocalBroadcastManager*'s method *sendBroadcast(Intent)* [51].



```
else if (Action.equals(getResources().getString(R.string.Comet_Get_Bluetooth_name))) {
    Intent resposta = new Intent(getResources().getString(R.string.LB_Bluetooth_Name));
    resposta.putExtra( name: "VALUES", values.toString());
    LocalBroadcastManager.getInstance(this).sendBroadcast(resposta);
}
```

Figure 4.8: COMET response to the information request.

4.2.3 Android Application

Back in “TopBTActivity”, a Broadcast Receiver is registered with the same string (Figs. 4.9 & 4.10), in order to receive the information from the COMET module. Therefore, we can retrieve the data and treat it, in this function.

First, if the received intent is not *null*, we initialize an *ArrayList* of *BluetoothObject* (which has two string fields: name and address; and an integer field: count) and then we retrieve the data from the intent into a *JSONArray* (Fig. 4.9).

Then, if that JSON array is not empty, we iterate its length and get another JSON array of *points*, which is a more specific attribute of the COMET object. However, it is not specific enough, yet. Therefore, we have to go through its length and get a *JSONObject* of its attribute *occur*, which is the attribute that has the information we want.

The *occur* object is composed by *keys* and *values*. The *keys* are the information that was read by the sensors and the *values* are the number of times that that information was read. So, in order to obtain the data, we iterate the occur object’s keys.

In a cycle, while there are still *keys* to be read, we get the corresponding value and a JSONArray with the data of the key. If the JSONArray is empty, it means that the participant was alone, that is why the value is added to the variable *sozinho*[5]. Otherwise, we go through the array’s length and get each MAC address and name. But, before we build a new object with the information retrieved and add it to the Array List of bluetooth objects, we verify if there is already an object on that array list with an equal MAC address. If that is true we only add the new value to the count of that object. If it is not true we built a new *BluetoothObject* object with the name, MAC address and value retrieved and add it to the Array List.

After we have all the information in the Array List, there is only one thing left to do. As what we want is a top of the “friends” that the participant spent the most time with, we have to order the array list in descending order of “count”. For that, we *sort* the Array List as can be seen in Fig. 4.10. This sort method receives the *List* to be sorted and a *Comparator* to determine the order of the list [52]. The *Comparator* has a method *compare* that receives two *BluetoothObject*’s objects and compares their “count” fields.

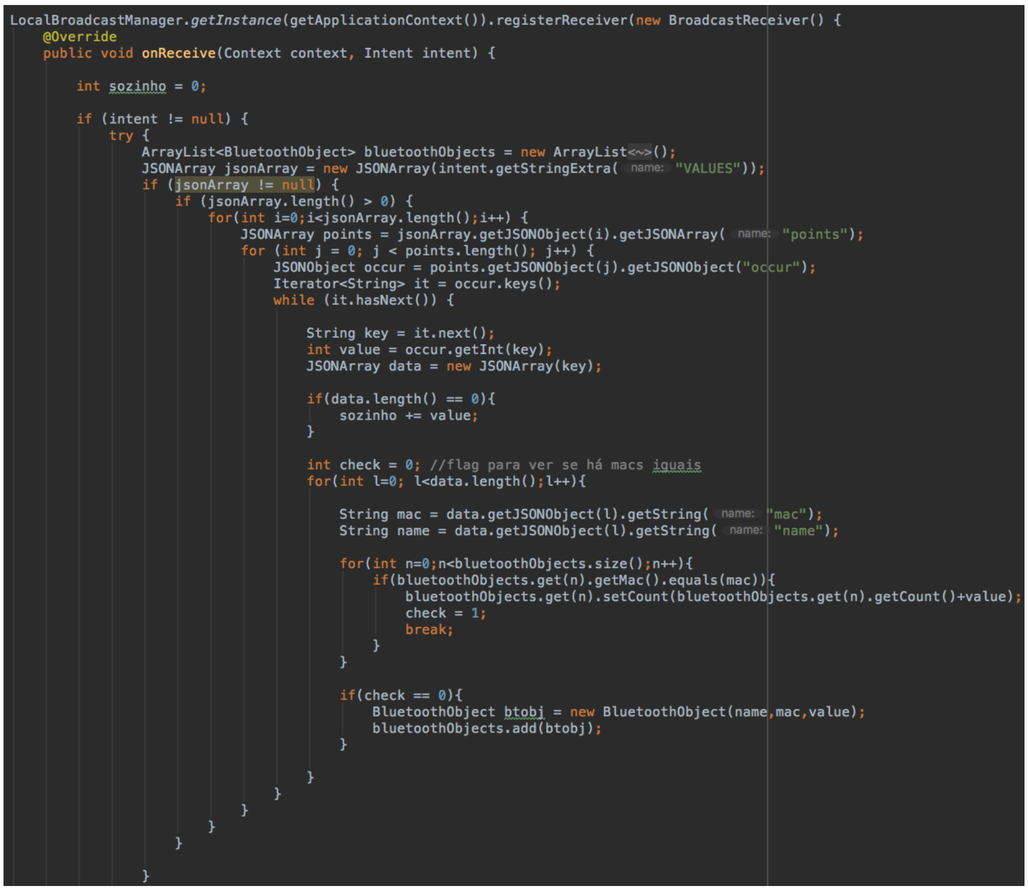
With the information all gathered and treated we can go to the display.

First of all, this activity is a screen with the alone time and a list composed by the name and MAC address of bluetooth devices and the time spent with “them”. So, we have to calculate the alone time, using the equation in 4.1:

$$alonetime = \frac{sozinho * 5}{60} \tag{4.1}$$

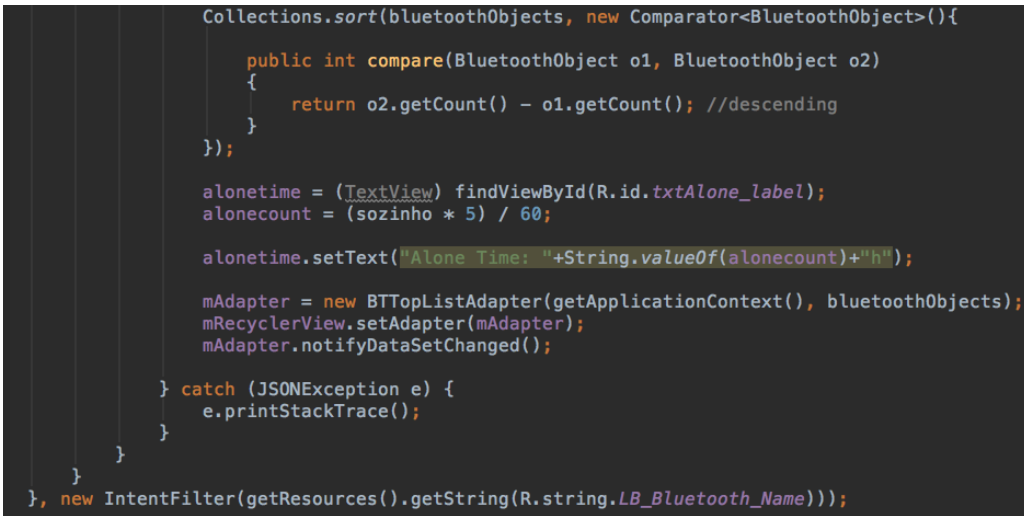
[5]Variable of type integer that will serve to count the alone time.

The variable *sozinho* is the number of empty JSONArrays that were read by the sensors. As each scan is made every 5 minutes, if we multiply the number of empty arrays by 5 we obtain the amount of time that the participant was alone, in minutes. Lastly, diving that by 60, we obtain the alone time in hours. The obtained result is transformed into a string and put in a *TextView* to appear on the screen.



```java
LocalBroadcastManager.getInstance(getApplicationContext()).registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        int sozinho = 0;

        if (intent != null) {
            try {
                ArrayList<BluetoothObject> bluetoothObjects = new ArrayList<~>();
                JSONArray jsonArray = new JSONArray(intent.getStringExtra( name: "VALUES"));
                if (jsonArray != null) {
                    if (jsonArray.length() > 0) {
                        for(int i=0;i<jsonArray.length();i++) {
                            JSONArray points = jsonArray.getJSONObject(i).getJSONArray( name: "points");
                            for (int j = 0; j < points.length(); j++) {
                                JSONObject occur = points.getJSONObject(j).getJSONObject("occur");
                                Iterator<String> it = occur.keys();
                                while (it.hasNext()) {

                                    String key = it.next();
                                    int value = occur.getInt(key);
                                    JSONArray data = new JSONArray(key);

                                    if(data.length() == 0){
                                        sozinho += value;
                                    }

                                    int check = 0; //flag para ver se há macs iguais
                                    for(int l=0; l<data.length();l++){

                                        String mac = data.getJSONObject(l).getString( name: "mac");
                                        String name = data.getJSONObject(l).getString( name: "name");

                                        for(int n=0;n<bluetoothObjects.size();n++){
                                            if(bluetoothObjects.get(n).getMac().equals(mac)){
                                                bluetoothObjects.get(n).setCount(bluetoothObjects.get(n).getCount()+value);
                                                check = 1;
                                                break;
                                            }
                                        }

                                        if(check == 0){
                                            BluetoothObject btobj = new BluetoothObject(name,mac,value);
                                            bluetoothObjects.add(btobj);
                                        }

                                    }
                                }
                            }
                        }
                    }

                }
```

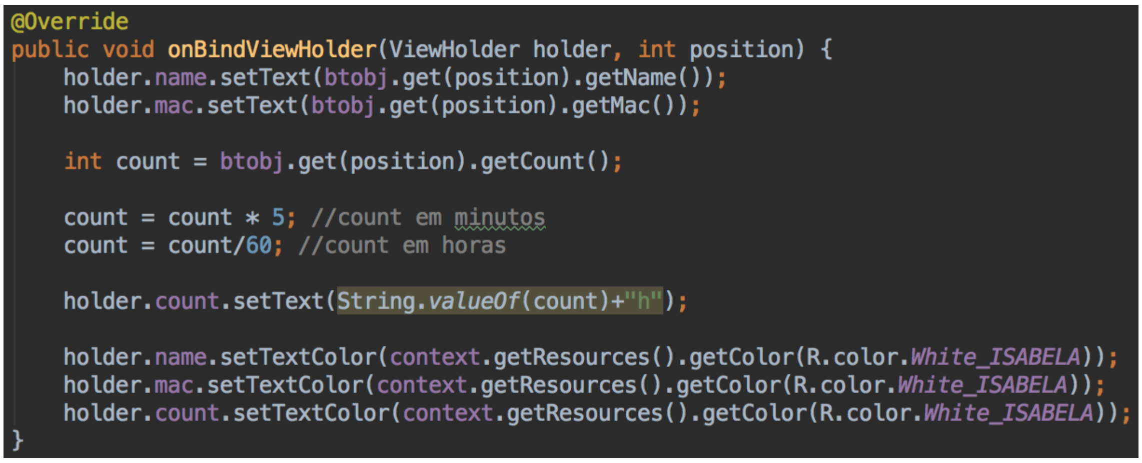
Figure 4.9: Data Treatment and Display | Part 1.



```java
                Collections.sort(bluetoothObjects, new Comparator<BluetoothObject>(){

                    public int compare(BluetoothObject o1, BluetoothObject o2)
                    {
                        return o2.getCount() - o1.getCount(); //descending
                    }
                });

                alonetime = (TextView) findViewById(R.id.txtAlone_label);
                alonecount = (sozinho * 5) / 60;

                alonetime.setText("Alone Time: "+String.valueOf(alonecount)+"h");

                mAdapter = new BTTopListAdapter(getApplicationContext(), bluetoothObjects);
                mRecyclerView.setAdapter(mAdapter);
                mAdapter.notifyDataSetChanged();

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
}, new IntentFilter(getResources().getString(R.string.LB_Bluetooth_Name)));
```

Figure 4.10: Data Treatment and Display | Part 2.

Each item, that is going to appear on the screen, is composed by three parallel *TextViews* with the name, MAC address and count (in hours) of an object of the ordered list, respectively. They are displayed through a *RecyclerView.Adapter*. The adapter has a method named *onBindViewHolder* (Fig. 4.11), to update the contents with the item at the given position [53]. The name and the MAC address are soon put in the respective *TextViews*. The count has to be put through the same calculation as the alone time (Eq. 4.1). This time, instead of the variable *sozinho*, the calculation is made with each object's "count" attribute. Then, the string value of this result is also put in its respective *TextView*.



```
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    holder.name.setText(btobj.get(position).getName());
    holder.mac.setText(btobj.get(position).getMac());

    int count = btobj.get(position).getCount();

    count = count * 5; //count em minutos
    count = count/60; //count em horas

    holder.count.setText(String.valueOf(count)+"h");

    holder.name.setTextColor(context.getResources().getColor(R.color.White_ISABELA));
    holder.mac.setTextColor(context.getResources().getColor(R.color.White_ISABELA));
    holder.count.setTextColor(context.getResources().getColor(R.color.White_ISABELA));
}
```

Figure 4.11: List Adapter.

Finally, by clicking on the option "Friends" in the navigation drawer (Fig. 4.12), the final result of the gathering and treatment of the data is displayed on the screen (Fig. 4.13) and a top of devices, around which the participant spent the most time, is obtained.

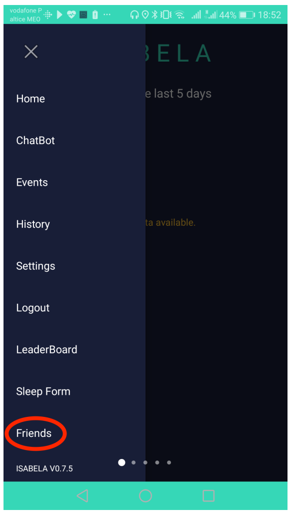
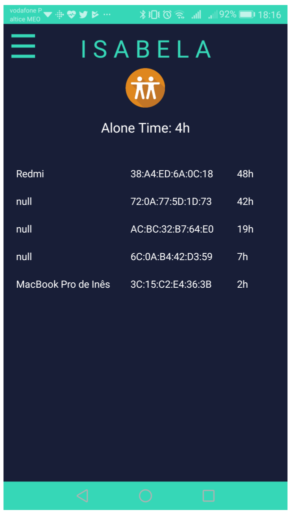


Figure 4.12: Navigation Drawer.



ISABELA

Alone Time: 4h

Redmi	38:A4:ED:6A:0C:18	48h
null	72:0A:77:5D:1D:73	42h
null	AC:BC:32:B7:64:E0	19h
null	6C:0A:B4:42:D3:59	7h
MacBook Pro de Inês	3C:15:C2:E4:36:3B	2h

Figure 4.13: Friends Screen.

4.2.4 Trial Tests

The next step was to test the system with real users. Consequently, we started by approaching some classes from DEI, from the Polytechnic Institute of Coimbra (Instituto

Superior de Contabilidade e Administração de Coimbra (ISCAC)) and from the Polytechnic School of Ecuador (Escuela Politécnica Nacional (EPN)) and presented the potential users with the idea of participating in the study.

This chapter serves to present the tests made using ISABELA in a real context, the participants and the obtained results.

4.2.4.1 Participants

Our aim was to have students from different realities and a big number of participants, therefore, the participants are from three different institutions and from various courses:

- 4 courses from DEI;
- 1 course from ISCAC;
- 1 course from EPN.

In Table 4.4, the number of participants are specified by institution.

	Girls	Boys	Total
DEI	3	4	7
ISCAC	1	4	5
EPN	8	22	30

Table 4.4: Number of Participants by Institution.

It was very difficult to recruit students at DEI and ISCAC, as it is evident by the numbers presented in the table.

All participants had to sign a consent form, in which they agreed to the terms and conditions of the project, after they read the objectives and the purpose of the collected data.

4.2.4.2 Results

While this period of tests was only supposed to be two weeks, the deadline was increased, in order to overlap with Finals Week at EPN and because, after two weeks, the amount of data collected was short. So, the period of tests began on the 13th of May and ended on the 6th of June.

The results were compiled into graphs to compare the participation of the students at each institution.

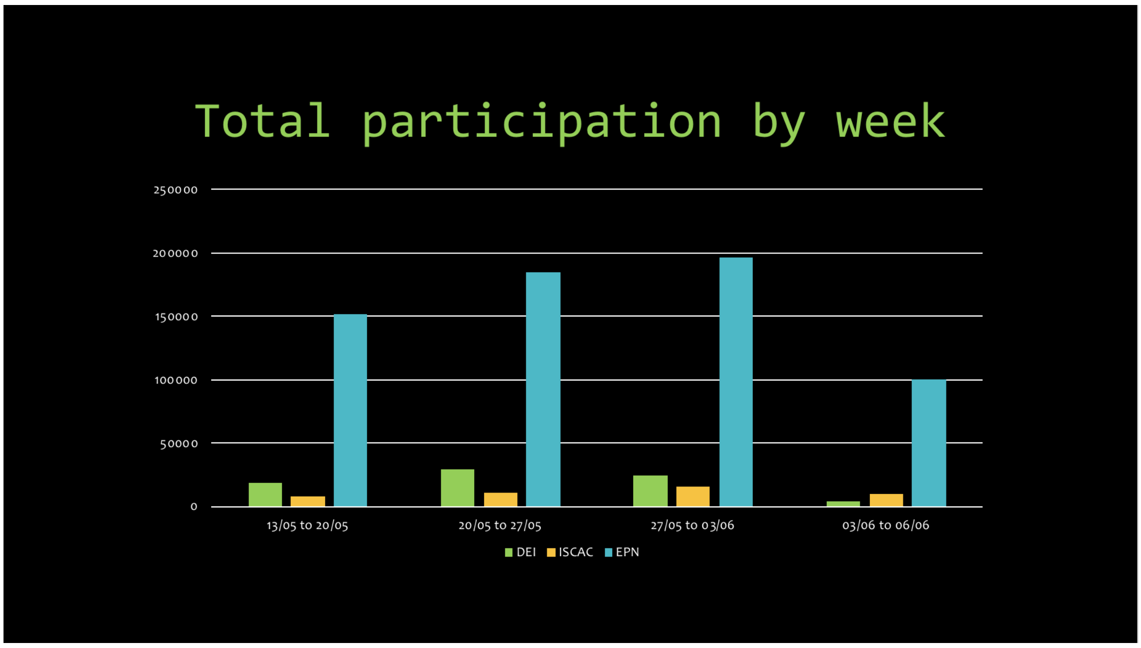


Figure 4.14: Total participation by week.

