



FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

José Marcelo da Silva Lopes Fernandes

ISABELA

IoT Student Advisor and BEst Lifestyle Analyzer

Thesis submitted to the
University of Coimbra for the degree of
Master in Biomedical Engineering

Supervisors:

Prof. Dr. Jorge Sá Silva (University of Coimbra)
Prof. Dr. Fernando Boavida (University of Coimbra)

Coimbra, 2017

This work was developed in collaboration with:

Center of Informatics and Systems



Department of Informatics Engineering
University of Coimbra



UNIVERSIDADE DE COIMBRA

Esta cópia da tese é fornecida na condição de que quem a consulta reconhece que os direitos de autor são pertença do autor da tese e que nenhuma citação ou informação obtida a partir dela pode ser publicada sem a referência apropriada.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognize that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.



Acknowledgments

This work was conducted under the guidance of Prof. Dr. Jorge Sá Silva and Prof. Dr. Fernando Boavida, to whom I must express my gratitude for all the support and constant presence, through out this project. I must also show my gratitude toward my colleagues from the research group, for all the companionship, support and work done in this project. Their presence everyday enriched this experience and made all of this work possible.

I extend my gratitude to all my friends and colleagues, that toasted me over the years with joy, happy moments and friendship.

I must also extend my deepest gratitude to my parents, my siblings and to Tatiana Inácio for all the support and love in this 6 year of college. This was only possible because they always made me believe in myself and never let me give up on my dreams.

Acknowledgments

Resumo

Os objetos e aparelhos do dia-a-dia estão cada vez mais inteligentes e interligados. Conceitos como a Internet das Coisas (IoT) e sistemas ciber-físicos (CPS) estão a ser usados para dar a tecnologia, que usamos todos os dias, capacidades de comunicação e sensoriais. Permitindo que estes dispositivos sejam utilizados como sensores, que podem ser usados para obter informação sobre as atividades do dia-a-dia das pessoas. Isto por sua vez, constitui uma grande oportunidade para se fazer investigação nesta área e para criar tecnologias que possam ajudar as pessoas. Porém, estas tecnologias ainda não são muito centradas nos seres Humanos e a maioria dos sistemas veem o ser Humano como um elemento externo ao sistema.

Contudo, ao utilizar o conceito de Human-in-the-loop Cyber-Physical System (HITLCPS) podemos criar sistemas que são mais focados nas pessoas. Este conceito consiste em integrar o Humano como parte dos CPS, isto é, fazer com que o sistema reconheça o Humano: os seus parâmetros físicos, as suas possíveis ações, intenções e estados emocionais. Ao adicionar o ser Humano como parte do sistema nós podemos obter informações sobre ele, processar essa informação para prever estados futuros e fechar o ciclo de controle ao devolver essa informação ao ser Humano.

O principal objetivo desta tese era implementar o conceito de HITLCPS num sistema, capaz de monitorizar os estudantes e de os ajudar a melhorar o seu desempenho académico. Este tese focou-se principalmente no desenvolvimento da estrutura necessária para a obtenção e armazenamento da informação e no desenvolvimento do front-end da plataforma (aplicação Android). Contudo, durante este projeto nós também desenvolvemos alguns algoritmos e aplicamos técnicas de machine learning.

Este projeto incluiu o desenvolvimento de duas aplicações Android (uma aplicação para Smartphone e uma aplicação para Smartwatch), a utilização da plataforma FIWARE e a utilização de dispositivos embebidos (Raspberry Pi e Arduino). Nós utilizamos o Smartphone e o Smartwatch para obter informação sobre os estudantes

e o dispositivos embebidos para obter dados do ambiente. A partir desta informação nós inferimos/classificamos a atividade, localização, sociabilidade e padrões de sono. Nós também recolhemos informação de sensores virtuais/sociais, nomeadamente do Facebook.

Tanto quanto sabemos, este projeto é uma das primeiras tentativas de juntar HITLCPS, IoT e machine learning para criar aplicações centradas nas pessoas.

Abstract

Everyday objects and devices are becoming, more and more, intelligent and interconnected. Concepts like Internet of Things (IoT) and Cyber-Physical Systems (CPS) are being used to empower the technology, we use everyday, with sensors and communication capabilities. Allowing these devices to be used as sensors, that can collect information about people's day-to-day activities. This of course, constitutes a great opportunity to perform research on this area and create new technologies to help people. However, these technologies are still not very Human centric and the majority of the systems sees the Human as an external factor to the system.

Although, by applying the concept of Human-in-the-Loop Cyber-Physical Systems (HITLCPS) we can create systems that are more centered on people . This concept consists in integrating the Human as part of the CPS, that is, make the system aware of the Human: his physical parameters, his possible actions, intents and emotional states. By adding the Human as part of the system we can collect information about him, process that information to infer future states and close the loop by giving that information to the Human.

The main objective of this thesis was to implement the concept of HITLCPS in a system capable of monitoring students and help them enhance their academic performances. This thesis was mainly focus on developing the necessary structure for collecting and storing data and on developing a front-end for the users (Android application). However, during this project we also developed some algorithms and applied machine learning techniques.

This project included the development of two Android applications (one Smartphone application and one Smartwatch application), the use of the FIWARE platform and the use of embedded devices (Raspberry Pi and Arduino). We used the Smartphone and the Smartwatch to collect the personal data from students and the embedded devices to collect data about the environment. From the data, we collected, we infer/classify the student's Activity, Location, Sociability and Sleep.

We also collected data from social/virtual sensors, namely from the Facebook.

As far as we know, this is one of the first attempts to use HITLCPS, IoT and machine learning together to create people centric applications.

List of Acronyms

AP Access Point.

API Application Programming Interface.

CISUC Center of Informatics and Systems from the University of Coimbra.

CPS Cypher-Physical System.

CPU Central Processing Unit.

DB Database.

dB Decibel.

DEI Department of Informatics Engineering.

FP7 Seventh Framework Programme.

GEs Generic Enablers.

GPS Global Positioning System.

HITLCPS Human-in-the-Loop Cypher-Physical Systems.

HTTP Hypertext Transfer Protocol.

ID IDentification.

IoT Internet of Things.

ISABELA IoT Student Advisor and BEst Lifestyle analyzer.

JDK Java Development Kit.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

LCT-group Laboratory of Communications and Telematics group.

lx Lux.

MAC Media Access Control.

Mb megabytes.

min minutes.

NDK Native Development Kit.

NLP Natural Language Processing.

ORM Object-Relational Mapping.

OS Operating System.

PPG Photoplethysmogram.

REST Representational State Transfer.

RFID Radio Frequency Identification.

SDK Software Development Kit.

SSID Service Set Identifier.

UI Users Interactions.

UL2.0 Ultra Light 2.0.

UX User experience.

VGS Version control systems.

XML eXtensible Markup Language.