In Fig. 4.14, it is very clear that students from EPN had more participations, but this was expected, as the total number of students from EPN participating in the study is much higher than of students from DEI and ISCAC. There is also a clear reduction in participations from DEI students in the final week, as it is the week the finals started.

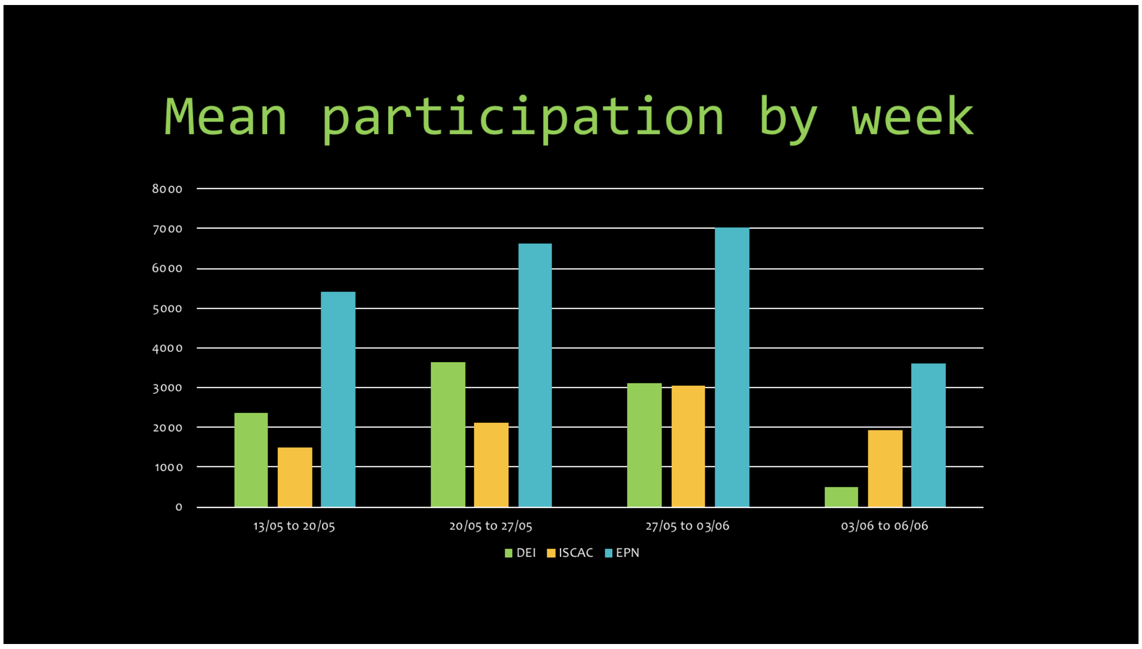


Figure 4.15: Mean participation by week.

In Fig. 4.15, the mean participation results support the total participations results, although the numbers are much closer.

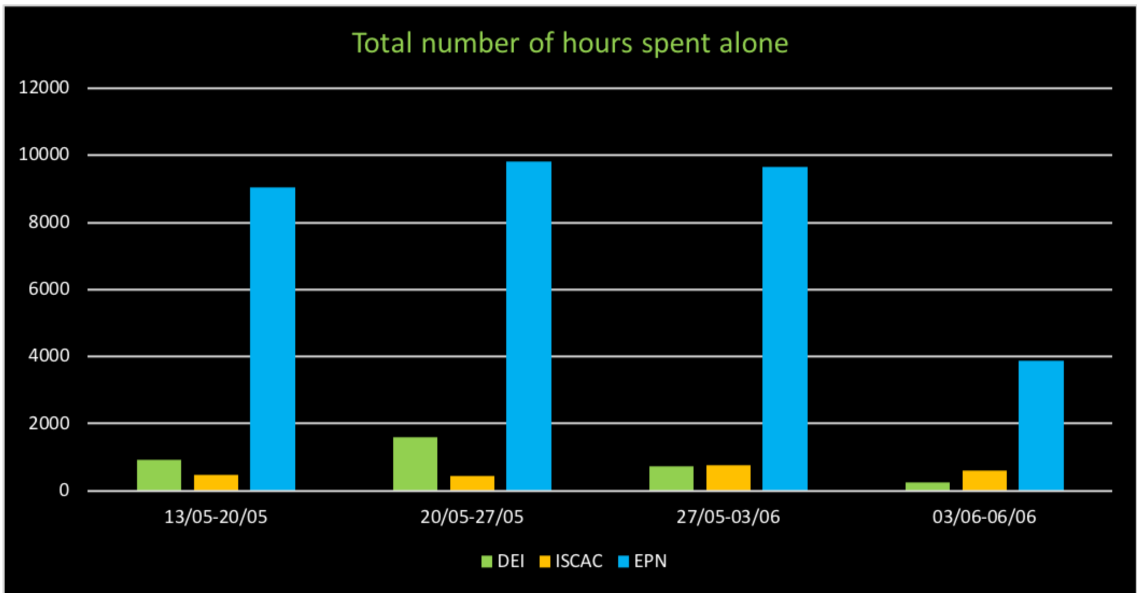


Figure 4.16: Total number of hours spent alone.

In Fig. 4.16, the tendency continues and the students from EPN have the highest total numbers of hours alone in all weeks, but that might just be a reflexion of the difference in the number of participants. Yet, contrary to the total number of participations per week, it was not on the third week that the highest numbers were achieved.

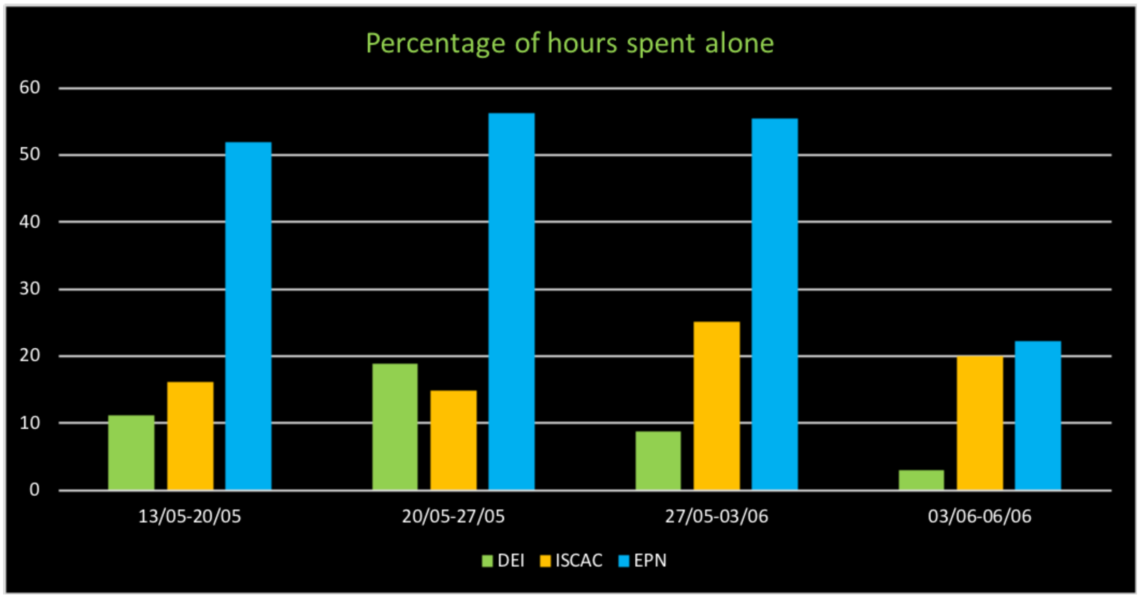


Figure 4.17: Percentage of hours spent alone.

In Fig. 4.17, it is presented the percentage of hours the participants from the various institutions spent alone, by week. This percentage is calculated through the equation in

4.2:

$$percentage = \frac{AH}{H * P} * 100 \tag{4.2}$$

AH means total number of alone hours.
H means total number of hours of the trial tests.
P means total number of participants.

As we can see in the graph, in no week and in no institution the participants spent all their time alone. But we can see that students from EPN, during the first 3 weeks of the trials tests, spent more than half of their time alone, on average.

Another thing we can take from these graphs is that the number of participations may influence the other results, as with a higher number of participations the data tends to be closer to what happen in the life of the student. Therefore, we cannot say for sure that students from DEI and ISCAC are better at socializing, because their samples of participants are much smaller than the sample of participants from EPN.

4.2.4.3 Positive Points

We can take some positive points from these early tests. For example, they lead to the finding and consequent resolution of various bugs. Thus, the application and the system became more stable and robust. These tests also helped us to the conclusion that the platform can handle these numbers of users.

Another positive point that we can take is the volume of data for analysis. With the added time to the trial, the volume of data grew and it helped us with the positive points previously mentioned.

Finally, these testes helped notice that the communication between the components of our system sometimes stopped, so my peers were able to resolve this problem.

4.2.4.4 Drawbacks

We also found some drawbacks. We can see in the participants table (Table 4.4) that few Portuguese students participated in the study. That may affect the quality and veracity of the results. Since the sample was small the results may be misleading and the impact on the academic performance on this population may not reflect the real impact.

Also, the participants had no social media presence. That means the system does not have information needed to assess the emotions of the users.

Finally, the students not always filled the sleep forms. As the sleep forms are our way to know the sleep cycles, we have no way to infer on the tiredness or good habits of sleep of the participants and, consequently, the feedback given by the platform may not be accurate.

4.3 Lightweight machine-to-machine Protocol

LWM2M is a protocol created by Open Mobile Alliance, a standards development organization.

This protocol was chosen for this project to help with device management as it was designed to mainly deal with constrained machine-to-machine devices, such as sensors. The reasons why were further explained in section 2.1.

In this section, the LWM2M protocol will be presented as it will be its setup and the integration with the existing ISABELA system.

4.3.1 Overview

LWM2M is a protocol which implements a client-server architecture and uses CoAP as the underlying transfer protocol over UDP and SMS bearers (optional). It also includes a secure channel through which the messages between the server and the client are interchanged. This level of security is provided by the Datagram Transport Layer Security (DTLS)[6] protocol [55].

The LWM2M Enabler[7] has two components: a server and a client. The server is typically located in a data center and can be hosted by the M2M, Network or Application Service Provider. The client resides in the device. It also has four interfaces between its two components: 1) Bootstrap; 2) Client Registration; 3) Device management and Service Enablement; 4) Information Reporting. The architecture of the LWM2M Enabler is represented in Fig. 4.18.

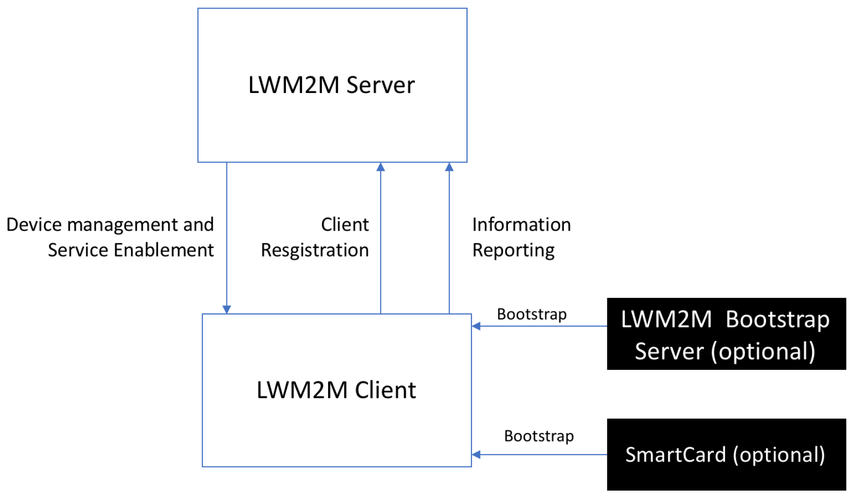


Figure 4.18: Architecture of the LWM2M Enabler [56].

The LWM2M model demmands the use of end-to-end IP connectivity between the client and the server. However, a gateway can be used when non-IP endpoints are needed, although LWM2M poses great challenges in respect to remote gateway management [21].

[6]DTLS is a protocol that secures communications between a client and sever, defined in RFC 6347 [54].
[7]Name given to the standard produced by OMA for the LWM2M solution.

Nonetheless, this protocol is widely implemented and used, thus becoming the solution for the management of constrained device with the highest acceptance of the scientific community [21]. Therefore, it was selected to be incorporated into the ISABELA project, to help manage all the physical sensors and, hopefully later, also manage the virtual sensors.

We tried two approaches to apply this protocol to our system: through the Eclipse's *Leshan* project and through an *IoT Agent*. They are going to be explained in the next sections.

4.3.2 Leshan Solution

This solution was implemented with the help of an Eclipse project called Leshan. Leshan provides libraries that help with what we wanted to do [57]. The project also provides a server and a client demonstration as an example of the Leshan API and for testing purposes.

The server of the LWM2M protocol was lodged in a virtual machine, running Ubuntu 16.04. The clients were a Raspberry Pi with a digital temperature and humidity sensor and a smartphone running Android. The server and the Raspberry Pi were connected through a switch via LAN connection and the digital sensor to the GPIO pins of the Raspberry Pi. The server and the smartphone were connected via the Wi-Fi network. The architecture of the Leshan solution is represented in Fig. 4.19.

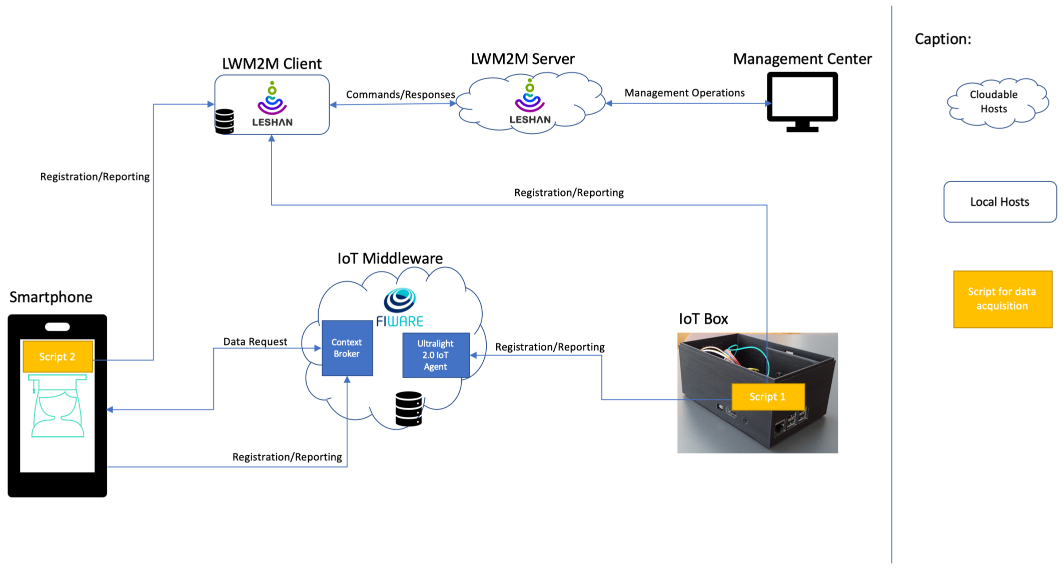


Figure 4.19: Architecture of the Leshan solution.

The configuration of the virtual machine where the server is lodged is explained in Appendix A.

4.3.2.1 Server

To launch the server, we just need to run, in Eclipse, the *leshan-server-demo* project as a Java Application and then select *LeshanServerDemo.class*.

Then, we need to open Mozilla Firefox and go to `https://localhost:8080` to open the server page (Fig. 4.20). Whenever a client device connects to the server, it appears listed on this page.

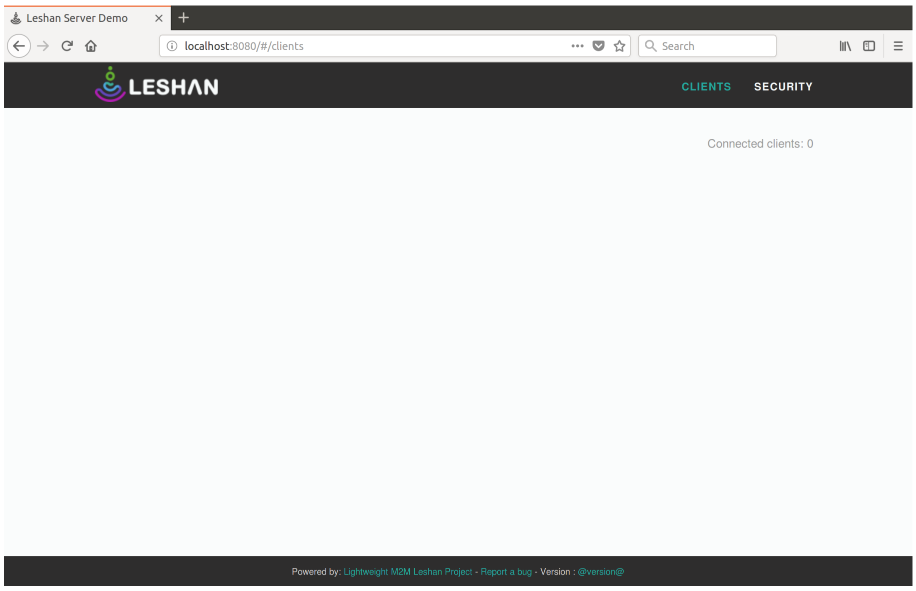


Figure 4.20: Server Page.

For the sensor's parameters and its values to appear on the device's page the server has to have files, written in .xml format, in which they are defined. For the more generic and used sensors, such as the accelerometer, these files are available in a LWM2M registry provided by OMA [58] and for the other sensors that are not so common, like the physical activity, they had to be created.

4.3.2.2 Clients

The client also needs to declare the same .xml models that the server has so that the parameters and the values sensed appear on the device's page.

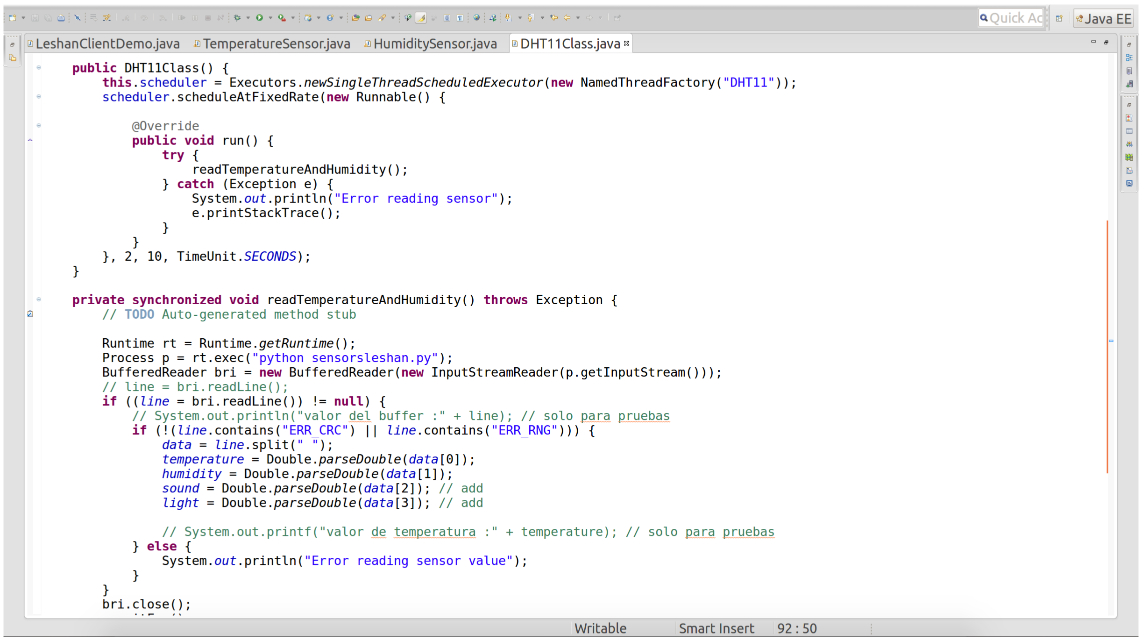
- **Raspberry Pi:**

  In order to adapt the demos to our system, some changes were made to the code provided by the Leshan project that we downloaded from its GitHub repository [59].

  First of all, the original client only had a (random) temperature sensor, so we had to prepare our client to read values of temperature and humidity and create classes for these sensors (*TemperatureSensor* and *HumiditySensor*). Then, we needed to create a class to deal with the DHT11 sensor[8] - *DHT11Class*. All sensor classes receive an object of type *DHT11Class* as parameter.

[8]Digital sensor used for temperature and humidity readings.

In ***DHT11Class*** (Fig. 4.21), the values from the sensors are read through the Python script *sensorsleshan.py*. The results returned from this script are put into a buffer and parsed into the respective variables.



```
LeshanClientDemo.java   TemperatureSensor.java   HumiditySensor.java   DHT11Class.java

    public DHT11Class() {
        this.scheduler = Executors.newSingleThreadScheduledExecutor(new NamedThreadFactory("DHT11"));
        scheduler.scheduleAtFixedRate(new Runnable() {

            @Override
            public void run() {
                try {
                    readTemperatureAndHumidity();
                } catch (Exception e) {
                    System.out.println("Error reading sensor");
                    e.printStackTrace();
                }
            }
        }, 2, 10, TimeUnit.SECONDS);
    }

    private synchronized void readTemperatureAndHumidity() throws Exception {
        // TODO Auto-generated method stub

        Runtime rt = Runtime.getRuntime();
        Process p = rt.exec("python sensorsleshan.py");
        BufferedReader bri = new BufferedReader(new InputStreamReader(p.getInputStream()));
        // line = bri.readLine();
        if ((line = bri.readLine()) != null) {
            // System.out.println("valor del buffer :" + line); // solo para pruebas
            if (!(line.contains("ERR_CRC") || line.contains("ERR_RNG"))) {
                data = line.split(" ");
                temperature = Double.parseDouble(data[0]);
                humidity = Double.parseDouble(data[1]);
                sound = Double.parseDouble(data[2]); // add
                light = Double.parseDouble(data[3]); // add

                // System.out.printf("valor de temperatura :" + temperature); // solo para pruebas
            } else {
                System.out.println("Error reading sensor value");
            }
        }
        bri.close();

Writable    Smart Insert    92 : 50
```

Figure 4.21: Part of *DHT11Class.java*.

In each sensor's class, there are variables for maximum, minimum and current values, of the parameter that is being read. We will need them to appear in the device's information page when that device connects to the server.

There are many operations that can be done through these classes. We can update the values read and we can reset and adjust the maximum and minimum measured values.
In order to update the values read, we need to declare a *DHT11Class* object. Through this object we can obtain a value, for example by invoking the function *getTemperature*.
In order to adjust the minimum and maximum measured values, a comparison between the values previously read and the current one is performed.
In order to reset the minimum and maximum measured values, these are made equal to the current values.

After the update of the code to match our system, we needed to run the client, so we used a .jar file of the *leshan-client-demo* project, with all the changes made. It was exported, in Eclipse, as a *JAVA App* and, then, a *runnable JAR file*. The .jar file was then put into the Raspberry Pi.

We needed a tool to help get the temperature and humidity values from the DHT11 sensor. We chose to install the *Adafruit Python DHT Sensor Library*, which is a Python library used to read the DHT series of humidity and temperature sensors on a Raspberry Pi [68].
First, we downloaded the library to the Raspberry Pi:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
```

Then, we made sure that our system was ready to compile Python extensions. So, we run the following commands:

```
sudo apt-get upgrade
sudo apt-get install build-essential python-dev
```

Eventually, we installed the library by running the following command, inside its folder:

```
sudo python setup.py install
```

Finally, we connected the server and the client. First we launched the server, as said in section 4.3.2.1, and, then, we opened a terminal tab inside the Raspberry Pi and run the command:

```
java -jar leshan-client-demo.jar -u <server’s ip address>:5683
```

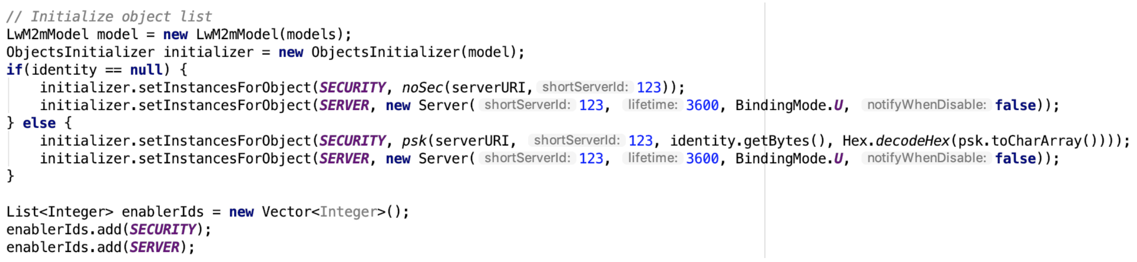
(5683 is the Leshan port.)

- **Android:**

  This part of the implementation presented some challenges. They are explained in section 6.1.

  For this part we found a demo application in a GitHub repository [60] that implemented a LWM2M Leshan client in Android. The demo included some Sensor Classes, where the values were read and sent to the server, and the Main Activity, where the LWM2M client was built with its objects and it connected to the server.

  To integrate the client in the ISABELA application, we started with the creation of a class called *LWM2MClient* that does the same that the Main Activity did in the demo application. The first change in this class was the server to which the clients were going to connect to. The demo application came with a connection to the Bootstrap Server[9] and the client needed to connect to the server we adapted for ISABELA and had running in the local machine. For this to happen, we needed to give the IP address and ID of the our server and a constructor of a Server with the same ID to the initializer of the LWM2M client, as it can be seen in Fig. 4.22, instead of passing the IP and constructor of the Bootstrap Server.



```
// Initialize object list
LwM2mModel model = new LwM2mModel(models);
ObjectsInitializer initializer = new ObjectsInitializer(model);
if(identity == null) {
    initializer.setInstancesForObject(SECURITY, noSec(serverURI, shortServerId: 123));
    initializer.setInstancesForObject(SERVER, new Server( shortServerId: 123,  lifetime: 3600, BindingMode.U,  notifyWhenDisable: false));
} else {
    initializer.setInstancesForObject(SECURITY, psk(serverURI,  shortServerId: 123, identity.getBytes(), Hex.decodeHex(psk.toCharArray())));
    initializer.setInstancesForObject(SERVER, new Server( shortServerId: 123,  lifetime: 3600, BindingMode.U,  notifyWhenDisable: false));
}

List<Integer> enablerIds = new Vector<Integer>();
enablerIds.add(SECURITY);
enablerIds.add(SERVER);
```

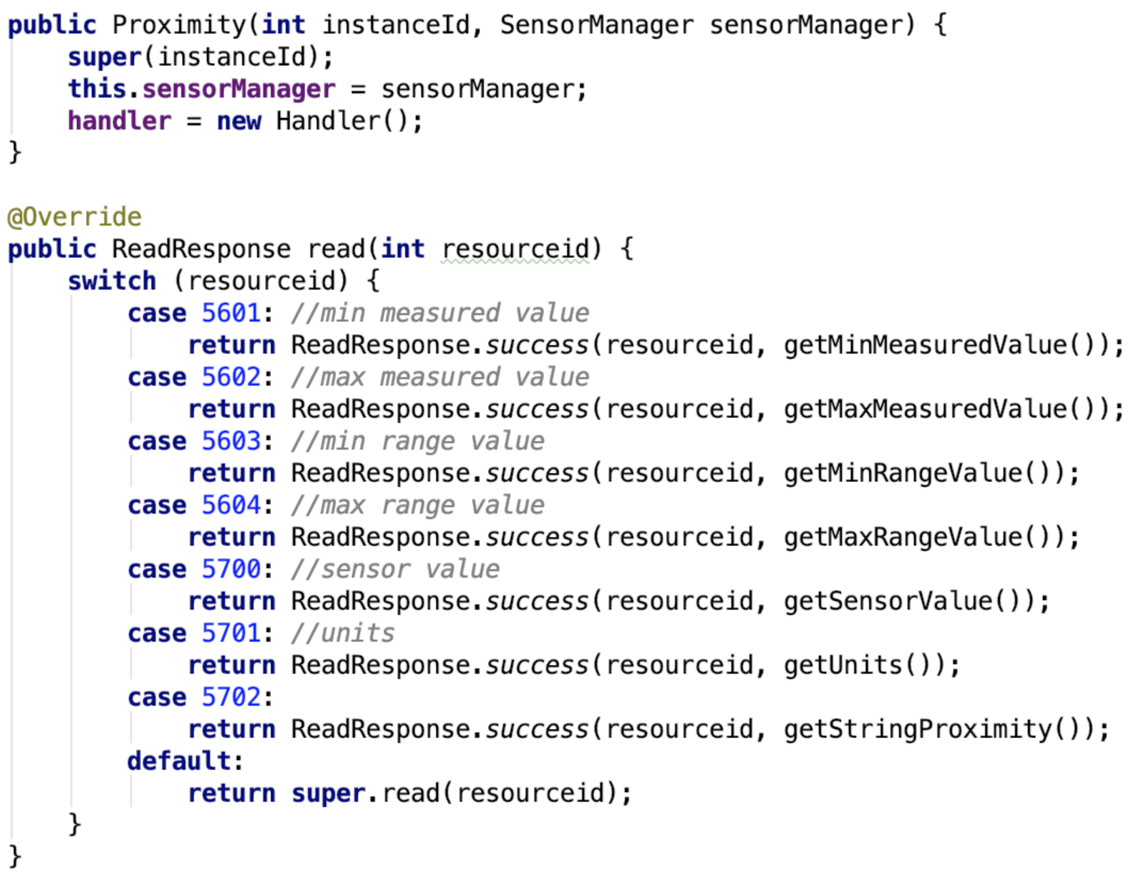
Figure 4.22: Server parameter for the LWM2M client initializer.

After that, we needed to create the classes for the sensors and the services that permitted reading the values.

Fundamentally, every Sensor class has a constructor and is divided in a *start* function, where the parameters that start the sensor are initialized, a *stop* function, where the same parameters are stopped, an *id*, where the id of that sensor is defined and has

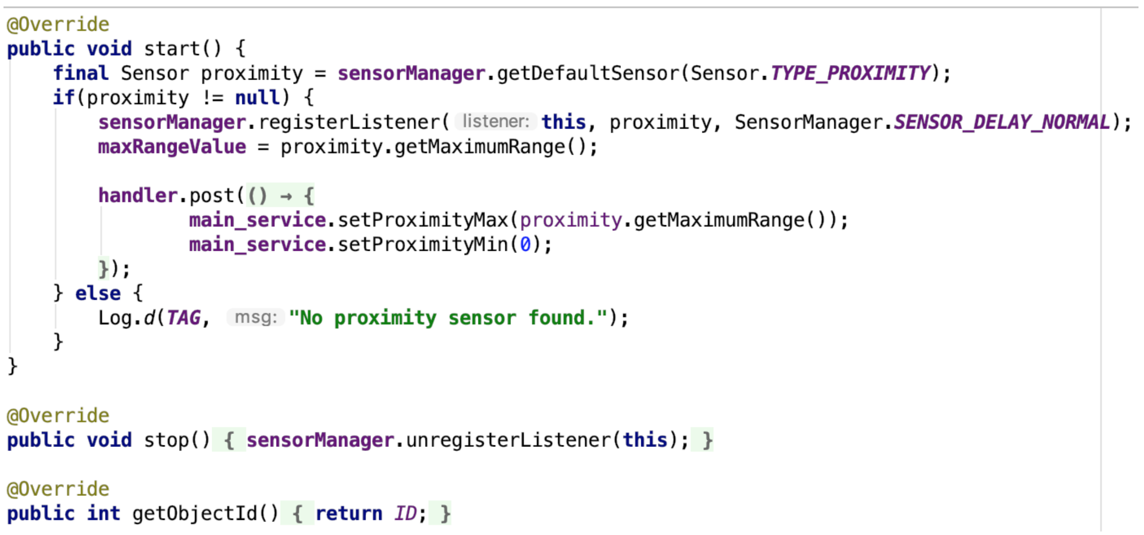
[9]Implementation offered by the project.

to correspond to the one defined in the .xml model, a function where the values are read and a *read* function that returns the values that are read and essentially updates the values in the device's page. These functions for the Proximity Sensor are represented in figs. 4.23, 4.24 and 4.25.



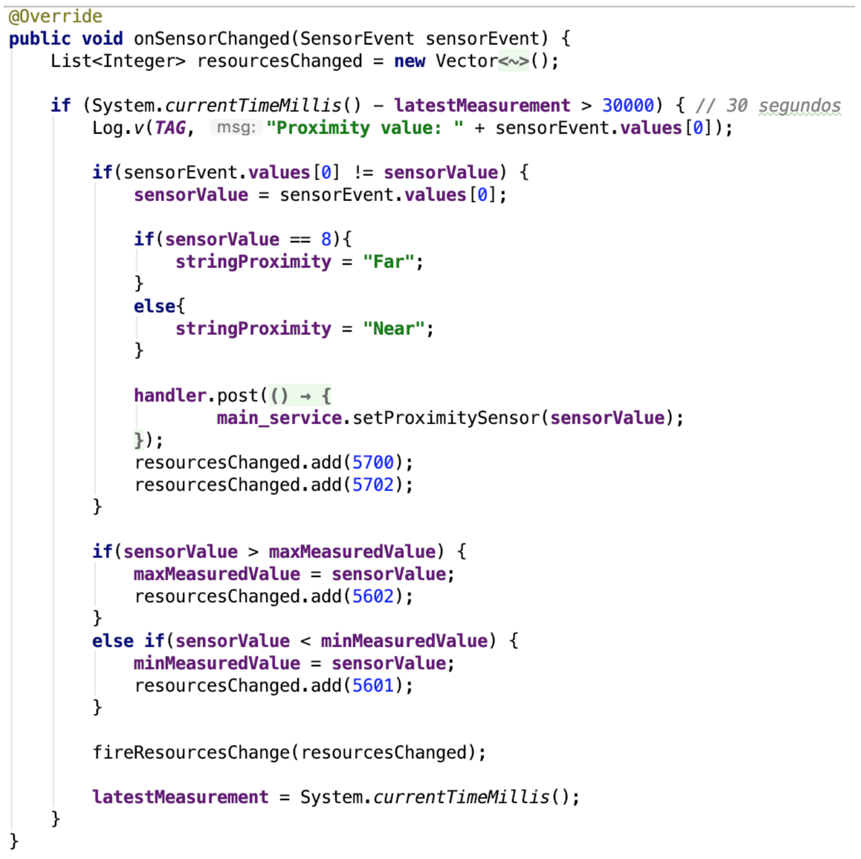
```java
public Proximity(int instanceId, SensorManager sensorManager) {
    super(instanceId);
    this.sensorManager = sensorManager;
    handler = new Handler();
}

@Override
public ReadResponse read(int resourceid) {
    switch (resourceid) {
        case 5601: //min measured value
            return ReadResponse.success(resourceid, getMinMeasuredValue());
        case 5602: //max measured value
            return ReadResponse.success(resourceid, getMaxMeasuredValue());
        case 5603: //min range value
            return ReadResponse.success(resourceid, getMinRangeValue());
        case 5604: //max range value
            return ReadResponse.success(resourceid, getMaxRangeValue());
        case 5700: //sensor value
            return ReadResponse.success(resourceid, getSensorValue());
        case 5701: //units
            return ReadResponse.success(resourceid, getUnits());
        case 5702:
            return ReadResponse.success(resourceid, getStringProximity());
        default:
            return super.read(resourceid);
    }
}
```

Figure 4.23: Constructor and Read functions.