List of Figures

1.1	Scheduling of the project in a Gantt diagram	4
1.2	GitLab project display	6
1.3	GitLab issues window	7
2.1	Number of connected devices installed base worldwide from 2015 to 2025 (in billions) [1].	16
2.2	Control loop for a HITLCPS. [2]	18
2.3	StudentLife Architecture [3]	19
3.1	Project architecture	23
3.2	ISABELA's FIWARE architecture	24
3.3	Mobile application architecture	28
3.4	Wear application architecture	30
3.5	IoT kit design	31
3.6	Arduino and sensors architecture	32
3.7	Rapsberry and Arduino connection	32
4.1	MainService flow diagram	47
4.2	Example of the communication between 2 distinct services using Local broadcast Manager and intents.	52
4.3	diagram of the operation of the Fiware_communication_service and Fiware_patch_service services	54
4.4	Flow of events when a user sends a message to the ChatBot	57
4.5	greenDAO working flow[4]	59
4.6	Database entities/ORM classes	60
4.7	Sleep Detection neural network structure	68
4.8	Mobile application flow	70
4.9	Introduction screen	71
4.10	Login screen (left) and New Account Registration screen (right)	72

4.11	First configurations screen	73
4.12	Activity screen before (left) and after an alarm is raised (right)	74
4.13	Location chart screen	75
4.14	Sociability chart screen	76
4.15	Navigation drawer (left) and Settings screen (right)	77
4.16	Events screen, with event details showing(right)	78
4.17	Historic screen	79
4.18	Sleep form screen	81
4.19	ChatBot screen	82
4.20	Wear data on the Smartphone	83
4.21	DB on the Smartwatch	86
4.22	Smartwatch screen	87
5.1	Battery consumption information on the	90
5.2	Network traffic reports	91
5.3	Structure of the tested networks	92

List of Tables

3.1	Requirements' parameters	35
4.1	Weight for the Sociability score (equation 4.6)	65
4.2	Sleep classification network performance	68
5.1	Results for network with 1 neurons on the hidden layers	93
5.2	Results for network with 2 neurons on the hidden layers	93
5.3	Results for network with 5 neurons on the hidden layers	94
5.4	Results for network with 10 neurons on the hidden layers	94
5.5	Results for network with 15 neurons on the hidden layers	95
5.6	Results for network with 20 neurons on the hidden layers	95
5.7	Results for different activation function on a network with a 20-20-20 structure	96
5.8	Results training data vs validation data	97

Contents

Acronyms	xi
List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Context	1
1.2 Objectives	2
1.3 Thesis Structure	3
1.4 Scheduling	4
1.5 Methodology	5
1.6 Technologies used	8
1.6.1 Android	8
1.6.2 FIWARE	9
1.6.3 Git - Version Control System	10
1.6.4 Encog	11
1.6.5 greenDao	11
1.6.6 Raspberry Pi and Arduino	11
1.6.7 api.ai	12
2 State-Of-Art	15
2.1 Concepts	15
2.1.1 Internet-of-Things	15
2.1.2 Cyber-Physical-Systems	16
2.1.3 Context Aware Computing	17
2.1.4 Human in the Loop Cyber-Physical Systems	17
2.2 Similar projects	18
2.2.1 StudentLife	18

2.2.2	Smart GPA	19
2.2.3	HealthyOffice	20
2.2.4	BeWell	21
2.2.5	ISABELA vs Similar projects	21
3	Overall System	23
3.1	Overall Architecture	23
3.2	FIWARE architecture	24
3.2.1	ORION	25
3.2.2	IDAS	25
3.2.3	CYGNUS	26
3.2.4	KEYROCK	26
3.2.5	COMET	26
3.2.6	CKAN	27
3.3	Mobile application architecture	27
3.4	Wear application architecture	29
3.5	IoT kits	31
3.6	ISABELA application	33
3.6.1	Context	33
3.6.2	Description	33
3.6.3	Requirements	34
3.6.3.1	Functional Requirements	35
3.6.3.2	Non-functional requirements	42
4	Development	45
4.1	Smartphone application	45
4.1.1	Data acquisition	45
4.1.1.1	Physical sensors	45
4.1.1.2	Context acquisition	49
4.1.2	Communication	51
4.1.2.1	Internal Communication	51
4.1.2.2	FIWARE communication	53
4.1.2.3	Wear communication	55
4.1.2.4	ChatBot communication	57
4.1.3	Storage	59
4.1.4	Processing	61
4.1.4.1	Activity	61
4.1.4.2	Location	62

4.1.4.3	Sociability	63
4.1.4.4	Sleep detection	65
4.1.4.5	Alarms	69
4.1.5	Application Flow and Design	69
4.2	SmartWatch application	84
4.2.1	Data acquisition	84
4.2.2	Communication	85
4.2.3	Storage	86
4.2.4	Display	87
5	Tests and Results	89
5.1	Battery life	89
5.2	Network traffic	90
5.3	Sleep Classification tests	92
6	Conclusions and Future work	99
6.1	Conclusions and Future work	99
6.2	Open challenges	100
	Bibliography	103
	Appendices	111
A	Articles	113
B	List of permissions	135

Introduction

1.1 Context

Nowadays the Internet of Things (IoT) is more than just a concept, we are surrounded by "Smart Things", approximately 20.37 billions of devices, and an expected growth of more 75 billions devices until 2025 [1]. Everyday devices are now improved with sensors, connected to the Internet and can now communicate between them, what allow them to share sensing data [5]. However this sensing data needs to be understood, modeled and reasoned with [6], creating context aware applications that can perceive everything around them.

On the other hand the Cypher-Physical System (CPS) relies on collaborative computing to control physical systems [7]. Combining the unprecedented amount of information that the IoT offers us and the functionalities of a CPS, we can now create new systems that are aware and are able to control the physical world. Technology is always created to improve Humans lives, and this kind of applications described above are no different. If we can apply this technologies to Human centered applications we now enter the realm of Human-in-the-Loop Cypher-Physical Systems (HITLCPS), where technology, accepts the Human as part of the system and takes the Human intents, actions (present and future), emotions and psychological states into consideration, creating systems that better care for Humans needs [8].

College's dropout numbers keep increasing. In Portugal ten out of every hundred students drop out of college in the second year[9]. The causes behind this can be varied but one of the main causes is depression. Depression can come from a lot of factors on the student life, from the lack of sleep and exercise to the pressure to get good grades or financial worries [10]. With the help of the technologies presented above, we can get a better scope of how students lives are and try to prevent this factors.

This dissertation describes the work carried out by some members of the Laboratory of Communications and Telematics group (LCT-group), from the Center of Informatics and Systems from the University of Coimbra (CISUC), in the Department of Informatics Engineering (DEI). The members of this project were myself José Marcelo Fernandes (Master's student of Biomedical Engineering), Prof. Dr. Jorge Sá Silva (DEI) and Duarte Raposo (Doctoral student of Sciences and Technologies of Information DEI).

1.2 Objectives

The aim of the IoT Student Advisor and BEst Lifestyle analyzer (ISABELA) project, which development is described on this thesis, was to create an system capable of understanding the user (students), processing his emotions, actions and intents and help him to perform better on is academic life. Our main objective was to be able to collect student data from various sensors and process it, so we could infer the possible academic outcome and prevent negative ones.

With this in mind, we set the project first objective as the "creation" of IoT platform. This platform should be able to communicate with different devices, store various data, implement security protocols to assure the data security and provide a way to retrieve the data.

The second objective of this project was the development of an mobile application that was able to retrieve the Smartphone sensors data, send it to the platform described above, process the data to infer the emotional, and physical states on the go. This mobile application is also the main front-end for the user and as such should provide relevant and important information. And as the front-end of the platform the Smartphone application is also the main actuator, where we can suggest behavior changes or some advice related to is expected academic outcome.

As the third objective, we decided to focus on the heterogeneity of the system. This point is important because even if we are moving towards an "IoT society" we can not expect the average student to have this devices in his household. And as such we should assure the system heterogeneity ourselves. That is, we should create various smart devices to be inserted in our system. For that to happen, those created devices, should have sensors to collect data as well as the capabilities required to connect with the platform described on the objective one.

As an complementary objective we choose to also implement an Smartwatch

application. The implementation of a Smartwatch in our system, added the possibility of getting biometric data such as the Heart-Rate and it also helped with the third objective described, since this implementation of an SmartWatch application gave our system the possibility of communicate with another type of device.

It should also be noted that this project does not aim to be an emotional, sociological or health study. This is just a first approach towards creating the technology to start such project. Also, towards the end of this project, we had 2 meetings with a group of investigators from the Faculty of Psychology and Educational Sciences. These meetings serve as starting point of a partnership between the 2 groups in order to create a new project, based in the technology created in this project.

1.3 Thesis Structure

This master thesis is structured in six chapters, Introduction, State of the art, Overall System, Development, Tests / Results and Conclusions / Future work.

The first chapter serves as introduction to the project, presenting the context in which it is inserted and the main objectives it had. In this chapter it is also presented a quick overview of the technologies used, the methodology and the scheduling for the project. In the second chapter it is presented a subtle overview of the state of the art, as well as a review of some similar projects and mobile applications.

The third chapter is focus on the Overall system developed, that is the system architecture and its components, where it is given a general overview of all the system components, the IoT platform, the Smartphone and Smartwatch applications and the Embedded sensors. In this chapter it is also given a general presentation of the mobile application ISABELA (Smartphone and Smartwatch version). And it is made a general Requirements overview for the project.

In the fourth chapter is presented the development of the system and all of its components as well as the several steps the project undertake. The fifth chapter is focus on testing and validation of the system, namely on the mobile application. The sixth chapter is dedicated to the conclusions taken from this project as well as an overview of future work and open challenges.

1.4 Scheduling

In this section we make a subtle presentation of the project scheduling that was define prior to the project beginning date, with of course some adaptations as the project was developed.

As we can see from figure 1.1, in the beginning of the project the first month was reserved to do a state-of-the-art review of the existing solutions and projects that resemble ours (more about this on the second chapter). Of course the first two months were also a period of adaptation, because the technologies used in this project are not taught as part of the University courses.

In November we started to develop the two version of the android application (mobile and wear). This included the acquisition of the sensors data and the integration with the FIWARE [11] platform that was assemble at the same time. By the end of January, the first version of the application, was concluded, and we deployed a testing version of it to the members of the LCT-group, to starting collecting data.

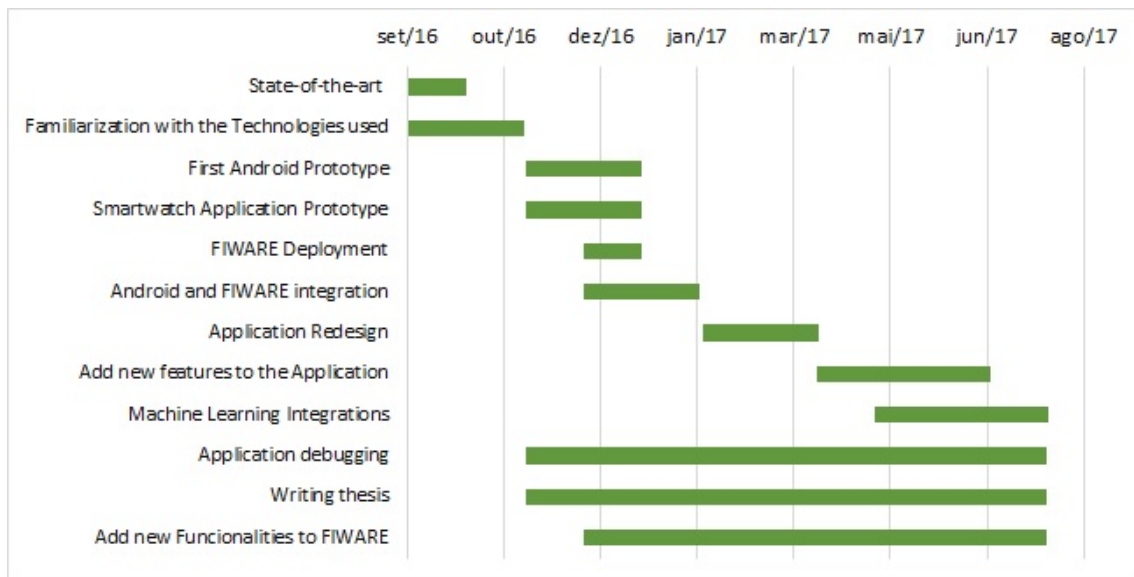


Figure 1.1: Scheduling of the project in a Gantt diagram

The months of February and March, were reserved to start the redesign of the application. This redesign consisted of changes in both the Smartphone application and in the wear application making them a bit more user friendly. This visual changes were accompanied by some functionalities, that were made in the next months. The last months of this master's thesis project were reserved for the implementation of Machine Learning algorithms on the Android application. Regarding

as well the mobile and wear application. There was also a debugging process that started when the first prototype was deployed and continued until the last month of the project. As the project progressed we also needed new modules of the FIWARE, and as such the platform development was treated as an continues process.

As we stated before in the objectives, we also had 2 meetings with an investigation group from the faculty of Psychology and Sciences of Education. This meetings served as a starting point of new collaborative project, where hope to add their scientific background and expertise to our already existing technology. On this new project we will try to infer student's emotions and sociability through continuous monitoring with the Smartphone.

The writing of this thesis, was also taken into account and treat as an continues process, starting in the earliest months of the project and ending in the last one.

1.5 Methodology

In this section we state how the project was planed and developed trough out is duration. The requirements and planning of this project were done before the project started, by Prof. Dr. Jorge Sá Silva and Prof. Dr. Fernando Boavida. Although, with the advances in the project some points were raised and the plan had to be adapted.

In this project we used the Scrum methodology [12]. This methodology consists of doing several iterations in the project, in order to obtain a product that adapts itself to the evolutions of the market. That is, instead of focusing only on planing in the start and moving to the next phase of development only when the previous one is finish, we work with Sprints. Sprints are iterations to the project, they usually take from 2 to 3 weeks and follow the same phases as normal software development : plan, build, test and review. A example of a potential Sprint can be for example the redesign of the User Interface. At the start of every Sprint we plan only the necessary requirements to make the small implementations of this Sprint, in the build we develop those changes, then test them and in the end of the Sprint we review those changes. At the end of every Sprint we should have a potentially shippable product, that we can choose to deploy or not. At the end of one Sprint we evaluate the project start a new iteration of the project (Sprint), if one is needed. This methodology is very simple and makes the development of software more easy, since sometimes is hard to have a full image of the final product right from the start.