```java
@Override
public void start() {
    final Sensor proximity = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
    if(proximity != null) {
        sensorManager.registerListener( listener: this, proximity, SensorManager.SENSOR_DELAY_NORMAL);
        maxRangeValue = proximity.getMaximumRange();

        handler.post(() → {
                main_service.setProximityMax(proximity.getMaximumRange());
                main_service.setProximityMin(0);
        });
    } else {
        Log.d(TAG,  msg: "No proximity sensor found.");
    }
}

@Override
public void stop() { sensorManager.unregisterListener(this); }

@Override
public int getObjectId() { return ID; }
```

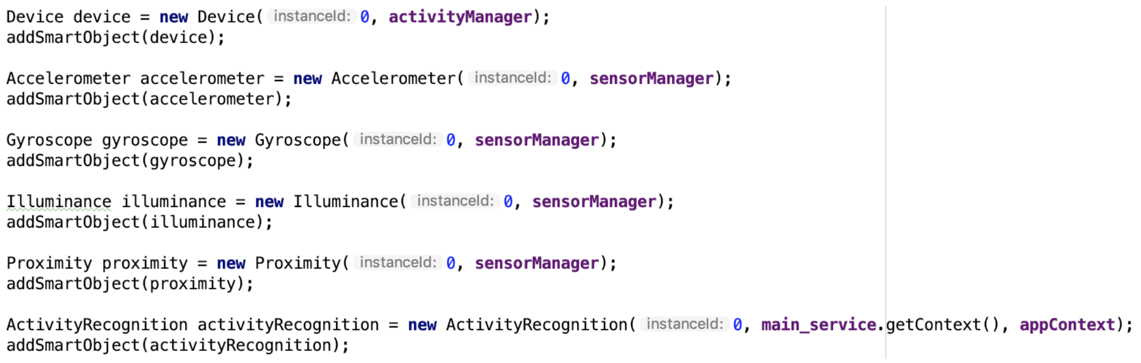
Figure 4.24: Start, Stop and ID functions.



```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    List<Integer> resourcesChanged = new Vector<~>();

    if (System.currentTimeMillis() - latestMeasurement > 30000) { // 30 segundos
        Log.v(TAG,  msg: "Proximity value: " + sensorEvent.values[0]);

        if(sensorEvent.values[0] != sensorValue) {
            sensorValue = sensorEvent.values[0];

            if(sensorValue == 8){
                stringProximity = "Far";
            }
            else{
                stringProximity = "Near";
            }

            handler.post(() → {
                    main_service.setProximitySensor(sensorValue);
            });
            resourcesChanged.add(5700);
            resourcesChanged.add(5702);
        }

        if(sensorValue > maxMeasuredValue) {
            maxMeasuredValue = sensorValue;
            resourcesChanged.add(5602);
        }
        else if(sensorValue < minMeasuredValue) {
            minMeasuredValue = sensorValue;
            resourcesChanged.add(5601);
        }

        fireResourcesChange(resourcesChanged);

        latestMeasurement = System.currentTimeMillis();
    }
}
```

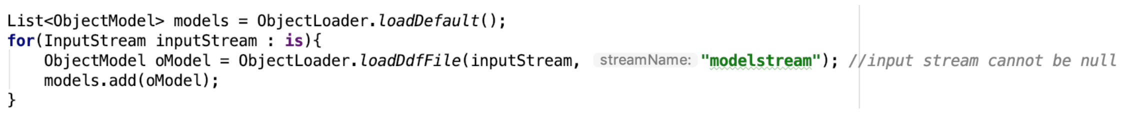
Figure 4.25: Function where the values are read.

Each one of these sensors are started in the *LWM2MClient* and added to the initializer of the LWM2M client along with its .xml models (Figs. 4.26, 4.27 & 4.28). That is why their parameters appear on the device's page on the server side.



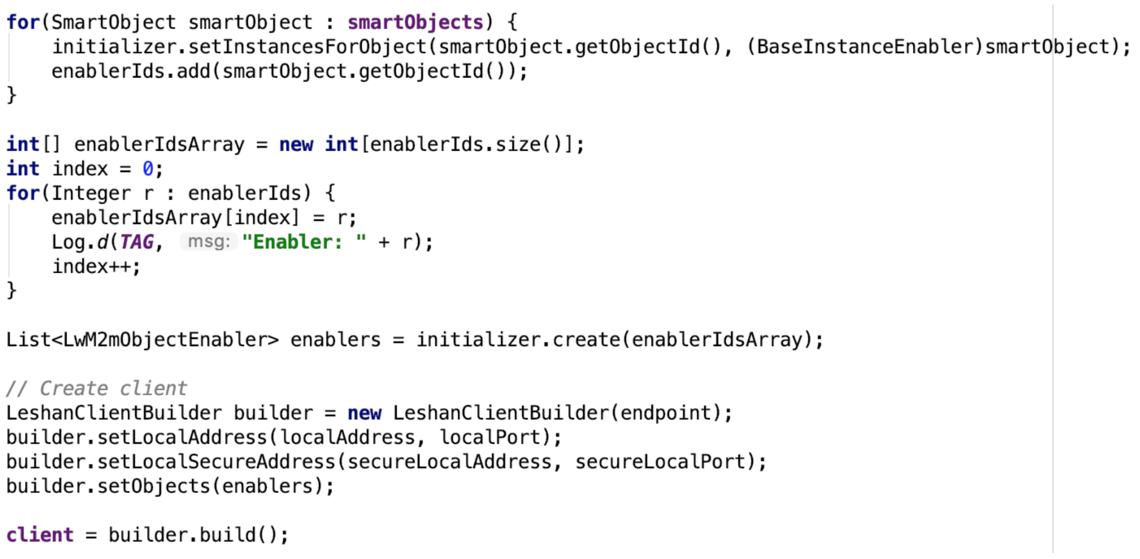
```
Device device = new Device( instanceId: 0, activityManager);
addSmartObject(device);

Accelerometer accelerometer = new Accelerometer( instanceId: 0, sensorManager);
addSmartObject(accelerometer);

Gyroscope gyroscope = new Gyroscope( instanceId: 0, sensorManager);
addSmartObject(gyroscope);

Illuminance illuminance = new Illuminance( instanceId: 0, sensorManager);
addSmartObject(illuminance);

Proximity proximity = new Proximity( instanceId: 0, sensorManager);
addSmartObject(proximity);

ActivityRecognition activityRecognition = new ActivityRecognition( instanceId: 0, main_service.getContext(), appContext);
addSmartObject(activityRecognition);
```

Figure 4.26: Creating the Sensors and adding them to SmartObjects.



```
List<ObjectModel> models = ObjectLoader.loadDefault();
for(InputStream inputStream : is){
    ObjectModel oModel = ObjectLoader.loadDdfFile(inputStream,  streamName: "modelstream"); //input stream cannot be null
    models.add(oModel);
}
```

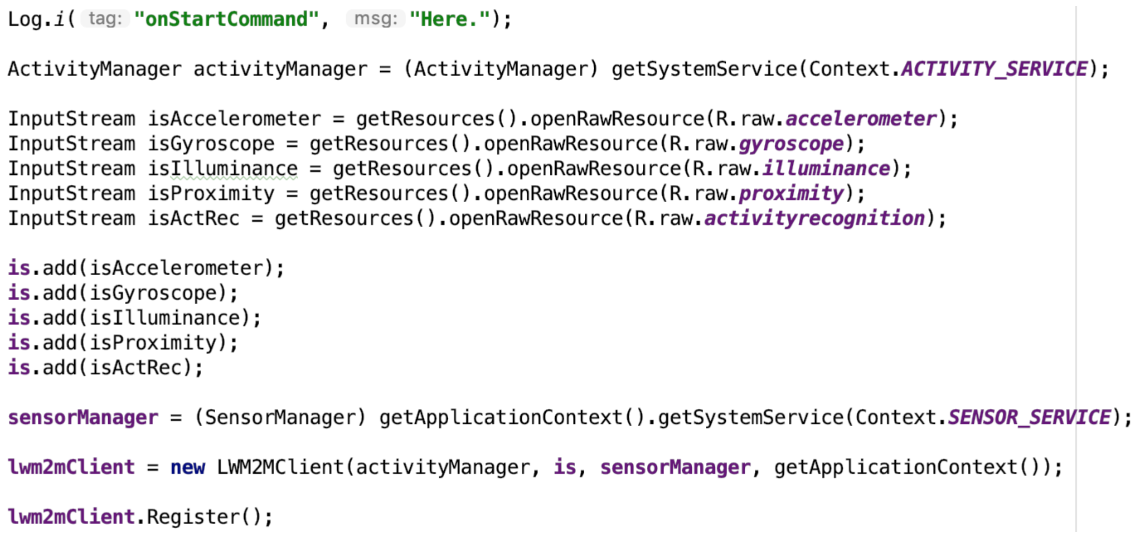
Figure 4.27: Loading the .xml Models.



```
for(SmartObject smartObject : smartObjects) {
    initializer.setInstancesForObject(smartObject.getObjectId(), (BaseInstanceEnabler)smartObject);
    enablerIds.add(smartObject.getObjectId());
}

int[] enablerIdsArray = new int[enablerIds.size()];
int index = 0;
for(Integer r : enablerIds) {
    enablerIdsArray[index] = r;
    Log.d(TAG,  msg: "Enabler: " + r);
    index++;
}

List<LwM2mObjectEnabler> enablers = initializer.create(enablerIdsArray);

// Create client
LeshanClientBuilder builder = new LeshanClientBuilder(endpoint);
builder.setLocalAddress(localAddress, localPort);
builder.setLocalSecureAddress(secureLocalAddress, secureLocalPort);
builder.setObjects(enablers);

client = builder.build();
```

Figure 4.28: Building the client with the Models and the SmartObjects.

An instance of the *LWM2MClient* is created in the *Main_Service* so that it creates the client, connects to the server and starts sensing as soon as the application starts.

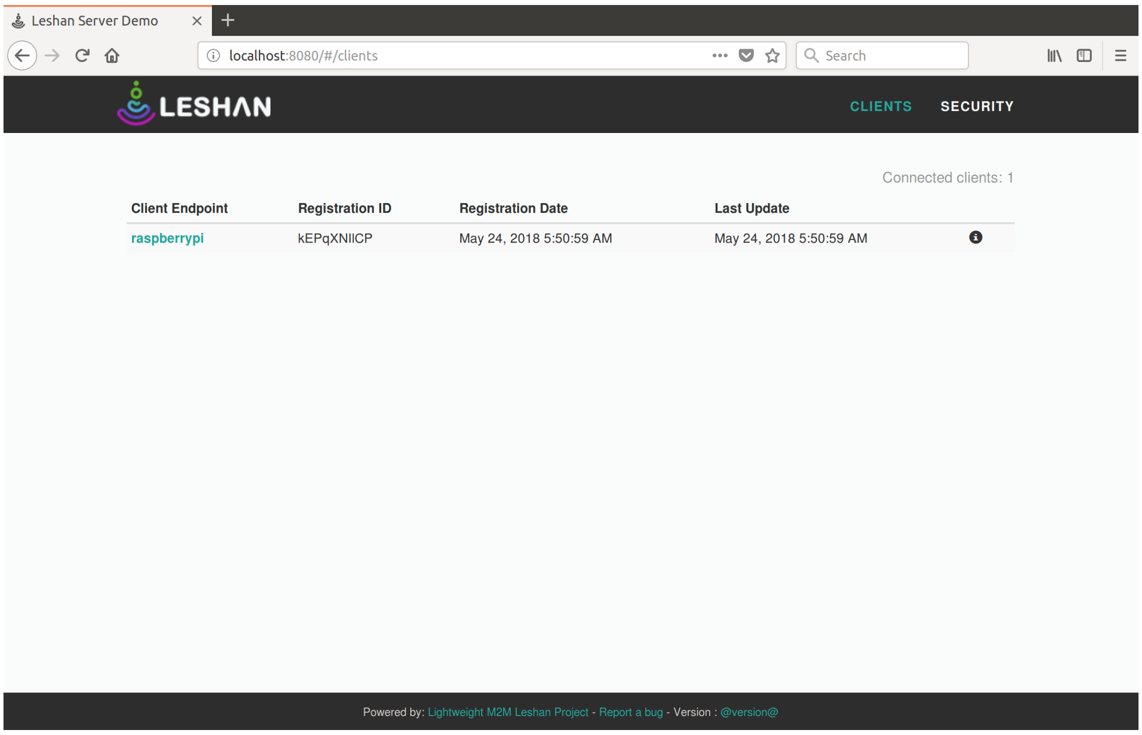


```
Log.i( tag: "onStartCommand",  msg: "Here.");

ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);

InputStream isAccelerometer = getResources().openRawResource(R.raw.accelerometer);
InputStream isGyroscope = getResources().openRawResource(R.raw.gyroscope);
InputStream isIlluminance = getResources().openRawResource(R.raw.illuminance);
InputStream isProximity = getResources().openRawResource(R.raw.proximity);
InputStream isActRec = getResources().openRawResource(R.raw.activityrecognition);

is.add(isAccelerometer);
is.add(isGyroscope);
is.add(isIlluminance);
is.add(isProximity);
is.add(isActRec);

sensorManager = (SensorManager) getApplicationContext().getSystemService(Context.SENSOR_SERVICE);

lwm2mClient = new LWM2MClient(activityManager, is, sensorManager, getApplicationContext());