1. Introduction

When the project started we were aiming to develop our own IoT platform, but after we had done some research about the state of the art we decided to implement an already existing platform, the FIWARE (explained further in the next section 1.6). Also new functionalities and capabilities were added to our solution as we dimmed important, either in the mobile application or in the FIWARE.

Through out the development of the project every week, on Fridays, we had a meeting with all the LCT-group members to discuss the development of this project, as well as other projects that the others members of the group were working on. These meetings started always with an point of situation presentations, letting all the members of the group know what the others were working on. Although these reunions were not only informative they were also used to get the opinion of all the members on the project's next steps. In addition every Monday we had individual reunions that were used to plan the week work and to expose problems and difficulties.

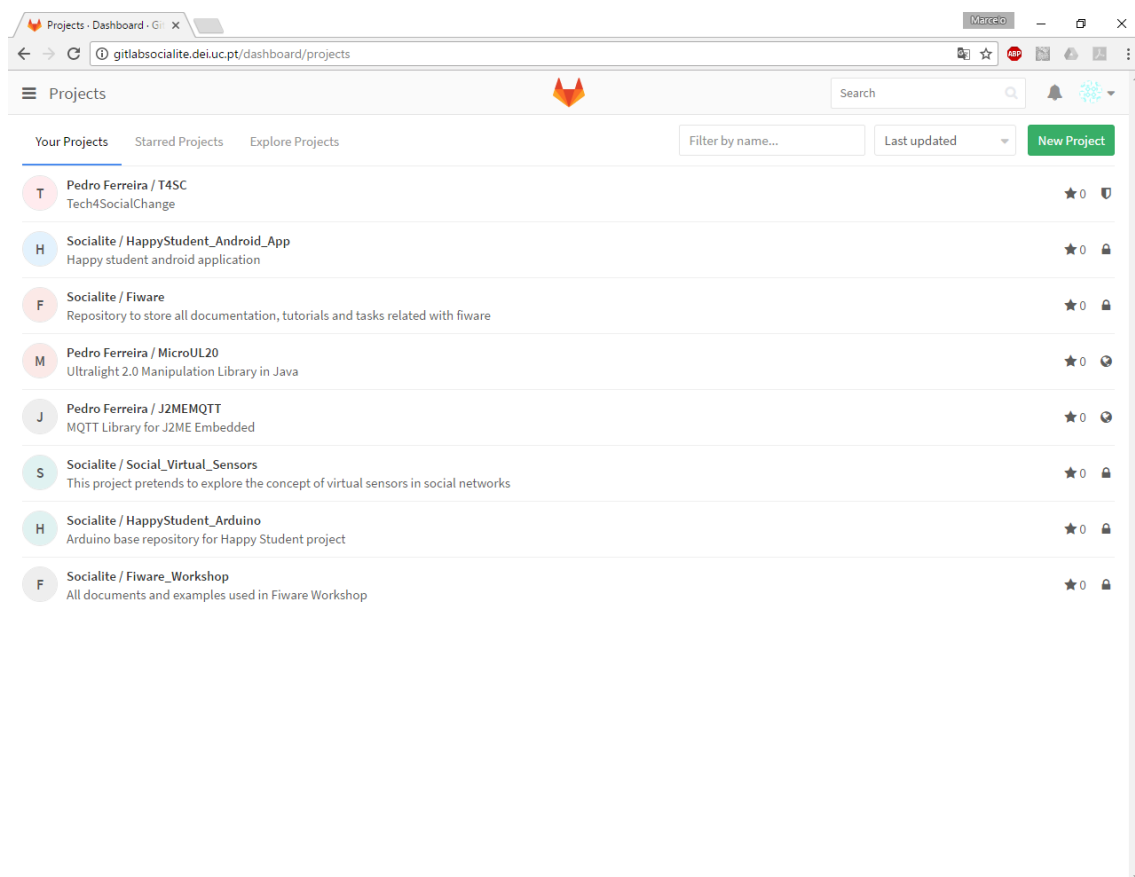


Figure 1.2: GitLab project display

To further improve the working strategy, and the collaboration between everyone, we also use an instance of the GitLab [13] (further explained on the next section 1.6) that worked not only as a version control system, but also as a team management tool, this helped us implement a very Scrum[12] like system. As we can see from figure 1.2 all the members had access to the different projects, that the group was working on. This allowed us to expose the work done, making the other members aware of our work, and create new issues on the projects. These issues could be labeled (Fix, Developing, Completed, Closed, Fixed), assigned to specific users and we could attribute them a deadline as can be seen in figure 1.3. This tool helped to coordinate the work, but also in the debugging process, since a bug could be reported as a new issue to be fixed.

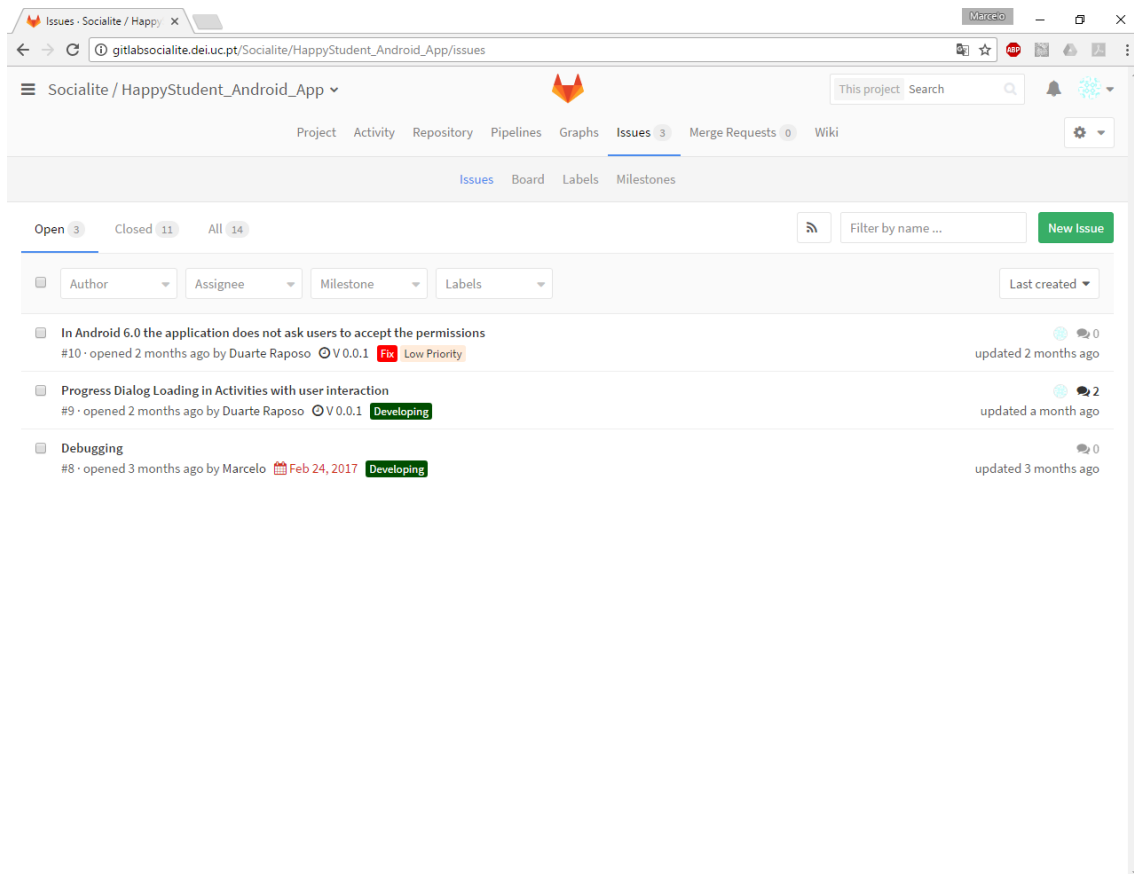


Figure 1.3: GitLab issues window

Also as form of communication, we utilized Slack [14], that allow us to separate the different projects in different channels. This way we were able to send and share relevant information to the members of the each project, we could also create topics for specific problems and find faster solutions this way. Skype [15] has also use as a means of communication when some of the members couldn't attend a meeting.

1.6 Technologies used

In this last section of the chapter we address the technologies used during the project and present a quick resume about them. As the main component of this project is the Android application, the most used technology in this project was Android Studio [16].

1.6.1 Android

The Android is an Operating System (OS) designed specially to mobile devices. It is based on the Linux Kernel [17] and is designed by Google. This OS is made to run in devices with touchscreen, like Smartphones, and the main interaction with it is trough touch gestures and it also has a virtual keyboard for text inputs. When the System started it was only designed for Smartphones and Tablets, but over the years new versions of the OS appeared, such as the Android TV for Smart TVs, the Android Auto for cars and the Android Wear [18] for Smartwatches. The last one is used in this project and so it requires further explanation.

The Android Wear as the name indicates was meant to be used in items that people wear, but until this day is only used in Smartwatches. This OS is for small devices, with even smaller screens and was designed to be used without hands, that is, almost all the features are available trough the use of voice command. We can also interact with it trough gestures like swipes and clicks and the newer version of this OS, Android Wear 2.0, comes with a virtual keyboard for text input (in this project we only used Android Wear 1.5 version). Android Wear was made to work as complement for the Smartphone, with it we can see notifications with a glance at the wrist instead of having to draw our phone from the pocket. But these devices are also designed for fitness and sports purposes and are embedded with some sensors that are not available for Smartphones, like the heart rate sensor and the pedometer sensor.

The Android is the most used OS in mobile devices with more than 1.6 billions devices in the world and with a 87.5% share of the mobile OS market [19]. Google also has a store for mobile applications, Google Play Store, where the majority of the Android applications and games can be acquired. The more recent numbers by Google indicate that there are more than 2 millions applications available at the Google Play store [20].

The Android OS, as stated before is based on Linux Kernel but it also runs a Java Virtual Machine (JVM) developed specially for these devices. The main programming language used in the development of Android applications is Java, by using the Java Development Kit (JDK), but some low-level programming can be done in C++ or C with the Native Development Kit (NDK).

The Android OS is a open-source technology, and as stated above it is the more used mobile platform in the world. These are the main reasons we choose this platform for our application. In addition to it the Android platform has also a large community of developers and many third party libraries with good documentation that ease the development of applications.

1.6.2 FIWARE

The FIWARE is an IoT platform that was developed as part of the Seventh Framework Programme (FP7) [21], and started in 2011. The FIWARE name stands for Future Internet Ware. The main objective of this project was to advance the process of harmonizing the European Technology platforms, in this case the IoT platforms. FIWARE is also an public and open-source [22]. The process of creating the FIWARE was divided into three phases. The first phase (from May 2011 to April 2014) was aimed at creating the technological core called FIWARE, while the second phase (from April 2013 to March 2015) was mainly aimed at the implementation of FIWARE nodes. Finally, the last phase covered from September 2014 to September 2016 and aimed primarily at creating a sustainable ecosystem for Small Medium Enterprises, through the selection of sixteen business accelerators.

The FIWARE Foundation was created to assure the project success, and they describe themselves as an legal independent body providing shared resources to help achieving the FIWARE mission by promoting, augmenting, protecting, and validating the FIWARE technologies as well as the activities of the FIWARE community, empowering its members including end users, developers and rest of stakeholders in the entire ecosystem [23]. This foundation also manages semi-annual summits for all members of the community.

FIWARE is based on a library of components called Generic Enablers (GEs) that are meant to implement Application Programming Interface (API)s. GEs offer reusable and commonly shared functions “as a Service”. Through APIs, GEs allow developers to put into effect functionalities making programming much easier by combining them. GEs are classified into seven technical chapters [24], namely:

- **Cloud Hosting.** A set of components to provide computation, storage and network resources on top of which services are provisioned and managed.
- **Data/Context Management.** A set of components to facilitate the access, gathering, processing, publication and analysis of context information at large scale.
- **Architecture of Applications / Services Ecosystem and Delivery Framework.** A set of enablers to co-create, publish, cross-sell and consume applications or services, addressing all business aspects.
- **Interface to Networks and Devices.** A set of components to make the most of the network infrastructure capabilities, building communication-efficient distributed applications, exploiting advanced network capabilities and easily managing robotic devices.
- **Security.** A set of components to provide mechanisms to make delivery and usage of services trustworthy by meeting security and privacy requirements.
- **Internet of Things Services Enablement.** A set of components to leverage the ubiquity of different devices, making connected things available, searchable, accessible, and usable.
- **Advanced Web-based User Interface.** A set of components to facilitate the use of 3D and Augmented Reality capabilities in web-based user interfaces.

Not all the GEs documented above are used during this project, but the implementation of FIWARE can be found in chapter 3.

1.6.3 Git - Version Control System

The use of Version control systems (VGS) is one of the best practice for software development. These systems allow us to store all the developed software remotely to ensure the project continuation and prevent any drawbacks in case of computers malfunction.

But this tool capabilities go beyond the repository functionalities. They also allow us to keep track of all the changes on the project files, letting us know when the files were change and by whom, and even what section of the file. This helps prevent several errors and save time as it is very easy to make a rollback in the project, when something undesirable happens [25]. Other functionalities also include the merging and branching of projects allowing us to keep different versions of the

same software, for instance we can keep a stable version on a branch while working on new functionalities in a different branch of the project.

As explained on the previous section (1.5), the version control system used during this project was a local instance of the GitLab. Using a local instance, with authentication assured that all the applications and code were secured and private.

In our case the version Control system was as well the team management tool. Allowing us to keep track of everyone work, as well as coordinate the work of the different project members. Creating issues, be it for debugging or for new tasks, allow us to keep track of the work that had to be done.