lwm2mClient.Register();
```

Figure 4.29: Instance of a *LWM2MClient* being called on the Main_Service.

Disclaimer: Before applying the changes directly in ISABELA, I adapted the demo application to connect to the server we had and created the classes of the sensors we needed and tested from that.

When the connection is established, the device appears on the server page (Fig. 4.30).



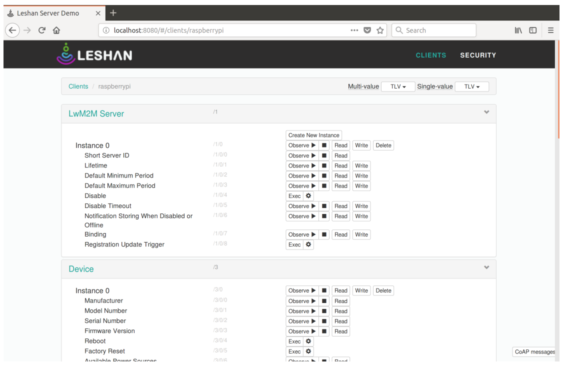
Connected clients: 1

Client Endpoint	Registration ID	Registration Date	Last Update
raspberrypi	kEPqXNIICP	May 24, 2018 5:50:59 AM	May 24, 2018 5:50:59 AM

Powered by: Lightweight M2M Leshan Project - Report a bug - Version : @version@

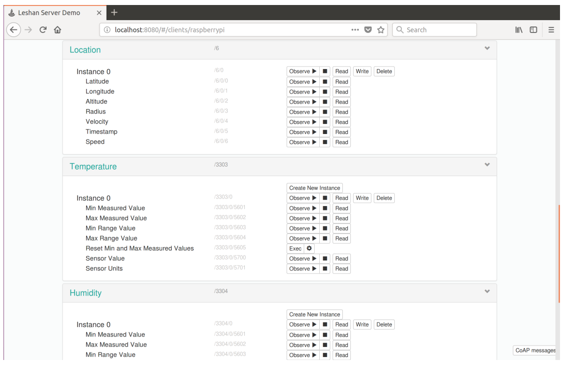
Figure 4.30: List of Devices connected to the Server.

Finally, by clicking on the device, we have access to its information (Figs. 4.31 & 4.32).



LwM2M Server	/1
Instance 0	/1/0
Short Server ID	/1/0/0
Lifetime	/1/0/1
Default Minimum Period	/1/0/2
Default Maximum Period	/1/0/3
Disable	/1/0/4
Disable Timeout	/1/0/5
Notification Storing When Disabled or Offline	/1/0/6
Binding	/1/0/7
Registration Update Trigger	/1/0/8

Device	/3
Instance 0	/3/0
Manufacturer	/3/0/0
Model Number	/3/0/1
Serial Number	/3/0/2
Firmware Version	/3/0/3
Reboot	/3/0/4
Factory Reset	/3/0/5



Location	/6
Instance 0	/6/0
Latitude	/6/0/0
Longitude	/6/0/1
Altitude	/6/0/2
Radius	/6/0/3
Velocity	/6/0/4
Timestamp	/6/0/5
Speed	/6/0/6

Temperature	/3303
Instance 0	/3303/0
Min Measured Value	/3303/0/5601
Max Measured Value	/3303/0/5602
Min Range Value	/3303/0/5603
Max Range Value	/3303/0/5604
Reset Min and Max Measured Values	/3303/0/5605
Sensor Value	/3303/0/5700
Sensor Units	/3303/0/5701

Humidity	/3304
Instance 0	/3304/0
Min Measured Value	/3304/0/5601
Max Measured Value	/3304/0/5602
Min Range Value	/3304/0/5603

Figure 4.31: Device's Properties | Part 1. Figure 4.32: Device's Properties | Part 2.

4.3.3 IoT Agent Solution

This approach consisted on integrating an LWM2M IoT agent in FIWARE to serve as a bridge of communication between the devices that use the LWM2M protocol and the Context Broker (ORION module) [61].

The *Lightweight M2M IoT Agent* is a standard FIWARE IoT Agent based in the public Node.js IoT Agent Library. The Agent described in that library is a component that facilitates the management and control of the information of a group of devices from a FIWARE NGSI Context Broker using their own native protocols.

These agents support various types of features [62]. They are:

- **Device Registration:** all the devices connected to the agent are mapped to a Context Broker entity.

- **Device Information Update:** when there are new values for an attribute, the devices should send the information to the agent and the agent should send a request to the Context Broker to update the devices entity.

- **Device Management:** the agent should offer a repository where the devices are registered and that holds information needed for the connection to the Context Broker.

- **Device Provisioning:** the agent should offer an external way for the user to be able to customize the device's entity name, type and information.

- **Type Configuration:** the agent should provide default values to the device attributes based on its type.

Devices measures can have three different behaviors:

- **Commands:** an attribute is set on the Context Broker entity and the agent will be responsible of contacting the device to perform the command itself, updating as soon as it has the information.

- **Lazy Attributes:** the sensors will wait for the agent to request data.

- **Active Attributes:** the sensed values are pushed from the device to the agent, that will request an update to the Context Broker with that information.

An implementation of this agent is given by *Telefónica Investigación y Desarrollo*[10] and is available in a GitHub repository [63]. *Telefónica* also offers a library that aims to be a simple way to build LWM2M servers and clients with Node.js [64] and we adapted it to fit the devices of our system.

Here, the server and the smartphone were also connected via the Wi-Fi network. The architecture of the IoT Agent solution is represented in Fig. 4.33.

[10]Spanish telecommunications company.

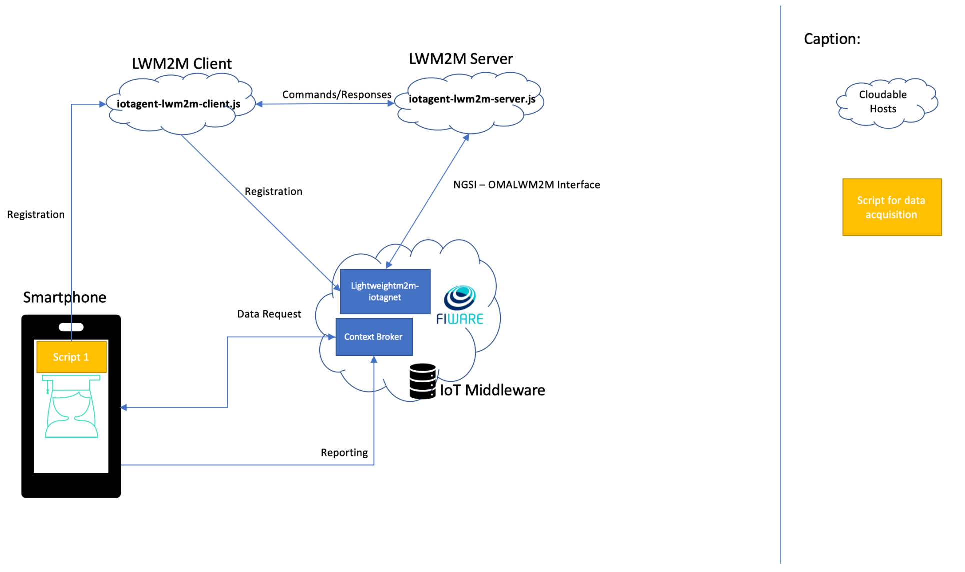


Figure 4.33: Architecture of the IoT Agent solution.

The configuration of the FIWARE environment is explained in Appendix B.

4.3.3.1 LWM2M IoT Agent

As previously said, the IDAS module is the one that handles communications with the devices through its IoT agents. So, its configuration was made to implement the *LightweightM2M IoT Agent*.

This agent uses the features provided by the Node.js IoT Agent Library with some adaptations to the LWM2M protocol, as is the case of the Mappings. For LWM2M mapping can be:

1. OMA Registry objects and resources from their URIs to their common names.
2. Custom device objects to the names defined by the user.

To accomplish that, the agent supports:

- *lwm2mResourceMapping*: an additional property that lets the user costumize the names for particular resources.
- *omaRegistry.json*: contains the OMA Registry previously mentioned and is used for automatic mappings in case there are not custom ones.

No alterations were made to the agent files for our work.

The server and the client are going to be explained in the next sections.

4.3.3.2 Server

The implementation of this server was based on the one provided by *Telefónica* [64] and the only changes we made were to add the customized models of the sensors that are not part of the OMA Registry.

The agent calls up the server in its implementation so it starts running when the FIWARE containers are started.

4.3.3.3 Client

As was the case for the Leshan client, this client had to be adapted to run on Android [64], as it was written in a different language.

For this, I made use of one of the agent's features - the provisioning - and created a script were I defined an Entity, its type and attributes. Then, in the same file, I mapped the LWM2M resources according to the .xml models of the sensors.

In the class *LWM2MClient*, the endpoint name was made to correspond to the one I had given to the Entity in the provisioning and the connection was made to the IP and port of the server, running in the Docker.

Disclaimer: This approach was not fully implemented. Although the server received connections from the client, it was not being registered in ORION and the readings did not appear. The challenges that appeared during this part of the work are explained in section 6.1.

In conclusion, the implementation of the Proximity Module served its purpose, as I got to know the system I was working with a little better, and the implementation of the IoT Agent solution posed more challenges than the *Leshan* solution, as it is going to be explained with greater detail in another section. Nevertheless, it can be said that the two main objectives defined for this thesis were accomplished.

Chapter 5

Performance Tests and Results

The majority of the work was focused on the enrichment of the ISABELA system. As part of that, a way to manage all the heterogeneous devices that make up the system was implemented. With the incorporation of a new protocol some questions about the performance of the system were raised, therefore some tests in respect to the performance of the system were performed and are presented in this chapter.

5.1 Battery Life

Battery life is a very important aspect that concerns and, potentially, limits a software's performance. So, when creating new software or integrating new functionalities, we must be sure that this does not affect the consume of battery of the devices that run the software. Also because if an application drains the battery of a phone too quickly, the user is probably going to uninstall said application. Therefore, in this project, we performed tests on the battery life of the devices that run our application.

For that we used a tool, developed by Google, named *Battery Historian*. This tool creates reports from the usage of a smartphone while it is not plugged in and allows the visualization of system events and statistics. It also allows the selection of a specific application and subsequent inspection of the metrics that affect the battery life of said application [69].

To use it, we had to install Battery Historian in a computer via *Docker*[1], as it is the easiest and fastest way, download the latest stable image and run it on a computer terminal with the following command:

```
docker run -p <port_number>:9999 gcr.io/android-battery-historian:3.0 --port
9999
```

On the browser we can see if it is running by typing `http://localhost:<port_number>`. The *port_number* is specified by the user.

Now that a way to visualize the data is installed and running, we needed a way to collect relevant data, so we used *Batterystats*[2]. For this, we had to connect the smartphone to a computer and do a reset of its battery stats, so that later we could have only the

[1]Docker is a tool, created by Docker Inc., that uses containers to make it easier to deploy and run applications [70].

[2]Batterystats is a file found on Android phones that contains states and information regarding the usage of battery by applications or the hardware [71].

information of the time interval that we wanted. After the reset, we disconnected the phone from the power source and left it to run the application. When the time intervals were fulfilled, we connect the smartphone to the computer again so we could create the files to be read with Battery Historian. To create the reports from raw data, we run the following line on the terminal:

```
adb bugreport > [path/]bugreport.zip
```

Then, we just had to go to the browser in which the Battery Historian was running and upload the report file and visualize the results.

The tests and results obtained from the data gathered are going to be presented and analyzed in the next sections.

5.2 Tests

The tests were focused on the impact of the LWM2M protocol on the battery performance of the Android ISABELA application. For these tests, we let the ISABELA application run for two days, using it normally to fill the sleep forms and view some information, and used the Batterystats, as explained before. Two situations were tested:

1. Only running the ISABELA application.
2. Running the ISABELA application with the LWM2M client (Leshan solution).

Additionally, other tests were performed to evaluate the impact of the interval of time between readings on the battery use. For these, we used the demo application that implemented the LWM2M client in Android with the changes I made before applying them to ISABELA.

These tests were made for 10, 20, 30, 40, 50 and 60 seconds in time periods of 6 hours each.

5.3 Results

After running the tests and analyzing the data that resulted with Battery Historian, the percentage of battery use for each of the systems previously mentioned is shown in the following table:

TESTED SYSTEM	BATTERY USE
1. ISABELA (only)	4,83%
2. ISABELA + LWM2M	7,50%

Table 5.1: Battery use for each system.

As expected, system 1 consumes less energy than system 2, as system 2 spends energy with the ISABELA parts and on top of it has the LWM2M client parts too. Because there

is a duplication of information in the Leshan solution, it does not go through the same route, as it does in the IoT Agent solution. Therefore, we thought it was best to test the system with the worst case scenario in terms of performance, because if the results were favorable for that one, they would be favorable for the best one.
Finally, as we can see on the table, the percentage of battery use of system 2 is not significantly higher than the percentage of battery use of system 1.

Furthermore, in terms of other important performance metrics, for example the CPU Usage, the values are the same or, at least, less different between them than the battery use ones.

As a result, we can conclude that by adding the LWM2M component to the ISABELA Android application we are not sacrificing its performance.

For the application running only the LWM2M client the results are shown in the following graph:

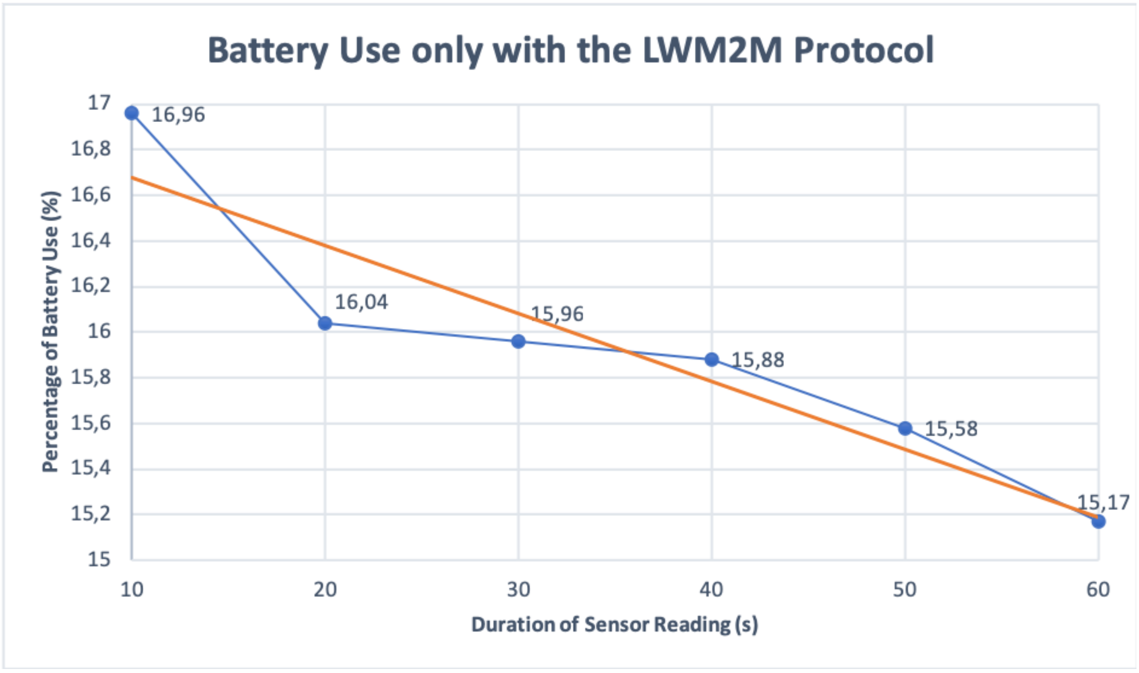


Figure 5.1: Battery Use only with the LWM2M Protocol.

As it was expected, the value of the percentage of battery use decreases as the time interval increases. The system has to perform less actions in the same amount of total time. Furthermore, as we can see, the values of the percentage of battery use are very close, not having a difference bigger than, approximately, 2% between the outermost values.

The ISABELA application collects data in a time interval of 30 seconds. As the values do not vary much, we can state that this time interval is not significantly more battery consuming than the bigger intervals considered, so it can be deemed appropriate.

In conclusion, the results were what it was expected and we became one step closer to conclude that the addition of this new feature is not prejudicial in terms of performance.

This page is intentionally left blank.

Chapter 6

Conclusions and Future Work

This semester the work carried out was a continuation of the work started last semester. Since July I continued studying the ISABELA system in order to find the best way to integrate the LWM2M protocol, which was the work I did this semester.

Some of the challenges faced through the development of this thesis, the conclusions reached at the end and the work left to be done are going to be the focus of this chapter.

6.1 Challenges

The integration of the LWM2M protocol with ISABELA posed some great and time consuming challenges.

The first challenge that I faced was the implementation of a Leshan LWM2M client for Android. The LWM2M client that we had for our system was implemented in Java and we needed it to run on Android. The first idea was to run the .jar file from the Main Service of the ISABELA application, as the Raspberry Pi, but after some research I realized that that was not the path and started searching for some applications that implemented a LWM2M client for Android.

After awhile of not finding what we were looking for, we finally found one that implemented a version of a LWM2M client for Leshan. With that, I started adapting it and creating objects of the sensors we needed values from. In this part I run into some trouble. First I could not connect the client to the server that we had running in the local machine, because the demo application was not configured to work with a server like ours. After some research, I figured it out and understood how to solve it and the solution was very simple, so I could move on to creating the classes for the sensors. I took the ones that were implemented in the demo and started emulating that approach, but when I run it the values were being sensed, because they appeared in the logs, but were not passing through to the server. Once again I had to do some research to see what I was doing wrong in sending the values. The problem was that I was not sending them through the right parameters ID that were defined in the .xml models. After solving this, I created the rest of the sensor classes and moved on to the next part of the project.

The next challenge came with having to work with FIWARE. The integration of different protocols with FIWARE comes through specific agents in the IDAS module. But I had never worked with it, so I had to spend some time doing a tutorial, prepared by a colleague, and studying the technology.

After this, I did some research about the LWM2M protocol and FIWARE and ways of integrating it. An LWM2M IoT Agent came up, FIWARE offers its implementation and also the implementations of a server and a client in Node.js. The different language posed another challenge, I had never worked with Node.js and in the other approach of integration of the LWM2M protocol that we tried for this project the server and the client were written in Java or similar. Even so, we configured the agent with our FIWARE modules and I tried to take advantage of the provisioning functionality and created a script to register an Entity in the ORION module so that the values read from the smartphone once it connected could be stored there. Another challenge appeared because when I tried to connect them the device could not be registered. After awhile I tried to connect the client with a non-existing entity name and it worked - the entity appeared in ORION but it had no values. For this reason, the next challenge was to adapt, once again, the LWM2M client on Android to work with the Node.js implementation instead of the Leshan one. Because these challenges were so time consuming, I could not finish the adaptation of the client and, therefore, the integration of the LWM2M protocol through this approach.

All in all, the errors were very hard to surpass and they took time to understand as I had never worked with this protocol or a similar one and had to study its implementation and configuration at the same time that I did it.

6.2 Conclusions

As the basis of this project is the monitorization of the surroundings of students in favor of improving their academic performances, we are dealing with a system that is composed by a lot of different sensors that are incorporated in different physical devices. With this in mind, we tried to explore the concept of management of a network of heterogeneous devices and evaluate the path to follow taking into account the components that already existed in the system.

For this, we researched and studied the protocols that might fit what we were aiming to accomplish and, although some of the options were recent and there was not a lot of information of its use in cyber-physical systems, we thought they were the best suited regarding the scope of the project and the type of technology and devices used, including their limitations. Therefore, we were able to adapt and incorporate a network management protocol with clients for Android and Raspberry Pi.

However, we can conclude that there is a long way to go and that each approach that we took for the integration of the LWM2M protocol has its limitations.

As a matter of fact, the *Leshan* implementation has reading management capabilities for both Android and stationary devices [72], but as it is:

- There is a duplication of the information. The same information that is sent to the LWM2M server is sent through another route to the FIWARE's context broker.
- There is absence of execute-write-delete capabilities.
- The values in the LWM2M server are only stored until there is a new reading that replaces the older one.
- There is no capability of automatic group device provisioning.

Even so this approach offers a simple way to see the each device that is connected, their attributes and sensed values through its interface.

The *IoT Agent* implementation was considered to be the solution to some of the limitations that the Leshan implementation offers. For example, it is a way of implementing a bridge between the LWM2M protocol and the system's components (FIWARE). So, it obliterates the duplication of information, as the information passes through the IoT agent to the server and the context broker directly. It also offers the capability of group device provisioning through running a script and the storage of information in a database.

On one hand, the existence of a tool, developed by the same corporation that developed the data center that we chose for our project, which gave answers to some of the limitations of other existent tools was of great help because the integration was a smoother process.

But, on the other hand, the fact that it implemented the server and the client in a complete different way and language posed as a very hard challenge in and of itself. Because the challenges that appeared were so time consuming, I could not finish the incorporation through this approach.

As I could learn, the time that I had to work with the protocol was not sufficient to benefit from its capabilities to the fullest. In retrospect, I could have saved time if I had started with the integration of the server in FIWARE, as I proposed on the midterm report, but this way I learned another implementation of this protocol and different technologies.

Even so, we could conclude that the incorporation of a protocol like this did not affect significantly the performance of this system. So, systems like this that are composed by heterogeneous devices and depend on them working to give accurate and real advice only have to gain if they can have a protocol that manages and controls them.

Finally, the two main objectives for this thesis were accomplished. Learning the system and the technologies and components that were used on it through the implementation of a module and the integration of a protocol was very valuable for me. It helped me emerge in the world of Technologies of Information and gain new skills that I did not have during my academic course. It, also, permitted me to be a co-author of the scientific paper that appears in Appendix C.

6.3 Future Work

Throughout this thesis I mentioned that this work could improve in many ways. Whether it was the better use of some technology's functionalities, an approach that still does not quite reach what we want from it or even work that is meant to be done in the future.

The proximity module is a good tool to infer sociability, but raises its limitations when we think about the fact that it gives values of the devices the smartphone is around more often and we are assuming it is always a person, but not always. Nowadays, a lot of everyday devices have bluetooth connections, such as car radios. So, this module is not 100% accurate and can be a point of betterment.

The previous point leads me to the need to correlate the results that we gather with the actual academic performance of the students to see until which degree we are doing a fair assessment, what needs improvement and what other factors that we may not be taking into account we should examine.

Another point of focus for some future work may be the addition of write-execute-delete fucntionalities to the management protocol. With these abilities implemented we may be capable of fully controlling a sensor or, at least, see if they are working correctly without having to touch them directly.

For now, the management protocol only has the physical sensors in its realm, even though this project is also made up of another class of sensors as are the social ones. A step further would be to open up the managing capabilities to the social sensors.

As I said, there were some major challenges throughout the project and I could not complete the adaptation of the LWM2M client for the IoT Agent approach. Therefore, finishing this approach is also future work.

Another point that might be interesting to focus on is the performance tests that were left to do. For this work, I only had time to test and compare results obtained from the smartphone application. In the future, it might be interesting to test the LWM2M server and the Raspberry Pi in other points of performance such as memory and CPU usage.

In conclusion, the work left to do described above represent good opportunities for new research work.

References

[1] Rose, Karen and Eldridge, Scott and Chapin, Lyman. The internet of things: An overview. In *The Internet Society (ISOC)*, pages 1–50. 2015.

[2] “Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)”. Available Online: `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`. Accessed in June 26th, 2018.

[3] Li, Shancang and Xu, Li Da and Zhao, Shanshan. The internet of things: a survey. In *Information Systems Frontiers*, vol. 27, pages 243–259. 2015.

[4] D. S. Sousa Nunes, P. Zhang, and J. Sa Silva. A Survey on human-in-The- loop applications towards an internet of all. In *IEEE Communications Surveys and Tutorials*. 2015.

[5] “Portugal tem a quarta taxa de abandono escolar mais elevada da UE”. Available Online: `https://www.dn.pt/sociedade/interior/portugal-tem-a-quarta-taxa-de-abandono-escolar-mais-elevada-da-ue---eurostat-6244999.html`. Accessed in June 26th, 2018.

[6] “Seven Reasons Freshmen Drop Out Of College”. Available online: `https://www.allianztuitioninsurance.com/resources/college-adjustment/seven-reasons-freshmen-drop-out-of-college`. Accessed in June 26th, 2018.

[7] “Fundação para a Ciência e a Tecnologia”. Available online: `https://www.fct.pt`. Accessed in June, 2018.

[8] “Scrum Methodology”. Available online: `http://scrummethodology.com`. Accessed in June 27th, 2018.

[9] Slack, “Slack”. Available online: `https://slack.com`. Accessed in June 27th, 2018.

[10] Rajkumar, Ragunathan Raj and Lee, Insup and Sha, Lui and Stankovic, John. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th design automation conference*, pages 731–736. AMC, 2010.

[11] Statista, “Android - Statistics & Facts”. Available online: `https://www.statista.com/topics/876/android/`. Accessed in June 18th, 2018.

[12] “75 Amazing Android Statistics and Facts”. Available online: `https://expandedramblings.com/index.php/android-statistics/`. Accessed in June 18th, 2018.

[13] “Number of available applications in the Google Play Store from December 2009 to March 2018”. Available online: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. Accessed in June 19th, 2018.

[14] “The Future Internet platform FIWARE”. Available online: https://ec.europa.eu/digital-single-market/en/future-internet-public-private-partnership. Accessed in June 24th, 2018.

[15] FIWARE, “What is FIWARE?”. Available online: https://www.fiware.org/about-us/. Accessed on June 24th, 2018.

[16] Telefónica, “FIWARE, the standard that the IoT needs”. Available online: https://iot.telefonica.com/blog/2016/09/en-fiware-standard-iot. Accessed in June 24th, 2018.

[17] FIWARE, “FIWARE GENERIC ENABLERS”. Available online: https://catalogue-server.fiware.org. Accessed in June 24th, 2018.

[18] Raspberry Pi Foundation, “Raspberry Pi”. Available online: https://www.raspberrypi.org. Accessed in June 21st, 2018.

[19] Arduino, “Arduino”. Available online: https://www.arduino.cc. Accessed in June 21st, 2018.

[20] “Raspberry Pi or Arduino Uno? One Simple Rule to Choose the Right Board”. Available online: https://makezine.com/2015/12/04/admittedly-simplistic-guide-raspberry-pi-vs-arduino/. Accessed in June 21st, 2018.

[21] Sinche Soraya and Jorge Sá Silva and Raposo, D. and Rodrigues, A. and Vasco Pereira and Boavida, F. , ”Towards Effective IoT Management”, in IEEE Sensors 2018 international conference, New Delhi, India, 28-31 October 2018., 2018.

[22] Hedstrom, Brian and Watwe, Akshay and Sakthidharan, Siddharth. Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions. In *University of Colorado, Master Thesis*, 2011.

[23] R.Enns, “NETCONF Configuration Protocol RFC 4741”, RFC Editor, 2006.

[24] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF) RFC 6241”, RFC Editor, 2011.

[25] A.Bierman, M.Bjorklund, and K.Watsen, “RESTCONF Protocol - RFC 8040,” no. 8040. RFC Editor, 2017.

[26] Open Mobile Alliance, “OMA SpecWorks”. Available online: https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight -m2m-lwm2m/. Accessed in May, 2018.

[27] “NETCONF versus RESTCONF: Capabilitity Comparisons for Data Model-driven Management”. Available online: https://www.claise.be/2017/10/netconf-versus-restconf-capabilitity-comparisons-for-data-model-driven-management-2/. Accessed in June, 2018.

[28] A. Sehgal, V. Perelman, S. Kuryla, J. Schonwalder, and O. In, “Management of resource constrained devices in the internet of things,” Commun. Mag. IEEE, vol. 50, no. 12, pp. 144–149, 2012.

[29] Bormann, C., K. Hartke, and Z. Shelby. ”The Constrained Application Protocol (CoAP).” RFC 7252 (2015).

[30] S. Sinche. “IoT Management System.ppt”, FCTUC/CISUC/LCT, October 2017.

[31] Rui Wang, Fanglin Chen, Zhenyu Chen, Tianxing Li, Gabriella Harari, Stefanie Tignor, Xia Zhou, Dror Ben-Zeev and Andrew T. Campbell. StudentLife: assessing mental health, academic performance and behavioral trends of college students using smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 3–14. AMC, 2014.

[32] Savvides, Andreas and Park, Heemin and Srivastava, Mani B. The bits and flops of the n-hop multilateration primitive for node localization problems. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 112–121. ACM, 2002.

[33] FIWARE, “CORE CONTEXT MANAGEMENT CHAPTER”. Available online: `https://www.fiware.org/developers/catalogue/`. Accessed in June 28th, 2018.

[34] Telefónica I+D, “FIWARE Short Time Historic (STH) - Comet”. Available online: `https://github.com/telefonicaid/fiware-sth-comet`. Accessed in June 28th, 2018.

[35] FIWARE, “INTERFACE WITH IOT, ROBOTS AND THIRD-PARTY SYSTEMS CHAPTER”. Available online: `https://www.fiware.org/developers/catalogue/`. Accessed in June 28th, 2018.

[36] FIWARE, “Backend Device Management - IDAS”. Available online: `https://catalogue-server.fiware.org/enablers/backend-device-management-idas`. Accessed in June 28th, 2018.

[37] Telefónica I+D, “Cygnus”. Available online: `https://github.com/telefonicaid/fiware-cygnus`. Accessed in June 28th, 2018.

[38] ckan, “CKAN: The Open Source Data Portal Software”. Available online: `https://github.com/ckan/ckan`. Accessed in June 28th, 2018.

[39] J. M. S. L. Fernandes, “ISABELA - IoT Student Advisor and BEst Lifestyle Analyzer”, *Master Thesis in Biomedical Engineering, Coimbra, 2017.*

[40] Android Developer, “Android Bluetooth APIs”. Available online: `https://developer.android.com/guide/topics/connectivity/bluetooth.html`. Accessed in April, 2018.

[41] Android Developer, “BluetoothAdapter”. Available online: `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html`. Accessed in April, 2018.

[42] Android Developer, “BroadcastReceiver”. Available online: `https://developer.android.com/reference/android/content/BroadcastReceiver.html`. Accessed in April, 2018.

[43] Android Developer, “BluetoothAdapter.getDefaultAdapter()”. Available online: `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getDefaultAdapter()`. Accessed in April, 2018.

[44] Android Developer, "Android Bluetooth API - Discovering Devices". Available online: https://developer.android.com/guide/topics/connectivity/bluetooth.html #DiscoveringDevices. Accessed in April, 2018.

[45] Android Developer, "IntentFilter". Available online: https://developer.android.com/reference/android/content/IntentFilter.html. Accessed in April, 2018.

[46] Android Developer, "Intent.getAction()". Available online: https://developer.android.com/reference/android/content/Intent.html #getAction(). Accessed in April, 2018.

[47] Android Developer, "BluetoothDevice.ACTION_FOUND". Available online: https://developer.android.com/reference/android/bluetooth/Bluetooth Device.html#ACTION_FOUND. Accessed in April, 2018.

[48] Android Developer, "BluetoothAdapter.ACTION_DISCOVERY_FINISHED". Available online: https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#ACTION_DISCOVERY_FINISHED. Accessed in April, 2018.

[49] Android Developer, "registerReceiver(BroadcastReceiver, IntentFilter)". Available online: https://developer.android.com/reference/android/content/Context .html#registerReceiver(android.content.BroadcastReceiver,android.content.IntentFilter). Accessed in April, 2018.

[50] FIWARE, "Fiware". Available online: https://www.fiware.org. Accessed in April, 2018.

[51] Android Developer, "LocalBroadcastManager.sendBroadcast(Intent)". Available online: https://developer.android.com/reference/android/support/v4/content /LocalBroadcastManager.html#sendBroadcast(android.content.Intent). Accessed in April, 2018.

[52] Android Developer, "Collections.sort(List, Comparator)". Available online: https://developer.android.com/reference/java/util/Collections.html #sort(java.util.List<T>,java.util.Comparator<? super T>). Accessed in April, 2018.

[53] Android Developer, "RecyclerView.Adapter.onBindViewHolder(ViewHolder, int)". Available in: https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#bindViewHolder(VH,int). Accessed in April, 2018.

[54] Rescorla, Eric and Modadugu, Nagendra, "Datagram transport layer security version 1.2", 2012.

[55] Klas, Guenter and Rodermund, Friedhelm and Shelby, Zach and Akhouri, Sandeep and Höller, J. Lightweight M2M: enabling device management and applications for the internet of things. In *White Paper from Vodafone, Ericsson and ARM*, vol. 26. 2014.

[56] Alliance, Open Mobile. Lightweight Machine to Machine Requirements, pages 1–12, 2017.

[57] Eclipse, "Eclipse Leshan". Available online: https://www.eclipse.org/leshan/. Accessed in June, 2018.

[58] Open Mobile Alliance, “LwM2M Registry”. Available Online: http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html. Accessed in January 6th, 2019.

[59] Eclipse, Leshan Project. Available Online: https://github.com/eclipse/leshan. Accessed in May, 2018.

[60] ApplicationPlatformForIoT, “LwM2MDemoClientAndroid”. Available Online: https://github.com/ApplicationPlatformForIoT/LwM2MDemoClientAndroid. Accessed in October, 2018.

[61] FIWARE, “WELCOME TO THE FIWARE IOT AGENT FOR OMA LIGHTWEIGHT M2M”. Available Online: https://fiware-iotagent-lwm2m.readthedocs.io/en/latest/index.html. Accessed in January 10th, 2019.

[62] Telefónica I+D, “Features”. Available Online: https://github.com/telefonicaid/iotagent-node-lib#features. Accessed in January 10th, 2019.

[63] Telefónica I+D, “lightweightm2m-iotagent”. Available Online: https://github.com/telefonicaid/lightweightm2m-iotagent. Accessed in November, 2018.

[64] Telefónica I+D, “lwm2m-node-lib”. Available Online: https://github.com/telefonicaid/lwm2m-node-lib. Accessed in November, 2018.

[65] Telefonica I+D, “Device to NGSI Mapping”. Available Online: https://github.com/telefonicaid/iotagent-node-lib#device-to-ngsi-mapping. Accessed in January 10th, 2019.

[66] “apt-get update”. Available online: https://askubuntu.com/questions/222348/what-does-sudo-apt-get-update-do. Accessed in April, 2018.

[67] Eclipse, “Eclipse Kepler”. Available online: http://www.eclipse.org/downloads/packages/eclipse-standard-432/keplersr2. Accessed in April, 2018.

[68] adafruit, “Adafruit Python DHT Sensor Library”. Available online: https://github.com/adafruit/Adafruit_Python_DHT. Acessed in June, 2018.

[69] Google, “Battery Historian”. Available Online: https://github.com/google/battery-historian. Accessed in January 3rd, 2019.

[70] Docker Inc., “Docker”. Available Online: https://www.docker.com. Accessed in January 3rd, 2019.

[71] Android Developer, “Profile battery usage with Batterystats and Battery Historian”. Available Online: https://developer.android.com/studio/profile/battery-historian. Accessed in January 3rd, 2019.

[72] N. Armando. “A Unified Management Approach for the Extended IoT.ppt”, FCTUC/CISUC/LCT, December 2018.

[73] Telefónica I+D, “lightweightm2m-iotagent Dockerfile”. Available Online: https://hub.docker.com/r/telefonicaiot/lightweightm2m-iotagent/dockerfile. Accessed in December, 2019.

This page is intentionally left blank.

Appendices

Appendix A

Configuration of the Virtual Machine for the Leshan LWM2M Server

First, we begin with a download of the package lists from the repositories and “update” them to get information on the newest versions of packages and their dependencies [66]. So, in terminal, we run this command:

```
sudo apt-get update
```

The next step is to install the Oracle JDK, so we have to run the following commands:

```
sudo apt-get install software-properties
sudo apt-add-repository ppa:webupd8/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

Then, it’s time to get Eclipse. We download the Kepler version, from the website [67] and we extract it on the *Download* folder. In terminal, we move the extracted folder to a new location through the following commands:

```
sudo mkdir -p /opt/ide/64
sudo mv .~/Downloads/eclipse /opt/ide/64
```

After that, we have to give administrator permissions to the new folder. Inside the new folder, we run these commands:

```
sudo chown -R root:root eclipse
```

At this point, we need Maven. So we download it, through terminal:

```
sudo apt-get update
sudo apt-get install maven
```

Now, we need to define and configure the workspace:

```
sudo mvn -e -Declipse.workspace=<workspace folder> eclipse:configure-workspace
```

(Our workspace folder is `/home/leshan/workspace`.)

Finally, we will clone the Leshan project from GitHub inside the *workspace* folder:

```
sudo git clone https://github.com/eclipse/leshan.git
```

Then, inside the Leshan folder that we just cloned, we have to run tests, with the help of the following commands:

```
mvn eclipse:eclipse
mvn install
```

Lastly, we import the project into Eclipse and, if there are no errors, the configuration is finished.

Appendix B

Configuration of the FIWARE Environment for the LWM2M IoT Agent

First, we needed to have a virtualization software installed and I used VMWare Player, then, we needed to download a FIWARE .ovf Image with all its modules installed and, finally, we needed an SSH client. The FIWARE modules were deployed using Docker containers.