1.6.4 Encog

Encog is an machine learning framework that supports several algorithms, as well as support classes to normalize and process data. When this framework was first released it only supported Artificial Neural Networks, but now it also supports different algorithms like Hidden Markov Models, Support Vector Machines[26].

This library has an Java implementation, what allow us to use it on the Android platform. The fact that it supports multi-threading is also a plus, since it provides us with better training performances.

1.6.5 greenDao

The greenDAO is an open source Android Object-Relational Mapping (ORM) framework that allows us to create SQLite databases. Writing SQL and parsing query results are sometimes difficult and time-consuming tasks. This framework frees us from these tasks by mapping Java objects to database tables [4]. This way we can create tables, store and update data or even delete it using object oriented API.

This framework also allows for database encryption what is a plus for us, since the security of the data we use is an important requirement of the project.

1.6.6 Raspberry Pi and Arduino

In this project we use two types of embedded circuits: the Arduino[27] and the Raspberry Pi[28]. These boards at the first glance look the same, but are very differ-

ent. While the Arduino is a microcontroller, the Raspberry Pi is a microprocessor, that is, the Raspberry Pi is a small computer with an OS, in turn the Arduino has no OS, has a less powerful Central Processing Unit (CPU) and lower memory.

While the points above make the Raspberry Pi look better than the Arduino, the truth is that we first need to consider the task at hand before choosing between them. The Raspberry Pi as stated is more powerful, multitasking and is able to process large data. But the Arduino is more suitable for small controlling task because it doesn't need an analog to digital converter. So tasks like turning the air conditioning on and off to maintain the temperature of a room are more easily done with the Arduino. The Arduino however, as stated before, has no OS and as such can only run a program at a time and repeatedly.

In our case we used both of them together. We used the Arduino to connect all the sensors and convert the values of Voltage to real values such as Decibel (dB)s and Lux (lx) and we used the Raspberry Pi to handle the communications. This implementation is further detailed in chapter 4.

1.6.7 api.ai

The api.ai [29] is a framework from Google that allow us to create agents, capable of doing Natural Language Processing (NLP).

In these agents, we can create entities that can represent physical objects or even concepts. For example we can create the Entity sensor and add the Temperature attribute, Light attribute and Sound attribute to this entity. We can also define synonyms that can be used to describe these entities, for example the sound sensor could be called the microphone.

We can also create Intents that have associate to them actions. These intents are the base to the NLP, in them we define possible phrases that the user may "say" to the agent. These examples may contain specific entities, what facilitates the classification. We can define one or more response message to which Intent. We can also define the Action that this intent represents and for more complex actions we can implement the business logic for that specific action on the user's front-end (in our case the mobile application).

By adding intents, and by making them as specific as possible, the agent can process the received messages and classify them into those Intents. Whenever a message is not recognized a Default Fallback Intent is triggered. There could also

be misclassifications, and we can deal with them as well, the platform allow us to review every call to the agent change it's classification and retrain the agent.

The development of the ISABELA's ChatBot is explained in more detail in the fourth Chapter.

State-Of-Art

In this section we will introduce some of the concepts used during the development of this project, as well as some of the application and projects with similar objectives. We make a slight comparison of the similarities and differences between these projects and ours.

2.1 Concepts

In this section we address some of the concepts and paradigms used throughout the project. These concepts were fundamental for all phases of the development, and as such we think that a brief summary of them should be made.

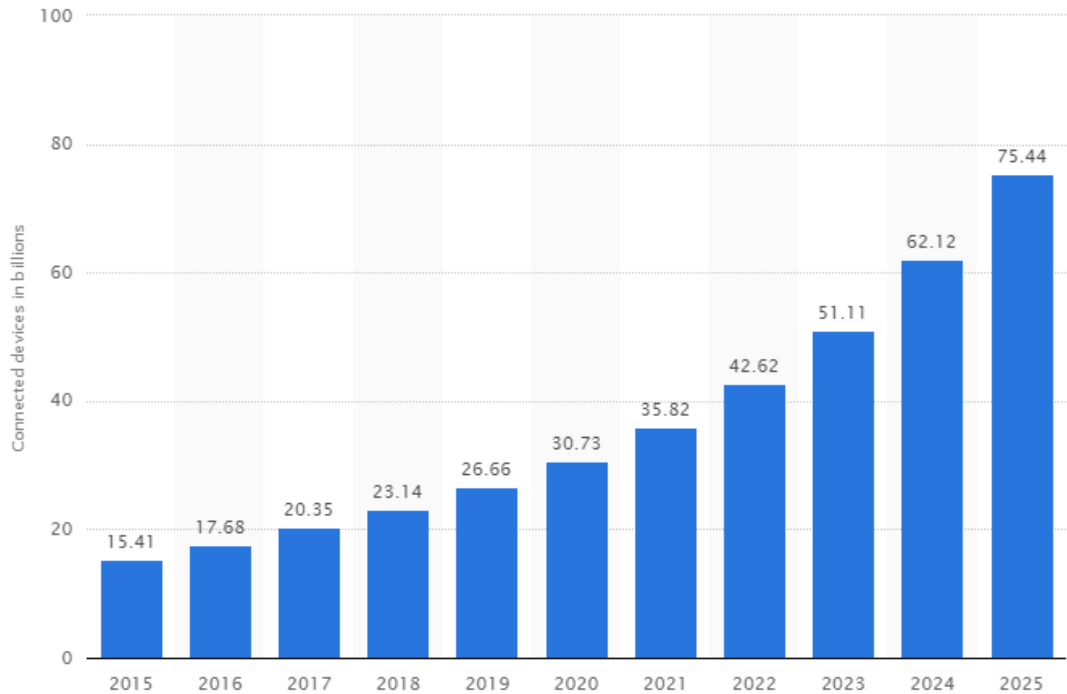
2.1.1 Internet-of-Things

The term "Internet of Things" was proposed, in 1999, by Kevin Ashton. He was then referring himself to Linking the new idea of Radio Frequency Identification (RFID) with the supply chain of Procter & Gambler (a company for whom he was making a presentation) [30]. He also referred to the concept as the possibility to empower computers with means to gather information by themselves so they can perceive the world without Human input.

Nowadays the concept evolved, and no longer refers to RFID, instead now we have objects with embedded systems capable of communicate with each other, produce sensory information and with self-configuring capabilities based on standards and interoperable communication protocols [5].

Today there are more than 20 billions of devices connected to the Internet and it is estimated that the number triple by 2055 [1] as we can see in figure 2.1. These devices are no longer only computers, tablets or smartphones, there are also

many everyday objects that are now empowered with processing and communication capabilities (such as fridges and TVs).



© Statista 2017

Figure 2.1: Number of connected devices installed base worldwide from 2015 to 2025 (in billions) [1].

Some first approaches to IoT applications have been made in health-care, transportations and more recently towards smart-cities, where connected devices with sensor capabilities are able to share information. And in a near future these devices will be even more common and in the grasp of everyone. But there are still much work to be done to make this systems capable of interacting with Humans in a more intuitive and meaningful way.

2.1.2 Cyber-Physical-Systems

As stated before CPS is the concept of using collaborative elements of a Cyber system to control physical aspects of the world, CPSs are defined as systems that offer integrations of computation, networking, and physical processes [7]. IoT and CPS are different concepts, but apply some of the same principles, with IoT focusing more on the networking and integration of different devices while the CPS focus more on the applicability and modeling of physical systems [31].

These systems are then a junction of sensor, decision and actuation capabilities. If we apply these capabilities to monitor Humans we are entering the Concept of HITLCPS, CPS that take Human actions and emotions into consideration [8].

2.1.3 Context Aware Computing

As we mentioned previously, the IoT provides us we large amounts of sensor data, that needs to be reason with and contextualized [6]. There are many ways to contextualize and process data, but in this present case we used supervised machine learning to process and data and acquire classifications.

As such the IoT paradigm needs to be supported by middleware solutions. *“Middleware is a software layer that stands between the networked operating system and the application and provides well known reusable solutions to frequently encountered problems like heterogeneity, interoperability, security, dependability“* [32]. In our case the solution is the FIWARE that supports data contextualization through the ORION (Context Broker) model [33].

2.1.4 Human in the Loop Cyber-Physical Systems

As stated previously if we use the IoT and CPS paradigms together to make Human centered applications we can we enter the realm of HITLCPS. Most of the IoT applications nowadays don't take the Human as an integrating part of the system. And if we want to create technology that better serves Humans we need to create systems capable of understanding Humans' actions and intents as well as their emotions [8].

In figure 2.2 we can see that in order to have a system with a close loop we need four components: Data acquisition, inference (contextualization), future inference and actuation. All of these phases are equally important and demanding, and the concepts presented in this chapter can be linked to each of this phases. We are applying the IoT concept to acquire data, with context-aware computations we process it, be it in the present or the future, and we are applying the concept of CPS by implementing behavior change interventions.

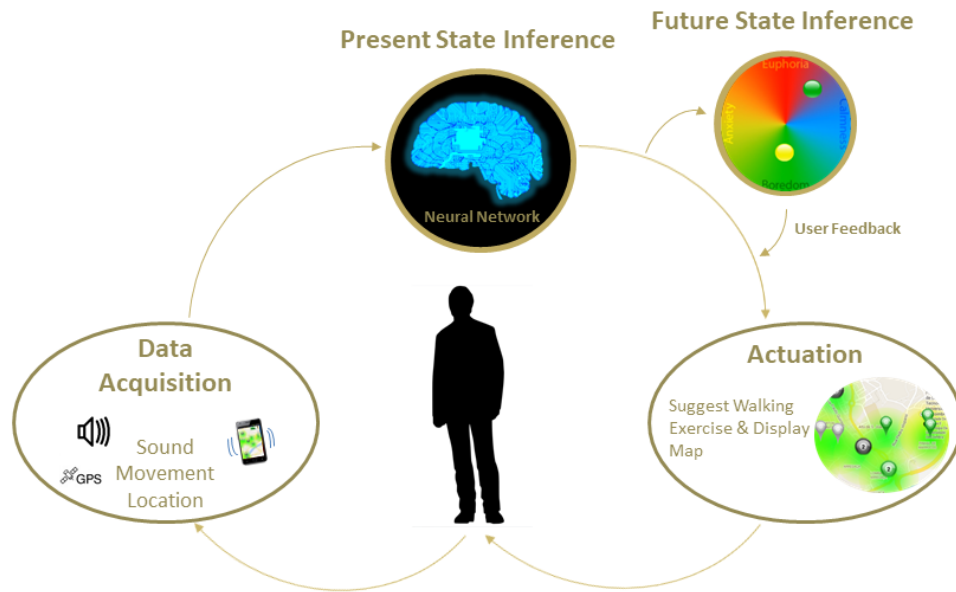


Figure 2.2: Control loop for a HITLCPS. [2]

2.2 Similar projects

In this section we present some projects that are similar to ours. We are only considering project that are centered on smartphones applications and not on all kinds of HITLCPS projects.

2.2.1 StudentLife

This study was developed by a research team in the Dartmouth College. They developed a mobile application for the Android OS, that was used to monitor the students and collect data from their day-to-day activities [3].

As can be seen in figure 2.3 they collect data from five sensors: accelerometer, microphone, light sensor, the GPS and Bluetooth. They used these 5 sensors to obtain context data about the user activity, conversation patterns, sleep periods and location. They then combine that context data with some surveys and correlated them with the students' academic performances, mental health and with the

Dartmouth term life cycle .

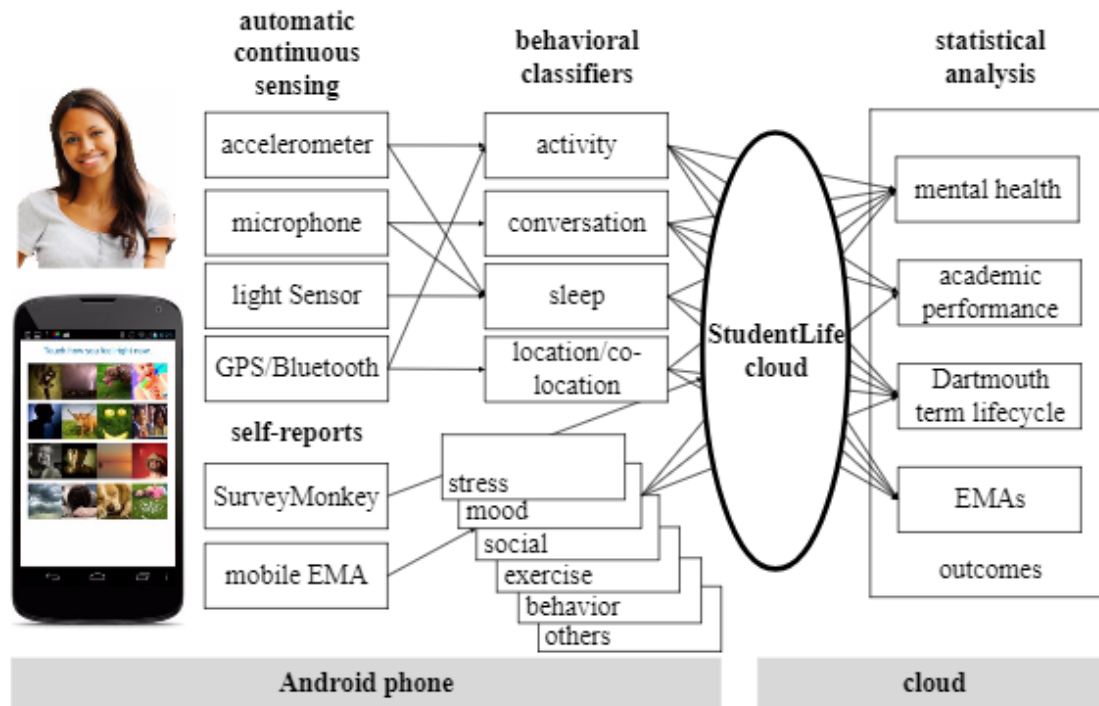


Figure 2.3: StudentLife Architecture [3]

The study was conducted through a ten weeks period and they found some correlations that were very interesting. For example they correlate the sleep patterns with the duration of the study and found that the midterms were when the students sleep less. They also found out that the stress tends to increase with the midterms, and with the number of deadlines that the students have.