After importing the .ovf image to the VMWare Player, we started configuring the modules so they fitted our system. We did this through the *docker-compose* file, in which we defined the services we used. For our system:

- MongoDB: a database to store the information.
- ORION module: an image of the FIWARE ORION module.
- IDAS module: an image of the LightweightM2M Agent provided by the Telefónica I+D [73].

In this file we, also, defined the host names, the container names and the ports of each of the services.

Finally, we started all FIWARE containers by running the following command:

```
docker-compose up -d
```

Note: To check the running modules we run `docker ps` and to stop all modules we run `docker-compose stop`.

Appendix C

Article

Towards the Development of IoT Management in Human-in-the-Loop Cyber-physical Systems

I. Mota[1], S. Sinche[1 2], D. Raposo[1], J. Fernandes[1], A. Ngombo[1], J. Sá Silva[1], A. Rodrigues[1 3] and F. Boavida[1]

Abstract— **This paper presents an innovative system to the monitorization of the surroundings of students in favor of improving their academic performances by using a Human-in-the-Loop Cyber-physical system. Therefore, the system is composed by a lot of different sensors that are incorporated in different physical devices. With this in mind, we explore the concept of management of a network of these heterogeneous devices and evaluate the path to follow taking into account the components that already existed in the system. Lightweight Machine-to-Machine was the chosen protocol, even though it is recent and there was not a lot of information of its use in cyber-physical systems. To integrate the protocol in the existent system we used Eclipse's Leshan project and FIWARE's LWM2M IoT Agent. Concerns about the consumption of energy by the smartphone application with the protocol were raised, so performance tests were made and the results were favorable.**

Keywords: Internet of Things, Cyber-physical Systems, Human-in-the-Loop, Lightweight machine-to-machine, CoAP, IoT Agent.

I. Introduction

IoT Student Advisor and Best Lifestyle Analyzer (ISABELA) is a Human-in-the-Loop Cyber-physical system that collects data from students and their environment in order to provide an accurate evaluation of their state. For that, the system is composed by a plethora of sensors distributed by several devices. Systems like ISABELA, that depend on their components work to achieve their objectives, can benefit from the use of protocols with management capabilities.

Management gives control over the system functionalities, as it can monitor network performance, detect faults and configure parameters. Therefore, the integration of a network management protocol that manages the heterogeneous devices that are part of the ISABELA system was the main objective of this work. A study of various protocols was made and the chosen one was LWM2M, a protocol design by Open Mobile Alliance SpecWorks for managing sensor networks and remote machine-to-machine devices. Its integration was made through two approaches: 1) Eclipse's *Leshan* project, where the management protocol is parallel to the ISABELA system and 2) FIWARE's *LWM2M IoT Agent*, where the server is integrated on the project's IoT middleware (FIWARE).

[1]Department of Informatic Engineering, University of Coimbra, Coimbra, Portugal

[2]Department of Eletronic, Telecommunications and Networks, Escuela Politcnica Nacional, Quito, Ecuador

[3]Polytechnic Institute of Coimbra, Coimbra, Portugal

imota@student.dei.uc.pt, {smaita, draposo, jmfernandes, narmando, sasilva, arod, boavida}@dei.uc.pt

The paper is structured as follows: in section II a presentation and evaluation of the various network management protocols considered to manage the several devices that form part of the system is given. In section III, there is a brief description of the ISABELA project and of some of its components. Section IV presents the process of development of the components that make it possible to fulfill the objective of this work. Section V is focused on the tests made to the application and consequent results. Finally, section VI is dedicated to the conclusions taken from the development of the work.

II. IoT Management

The number of active devices is rising and their complexity is increasing. This resulted in the creation of various tools and technologies specialized in IoT management.

Management functionalities require data exchanges between the manager and the managed systems. The managed systems are often different from each other in terms of computing capabilities, storage and energy consumption.

The heterogeneity of the devices that make up the system must be considered when a way to manage them is being chosen. Several organizations developed standards able to handle heterogeneous devices and some of those were considered for this work and are going to be presented in the next subsections.

A. Protocols

1) Network Configuration Protocol (NETCONF): NETCONF is a network management protocol proposed by the Internet Engineering Task Force (IETF). It was created so that it was possible to configure networks of devices, implementing functions to install, edit and delete its parameters [1]. It uses XML encoding for the configuration data as well as the protocol messages. The transport protocol is TCP, generally. Its operations are performed as remote procedure calls. NETCONF establishes SSH sessions between its server and its client [2] and defines the configuration of data stores and a set of operations - Create, Read, Update and Delete (CRUD) - that can be used to access the aforementioned data stores [3][4].

2) RESTCONF: RESTCONF is a protocol based on HTML that provides an interface for accessing data defined in YANG[1], using datastores defined in NETCONF. Its operations are the HTTP operations *GET*, *POST*, *PUT*, *PATCH*

[1]Data modeling language for the definition of data sent over the NETCONF protocol.

and *DELETE* and the transport protocol is HTTP. It uses HTTP methods as equivalents for NETCONF operations. This protocol was also designed by the IETF [5] and does not intend to replace NETCONF. In fact, they can coexist (Fig. 1), so that RESTCONF can offer an additional interface with REST-like functionalities to the NETCONF protocol, or it can be used alone (Fig. 2) [1].

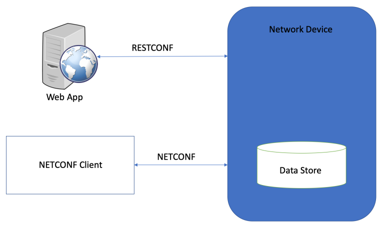


Fig. 1: RESTCONF with the NETCONF Server [1].

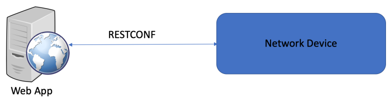


Fig. 2: RESTCONF without the NETCONF Server [1].

3) Lightweight Machine-to-machine (LWM2M): LWM2M is a protocol for IoT device management from Open Mobile Alliance SpecWorks [6]. It was designed for sensor networks and remote management of machine-to-machine devices. Its architectural design is based on REST, defines an extensible resource and data model and builds on an efficient secure data transfer standard called CoAP[2]. The target devices are mainly resource constrained, so the protocol is light and compact.

LWM2M is a protocol which implements a client-server architecture and uses CoAP as the underlying transfer protocol over UDP and SMS bearers (optional). It also includes a secure channel through which the messages between the server and the client are interchanged. This level of security is provided by the DTLS[3] protocol [11].

The LWM2M Enabler[4] has two components: a server and a client. The server is typically located in a data center and can be hosted by the M2M, Network or Application Service Provider. The client resides in the device. It also has four interfaces between its two components: 1) Bootstrap; 2) Client Registration; 3) Device management and Service Enablement; 4) Information Reporting. The architecture of the LWM2M Enabler is represented in Fig. 3.

[2]Constrained Application Protocol is a specialized Internet Application Protocol for constrained devices, defined in RFC 7252 [9].

[3]Data Transport Layer Security (DTLS) is a protocol that secures communications between a client and sever, defined in RFC 6347 [10].

[4]Name given to the standard produced by OMA for the LWM2M solution.

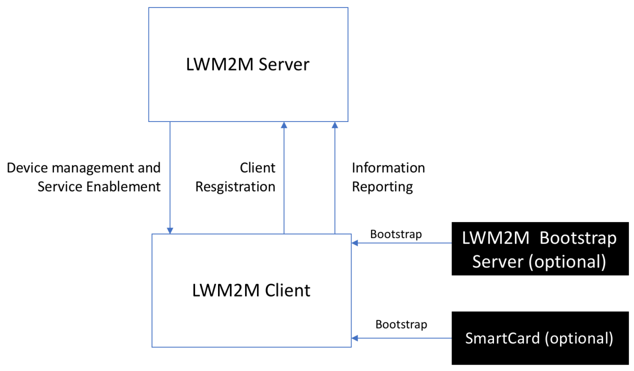


Fig. 3: Architecture of the LWM2M Enabler [12].

The LWM2M model demmands the use of end-to-end IP connectivity between the client and the server. However, a gateway can be used when non-IP endpoints are needed, although LWM2M poses great challenges in respect to remote gateway management [1].

B. Comparative Analysis

NETCONF was the first real solution for the integration of a network management protocol as one of its strengths is its support for robust configuration change involving multiple devices. These transactions are mostly important when configuring services across network elements [7]. But for networks of constrained devices the operations tested were only *get*, *get-config*, *copy-config*, *lock* and *unlock*. NETCONF does not necessarily require a security mechanism but some authors tested a light version of the protocol with some mechanisms in constrained devices and concluded that it is inefficient in terms of memory usage [8].

RESTCONF is a more recent bet from the IETF and, as said before, does not intend to replace NETCONF. Unlike NETCONF, RESTCONF allows for XML and JSON to be used and does not have the concept of distributed transactions, only device-by-device configuration. Also, a call from this protocol is a transaction by itself as it uses HTTP operations to edit data resources [7] and it requires a security mechanism - Transport Layer Security (TLS).

LWM2M is a client-server model and requires end-to-end IP connectivity. Sometimes this poses as a challenge but some enhancing strategies are being proposed in recent works. Despite this, LWM2M is widely implemented and used, thus becoming the solution for the management of constrained device with the highest acceptance of the scientific community [1]. Therefore, it was selected to be incorporated into the ISABELA project, to help manage all the physical sensors and, hopefully later, also manage the virtual sensors.

The characteristics of each protocol analyzed are summarized in table I.

Protocol	NETCONF	RESTCONF	LWM2M
Standard	IETF	IETF	OMA
Definition Language	YANG	YANG	LWM2M Language, YANG Extended
Information Model	YANG Modules	YANG Modules	LWM2M Objects
Instantiated Information/Transfer Syntax (payload)	XML	XML or JSON	Plain Text, JSON or TLV
Transfer Protocol	SSH, SSL, HTTP TLS	HTTP, TLS, HTTPS with X.509v3	CoAP, DTLS, UDP
Security	Yes	Yes	Yes
Used on Constrained Devices	Yes, reducing some operations	Yes, reducing some operations	Yes

TABLE I: Summarized Protocols [13].

III. ISABELA

ISABELA is an Human-in-the-Loop Cyber-physical system, whose goal is to prevent bad academic performances. It was developed to collect data from students and evaluate it in order to assess the state of the participants and help them, by providing feedback and advices to try to prevent poor academic results.

A. General Overview

ISABELA uses smartphones' sensors, other physical sensors and some virtual ones, too, to collect information of the participants' day-to-day life, in order to detect behavior that is potentially bad for good academic results. This application is meant to be used by college students.

The application monitors the students continuously. The sensors retrieve information that is sent to FIWARE through an Internet connection. Then, the data is retrieved from FIWARE, it is processed and shown to the user. When behavior that is not deemed helpful to achieve academic success is detected, a ChatBot sends a notification and a message and if the user wants to know more about how to prevent that type of behavior, they can ask the ChatBot and it gives back a well-founded answer.

To assess the students' behavior, ISABELA collects information about their activity, sleep, sociability, social media presence and location.

B. System Architecture and Components

Human-in-the-Loop Cyber-physical systems work in four phases: data acquisition, inference, future inference and actuation. With that in mind, the architecture, represented in Fig. 4, emerged.

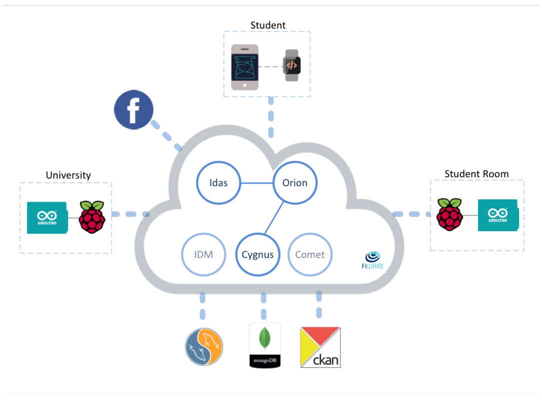


Fig. 4: System architecture.

We use the smartphone and all the sensors to acquire the data. FIWARE provides the storage capabilities and allows the communication between the devices. State inference is made in the smartphone and the actuation is performed through messages and notifications, based on the previous inferences.

FIWARE has the objective of standardize the way we collect, manage and publish context information and to solve the heterogeneity in IoT protocols. Therefore, it is used as the backend of the ISABELA system. In Fig. 5 its architecture is represented.

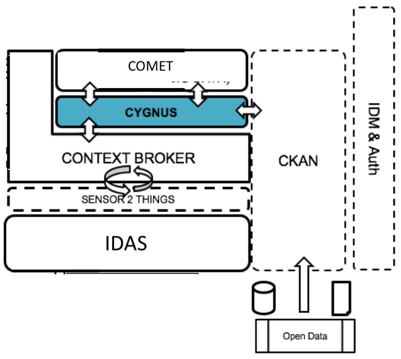


Fig. 5: FIWARE architecture.

The ISABELA system uses many FIWARE modules. These modules are going to be explained in the following sections.

1) ORION: the context broker. This broker is capable of representing several IoT contexts using a new representation standard – FIWARE NGSIv2 API. This API implements a REST API, therefore, is capable of performing updates, queries or react to changes. This is a must have capability, as we want to create connections between the sensors and the applications that consume the information.

ORION only holds information about the last instance of an entity and/or an attribute. To save the context history, created by the evolution of context information overtime, we need to accompany ORION with COMET and CYGNUS [14].

2) COMET: or FIWARE Short Time Historic is a component capable of storing and retrieving historical context information [15]. As said before, it communicates with ORION and lets external clients, like ISABELA, query the stored information, through REST API.

COMET is very useful because it allows us to query specific time intervals and aggregate information by time, sums and occurrences.

COMET is a very important component of this system, as it is from it that it gets most of the data in the application.

3) IDAS: is the module that handles the communications with the devices that make up the system, as it offers a wide range of IoT agents [16].

This module is needed to connect objects to gather data. While the smartphone connects directly with ORION, the embedded devices connect to IDAS. Its IoT agents translate IoT-specific protocols into FIWARE standard data exchange model. By using an IoT agent, the devices can subscribe entities and can query and be updated if a value of that entity is changed [17].

4) Other Modules:

- **CYGNUS:** is a connector in charge of coordinating the data, creating a historical view of such data. We can introduce this data into third-party storage systems, such as MongoDB or MySQL. It also connects the ORION to many FIWARE storages, like CKAN and COMET [18].

- **CKAN:** is not a FIWARE module. It is an open data platform that makes it easy to publish, share and work with data. This platform allows us to store the retrieved data, with the advantage that it has a rich front-end and visualization tools, that we can use to see the data [19]. CKAN provides a powerful way for cataloging and accessing datasets. If the data retrieved during this project is available on this platform, it can be used by other investigators.

IV. Implementation

We tried two approaches to apply LWM2M to our system: through the Eclipse's Leshan project and through an IoT Agent. They are going to be explained in the next sections.

A. Leshan Solution

This solution was implemented with the help of an Eclipse project called Leshan. Leshan provides libraries that help with what we wanted to do [20]. The project also provides a server and a client demonstration as an example of the Leshan API and for testing purposes.

The server of the LWM2M protocol was lodged in a virtual machine, running Ubuntu 16.04. The clients were a Raspberry Pi with a digital temperature and humidity sensor and a smartphone running Android. The server and the Raspberry Pi were connected through a switch via LAN connection and the digital sensor to the GPIO pins of the Raspberry Pi. The server and the smartphone were connected via the Wi-Fi network. The architecture of the Leshan solution is represented in Fig. 6.

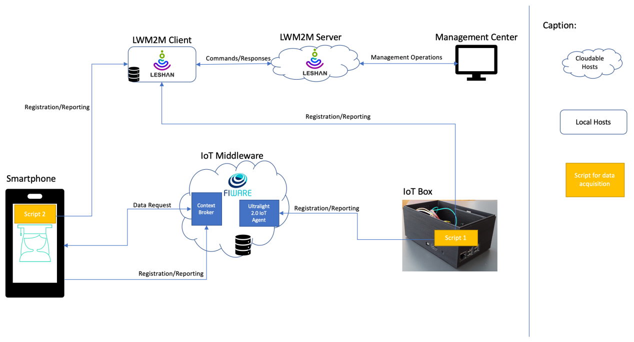


Fig. 6: Architecture of the Leshan solution.

1) Server: To launch the server, we just need to run, in Eclipse, the *leshan-server-demo* project as a Java Application and then select *LeshanServerDemo.class*.

Then, we need to open Mozilla Firefox and go to `https://localhost:8080` to open the server page.

For the sensor's parameters and its values to appear on the device's page the server has to have files, written in .xml format, in which they are defined. For the more generic and used sensors, such as the accelerometer, these files are available in a LWM2M registry provided by OMA [21] and for the other sensors that are not so common, like the physical activity, they had to be created.

2) Clients: The client also needs to declare the same .xml models that the server has so that the parameters and the values sensed appear on the device's page.

- **Raspberry Pi:**
  In order to adapt the demos to our system, some changes were made to the code provided by the Leshan project that we downloaded from its GitHub repository [22].
  First of all, the original client only had a (random) temperature sensor, so we had to prepare our client to read values of temperature and humidity and create classes for these sensors (*TemperatureSensor* and *HumiditySensor*). Then, we needed to create a class to deal with the DHT11 sensor[5] - *DHT11Class*. All sensor classes receive an object of type *DHT11Class* as parameter.
  In ***DHT11Class***, the values from the sensors are read through the Python script *sensorsleshan.py*. The results returned from this script are put into a buffer and parsed into respective variables.

[5]Digital sensor used for temperature and humidity readings.

In each sensor's class, there are variables for maximum, minimum and current values, of the parameter that is being read. We will need them to appear in the device's information page when that device connects to the server.
There are many operations that can be done through these classes. We can update the values read and we can reset and adjust the maximum and minimum measured values.
In order to update the values read, we need to declare a *DHT11Class* object. Through this object we can obtain a value, for example by invoking the function *getTemperature*.
In order to adjust the minimum and maximum measured values, a comparison between the values previously read and the current one is performed.
In order to reset the minimum and maximum measured values, these are made equal to the current value.

After the update of the code to match our system, we needed to run the client, so we used a .jar file of the *leshan-client-demo* project, with all the changes made. It was exported, in Eclipse, as a *JAVA App* and, then, a *runnable JAR file*. The .jar file was then put into the Raspberry Pi.
We needed a tool to help get the temperature and humidity values from the DHT11 sensor. We chose to install the *Adafruit Python DHT Sensor Library*, which is a Python library used to read the DHT series of humidity and temperature sensors on a Raspberry Pi [23].
First, we downloaded the library to the Raspberry Pi:

```
git clone https://github.com/adafruit/
Adafruit_Python_DHT.git
```

Then, we made sure that our system was ready to compile Python extensions. So, we run the following commands:

```
sudo apt-get upgrade
sudo apt-get install build-essential
python-dev
```

Eventually, we installed the library by running the following command, inside its folder:

```
sudo python setup.py install
```

Finally, we connected the server and the client. First we launched the server, as said in section IV-A.1, and then we opened a terminal tab inside the Raspberry Pi and run the command:

```
java -jar leshan-client-demo.jar -u
<server's ip address>:5683
```

(5683 is the Leshan port.)

- **Android:**
  For this part we found a demo application in a GitHub repository [24] that implemented a LWM2M Leshan client in Android. The demo included some Sensor Classes, where the values were read and sent to the server, and the Main Activity, where the LWM2M client was built with its objects and it connected to the server.
  To integrate the client in the ISABELA application, we started with the creation of a class called *LWM2MClient* that does the same that the Main Activity did in the demo application. The first change in this class was the server to which the clients were going to connect to. The demo application came with a connection to the Bootstrap Server[6] and the client needed to connect to the server we adapted for ISABELA and had running in the local machine. For this to happen, we needed to give the IP address and ID of the our server and a constructor of a Server with the same ID to the initializer of the LWM2M client, instead of passing the IP and constructor of the Bootstrap Server.
  After that, we needed to create the classes for the sensors and the services that permitted reading the values.
  Fundamentally, every Sensor class has a constructor and is divided in a *start* function, where the parameters that start the sensor are initialized, a *stop* function, where the same parameters are stopped, an *id*, where the id of that sensor is defined and has to correspond to the one defined in the .xml model, a function where the values are read and a *read* function that returns the values that are read and essentially updates the values in the device's page.
  Each one of these sensors are started in the *LWM2MClient* and added to the initializer of the LWM2M client along with its .xml models. That is why their parameters appear on the device's page on the server side.
  An instance of the *LWM2MClient* is created in the *Main_Service* so that it creates the client, connects to the server and starts sensing as soon as the application starts.

  Disclaimer: Before applying the changes directly in ISABELA, I adapted the demo application to connect to the server we had and created the classes of the sensors we needed and tested from that.

When the connection is established, the device appears on the server page. By clicking on the device, we have access to its information.

B. IoT Agent Solution

This approach consisted on integrating an LWM2M IoT agent in FIWARE to serve as a bridge of communication between the devices that use the LWM2M protocol and the Context Broker (ORION module) [25].

[6]Implementation offered by the project.

The *Lightweight M2M IoT Agent* is a standard FIWARE IoT Agent based in the public Node.js IoT Agent Library. The Agent described in that library is a component that facilitates the management and control of the information of a group of devices from a FIWARE NGSI Context Broker using their own native protocols.

These agents support various types of features [26]. They are:

- **Device Registration:** all the devices connected to the agent are mapped to a Context Broker entity.
- **Device Information Update:** when there are new values for an attribute, the devices should send the information to the agent and the agent should send a request to the Context Broker to update the devices entity.
- **Device Management:** the agent should offer a repository where the devices are registered and that holds information needed for the connection to the Context Broker.
- **Device Provisioning:** the agent should offer an external way for the user to be able to customize the device's entity name, type and information.
- **Type Configuration:** the agent should provide default values to the device attributes based on its type.

Devices measures can have three different behaviors:

- **Commands:** an attribute is set on the Context Broker entity and the agent will be responsible of contacting the device to perform the command itself, updating as soon as it has the information.
- **Lazy Attributes:** the sensors will wait for the agent to request data.
- **Active Attributes:** the sensed values are pushed from the device to the agent, that will request an update to the Context Broker with that information.

An implementation of this agent is given by *Telefońica Investigacin y Desarrollo*[7] and is available in a GitHub repository [27]. *Telefońica* also offers a library that aims to be a simple way to build LWM2M servers and clients with Node.js [28] and we adapted it to fit the devices of our system.

Here, the server and the smartphone were, also, connected via the Wi-Fi network. The architecture of the IoT Agent solution is represented in Fig. 7.

[7]Spanish telecommunications company.

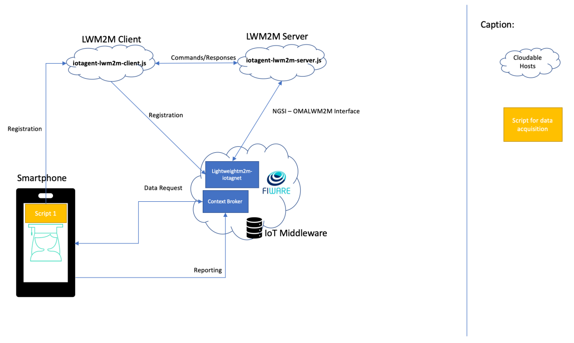


Fig. 7: Architecture of the IoT Agent solution.

1) LWM2M IoT Agent: As previously said, the IDAS module is the one that handles communications with the devices through its IoT agents. So, its configuration was made to implement the *LightweightM2M IoT Agent*.

This agent uses the features provided by the Node.js IoT Agent Library with some adaptations to the LWM2M protocol, as is the case of the Mappings. For LWM2M mapping can be:

1) OMA Registry objects and resources from their URIs to their common names.
2) Custom device objects to the names defined by the user.

To accomplish that, the agent supports:

- *lwm2mResourceMapping*: an additional property that lets the user customize the names for particular resources.
- *omaRegistry.json*: contains the OMA Registry previously mentioned and is used for automatic mappings in case there are not custom ones.

No alterations were made to the agent files for our work. The server and the client are going to be explained in the next sections.

2) Server: The implementation of this server was based on the one provided by *Telefońica* [28] and the only changes we made were to add the customized models of the sensors that are not part of the OMA Registry.

The agent calls up the server in its implementation so it starts running when the FIWARE containers are started.

3) Client: As was the case for the Leshan client, this client had to be adapted to run on Android [28], as it was written in a different language.

For this, I made use of one of the agent's features - the provisioning - and created a script were I defined an Entity, its type and attributes. Then, in the same file, I mapped the LWM2M resources according to the .xml models of the sensors.

In the class *LWM2MClient*, the endpoint name was made to correspond to the one I had given to the Entity in the provisioning and the connection was made to the IP and port of the server, running in the Docker.

Disclaimer: This approach was not fully implemented. Although the server received connections from the client, it was not being registered in ORION and the readings did not appear. The challenges that appeared during this part of the work are explained in section VI.

V. Validation and Tests

With the incorporation of a new protocol some questions about the performance of the system were raised, therefore some tests in respect to the performance of the system were performed and are presented in this section.

For the tests we used a tool, developed by Google, named *Battery Historian*. This tool creates reports from the usage of a smartphone while it is not plugged in and allows the visualization of system events and statistics. It also allows the selection of a specific application and subsequent inspection of the metrics that affect the battery life of said application [29].

A. Tests

The tests were focused on the impact of the LWM2M protocol on the battery performance of the Android ISABELA application. For these tests, we let the ISABELA application run for two days, using it normally to fill the sleep forms and view some information. Two situations were tested:

1) Only running the ISABELA application.
2) Running the ISABELA application with the LWM2M client (Leshan solution).

Additionally, other tests were performed to evaluate the impact of the interval of time between readings on the battery use. For these, we used the demo application that implemented the LWM2M client in Android with the changes I made before applying them to ISABELA.

These tests were made for 10, 20, 30, 40, 50 and 60 seconds in time periods of 6 hours each.

B. Results

After running the tests and analyzing the data that resulted with Battery Historian, the percentage of battery use for each of the systems previously mentioned is shown in the following table:

TESTED SYSTEM	BATTERY USE
1. ISABELA (only)	4,83%
2. ISABELA + LWM2M	7,50%

TABLE II: Battery use for each system.

As expected, system 1 consumes less energy than system 2, as system 2 spends energy with the ISABELA parts and on top of it has the LWM2M client parts too. Because there is a duplication of information in the Leshan solution, it does not go through the same route, as it does in the IoT Agent solution. Therefore, we thought it was best to test the system with the worst case scenario in terms of performance, because if the results were favorable for that one, they would be favorable for the best one.

Finally, as we can see on the table, the percentage of battery use of system 2 is not significantly higher than the percentage of battery use of system 1. Furthermore, in terms of other important performance metrics, for example the CPU Usage, the values are the same or, at least, less different between them than the battery use ones.

As a result, we can conclude that by adding the LWM2M component to the ISABELA Android application we are not sacrificing its performance.

For the application only running the LWM2M client the results are shown in the following graph:

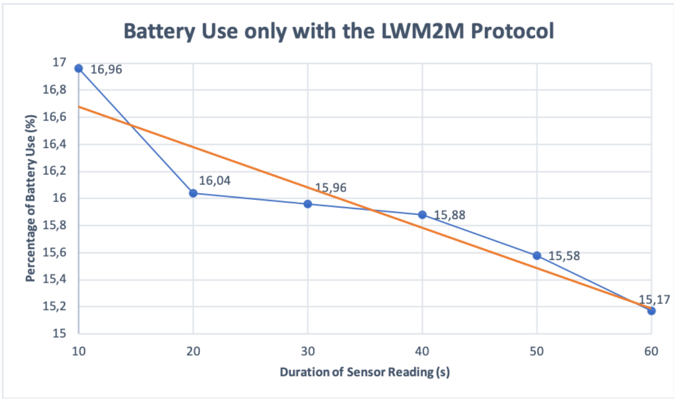


Fig. 8: Battery Use only with the LWM2M Protocol.

As it was expected, the value of the percentage of battery use decreases as the time interval increases. The system has to perform less actions in the same amount of total time. Furthermore, as we can see, the values of the percentage of battery use are very close, not having a difference bigger than, approximately, 2% between the outermost values.

The ISABELA application collects data in a time interval of 30 seconds. As the values do not vary much, we can state that this time interval is not significantly more battery consuming than the bigger intervals considered, so it can be deemed appropriate.

VI. Conclusions

We were able to adapt and incorporate a network management protocol with clients for Android and Raspberry Pi. However, we can conclude that there is a long way to go and that each approach that we took for the integration of the LWM2M protocol has its limitations.

As a matter of fact, the *Leshan* implementation has reading management capabilities for both Android and stationary devices [30], but as it is:

- There is a duplication of the information. The same information that is sent to the LWM2M server is sent through another route to the FIWARE's context broker.

- There is absence of execute-write-delete capabilities.
- The values in the LWM2M server are only stored for until there is a new reading that replaces the older one.
- There is no capability of automatic group device provisioning.

Even so this approach offers a simple way to see the each device that is connected, their attributes and sensed values through its interface.

The *IoT Agent* implementation was considered to be the solution to some of the limitations that the Leshan implementation offers. For example, it is a way of implementing a bridge between the LWM2M protocol and the system's components (FIWARE). So, it obliterates the duplication of information, as the information passes through the IoT agent to the server and the context broker directly. It also offers the capability of group device provisioning through running a script and the storage of information in a database.

On one hand, the existence of a tool, developed by the same corporation that developed the data center that we chose for our project, which gave answers to some of the limitations of other existent tools was of great help because the integration was a smoother process.

But, on the other hand, the fact that it implemented the server and the client in a complete different way and language posed as a very hard challenge in and of itself. As I could learn, the time that I had to work with the protocol was not sufficient to benefit from its capabilities to the fullest. Because the challenges that appeared were so time consuming, I could not finish the incorporation through this approach.

Even so, we could conclude that the incorporation of a protocol like this did not affect significantly the performance of this system. So, systems like this that are composed by heterogeneous devices and depend on them working to give accurate and real advice only have to gain if they can have a protocol that manages and controls them.

REFERENCES

[1] Sinche Soraya and Jorge S Silva and Raposo, D. and Rodrigues, A. and Vasco Pereira and Boavida, F. , "Towards Effective IoT Management", in IEEE Sensors 2018 international conference, New Delhi, India, 28-31 October 2018., 2018.

[2] Hedstrom, Brian and Watwe, Akshay and Sakthidharan, Siddharth. Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions. In *University of Colorado, Master Thesis*, 2011.

[3] R.Enns, "NETCONF Configuration Protocol RFC 4741", RFC Editor, 2006.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF) RFC 6241, RFC Editor, 2011.

[5] A.Bierman, M.Bjorklund, and K.Watsen, "RESTCONF Protocol - RFC 8040, no. 8040. RFC Editor, 2017.

[6] Open Mobile Alliance, "OMA SpecWorks". Available online: `https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/`. Accessed in May, 2018.

[7] "NETCONF versus RESTCONF: Capabilitity Comparisons for Data Model-driven Management". Available online: `https://www.claise.be/2017/10/netconf-versus-restconf-capabilitity-comparisons-for-data-model-driven-management-2/`. Accessed in June, 2018.

[8] A. Sehgal, V. Perelman, S. Kuryla, J. Schonwalder, and O. In, "Management of resource constrained devices in the internet of things, Commun. Mag. IEEE, vol. 50, no. 12, pp. 144149, 2012.

[9] Bormann, C., K. Hartke, and Z. Shelby. "The Constrained Application Protocol (CoAP)." RFC 7252 (2015).

[10] Rescorla, Eric and Modadugu, Nagendra, "Datagram transport layer security version 1.2", 2012.

[11] Klas, Guenter and Rodermund, Friedhelm and Shelby, Zach and Akhouri, Sandeep and Höller, J. Lightweight M2M: enabling device management and applications for the internet of things. In *White Paper from Vodafone, Ericsson and ARM*, vol. 26. 2014.

[12] Alliance, Open Mobile. Lightweight Machine to Machine Requirements, pages 1–12, 2017.

[13] S. Sinche. "IoT Management System.ppt", FCTUC/CISUC/LCT, October 2017.

[14] FIWARE, "CORE CONTEXT MANAGEMENT CHAPTER". Available online: `https://www.fiware.org/developers/catalogue/`. Accessed in June 28th, 2018.

[15] Telefnica I+D, "FIWARE Short Time Historic (STH) - Comet". Available online: `https://github.com/telefonicaid/fiware-sth-comet`. Accessed in June 28th, 2018.

[16] FIWARE, "INTERFACE WITH IOT, ROBOTS AND THIRD-PARTY SYSTEMS CHAPTER". Available online: `https://www.fiware.org/developers/catalogue/`. Accessed in June 28th, 2018.

[17] FIWARE, "Backend Device Management - IDAS". Available online: `https://catalogue-server.fiware.org/enablers/backend-device-management-idas`. Accessed in June 28th, 2018.

[18] Telefońica I+D, "Cygnus". Available online: `https://github.com/telefonicaid/fiware-cygnus`. Accessed in June 28th, 2018.

[19] ckan, "CKAN: The Open Source Data Portal Software". Available online: `https://github.com/ckan/ckan`. Accessed in June 28th, 2018.

[20] Eclipse, "Eclipse Leshan". Available online: `https://www.eclipse.org/leshan/`. Accessed in June, 2018.

[21] Open Mobile Alliance, "LwM2M Registry". Available Online: `http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html`. Accessed in January 6th, 2019.

[22] Eclipse, Leshan Project. Available Online: `https://github.com/eclipse/leshan`. Accessed in May, 2018.

[23] adafruit, "Adafruit Python DHT Sensor Library". Available online: `https://github.com/adafruit/Adafruit_Python_DHT`. Accessed in June, 2018.

[24] ApplicationPlatformForIoT, "LwM2MDemoClientAndroid". Available Online: `https://github.com/ApplicationPlatformForIoT/LwM2MDemoClientAndroid`. Accessed in October, 2018.

[25] FIWARE, "WELCOME TO THE FIWARE IOT AGENT FOR OMA LIGHTWEIGHT M2M". Available Online: `https://fiware-iotagent-lwm2m.readthedocs.io/en/latest/index.html`. Accessed in January 10th, 2019.

[26] Telefońica I+D, "Features". Available Online: `https://github.com/telefonicaid/iotagent-node-lib#features`. Accessed in January 10th, 2019.

[27] Telefońica I+D, "lightweightm2m-iotagent". Available Online: `https://github.com/telefonicaid/lightweightm2m-iotagent`. Accessed in November, 2018.

[28] Telefońica I+D, "lwm2m-node-lib". Available Online: `https://github.com/telefonicaid/lwm2m-node-lib`. Accessed in November, 2018.

[29] Google, "Battery Historian". Available Online: `https://github.com/google/battery-historian`. Accessed in January 3rd, 2019.

[30] N. Armando. "A Unified Management Approach for the Extended IoT.ppt", FCTUC/CISUC/LCT, December 2018.