Although this study is the main inspiration to ours, they differ in various points, namely we aim to develop a HITLCPS, that is, we intend to offer feedback to the student and prevent bad academic performance while their study was mainly statistical. There are also other differences, such as, the use of an IoT architecture in our system, collecting data from more sources than just the smartphone, the smartwatch application on our project, the use of ChatBot as means of interaction with the students and the use of virtual/social sensors.

2.2.2 Smart GPA

The same researcher group from Dartmouth College, continuing his previous work from StudentLife, created a new study with the aim of predicting the academic

outcome of students. For that they use a subset of the StudentLife dataset, that is available online.

In this study they used activity, sociability, sleep, personality, class attendance and studying as features for the classification. The personality features were taken from the psychological surveys' data and the other features were taken from the Smartphone sensors data. They obtained good results with error in the predicted GPA (Grade Point Average) of ± 0.179 [34].

This study is a good starting point for systems that monitor students results, but in their particular case they did not try to change the outcomes or even predict it in real time. And so this work does not apply the concept of HITLCPS.

2.2.3 HealthyOffice

The HealthyOffice[35] was a study developed by a research team from the Telecommunications Research Laboratory of Toshiba Research Europe in collaboration with the University of Southampton and they developed a system to predict and monitor the mood of office workers.

In this work they focus themselves on physiological data from wearable sensors. They used the Toshiba SilmeeTM to obtain data of an Electrocardiogram, the Photoplethysmogram (PPG) to obtain the heart rate and a sensor to measure skin temperature. To obtain the ground truth for the mood they created an Android application namely the HappyOffice application, where the users could give information about how they feel. The application was also able to show information to the user, in charts or text messages [35].

Although this study focus group is different from ours, they implement a very HITLCPS like architecture, that resembles our proposed architecture. However they lack a better mechanism to close the loop and give feedback. Their main method of closing the loop is by giving aggregated information about the average state of the mood in the office to the employers so they could do something about it. Our study also differs by using the Smartphone as the main method of collecting data instead of wearables.

2.2.4 BeWell

BeWell is an Android application developed by a team with researchers from the Dartmouth University and the IT University of Copenhagen. In this project they tried to access person well being through smartphones. They measured physical activity, sociability and sleep through smartphone sensor, namely GPS, accelerometer and microphone.

In this study they make a close loop approach where they, monitor the people's day-to-day activities passively, infer their well being and inform them about the assessment made. In their work they developed two ways to give informations to the users, the first through a widget in the smartphone where they showed the users their stats to the sleep, activity and sociability measures. And the second a web portal where the user can see more information [36].

This work was done back in 2010 and as such, was just one of the first attempts of monitor peoples well being. Our work differs from theirs because we use more sensors in the smartphone, we used only the smartphone to present the information to the user, our work is focus on students and we used an IoT architecture.

2.2.5 ISABELA vs Similar projects

As we stated before there are several differences between our project and the previous projects in the are of mobile monitoring. In this section we make a overview of those differences.

The main difference is, without doubt, the implementation of a HITLCPS architecture where we give feedback. Apart from the BeWell project none of the other gave feedback to the users and the BeWell only showed the data and did not give any suggestions on how they could change their behaviors. Our system gives feedback to the users through ChatBot messages, and allows the user to engage in a conversation with the ChatBot asking it what they can do to change a certain behavior.

Another difference in our system, when compared to the other, is our architecture. Our system's architecture does not implement a simple server-mobile application that is common in the majority of mobile application and in the similar projects. We implemented an IoT architecture with an IoT middleware platform that implements the communication protocols needed to communicate with all kind

2. State-Of-Art

of embedded devices. This in turn allows us to create new scenarios with more devices as we please, in the future.

We also use virtual/social sensors on our project and intend to explore more this area in the future. In comparison none of the project presented above explore this possibility.

Overall System

In this chapter we present the system in a overall way. In the first section we present the architecture of the system and of its components separately. In the last section we present the ISABELA Application and its requirements.

3.1 Overall Architecture

The ISABELA application is a mobile application developed to promote good academic results. The application context is explained in further detailed in section 3.2.

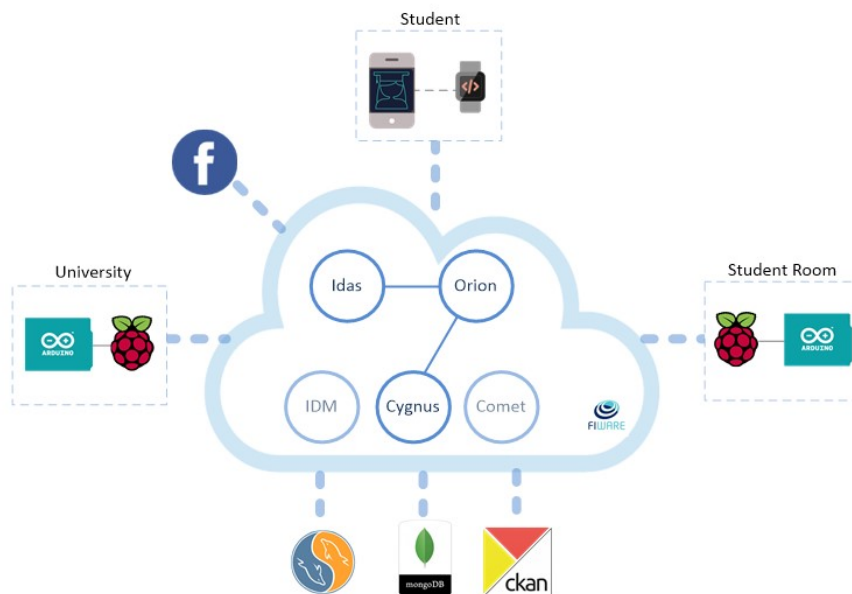


Figure 3.1: Project architecture

As explained previously to create a system that implements HITLCPS we need to implement four main capabilities: data acquisition, inference, future inference and actuation. With that in mind we created the system architecture of figure 3.1. Where the Smartphone, Smartwatch and embedded sensors are used to acquire data, the FIWARE is our cloud based module that implements the storage capabilities and allows the communication between all the devices. The Smartphone is also where all states are inferred and where the actuation is made through notifications and messages. The Smartwatch is the only device that doesn't communicate directly with the FIWARE, because of its architecture, but that implementation is explained with further detail in chapter 4.

3.2 FIWARE architecture

As stated before the FIWARE is our back-end component that implements the storage and communications capabilities normally seen in mobile applications systems. In figure 3.2 we can see that the FIWARE itself is divided in several modules, namely the ORION, the IDAS, the CYGNUS, the KEYROCK or IDM, the COMET and the CKAN. There are more modules on the FIWARE but only this six were used in this project, to this date.

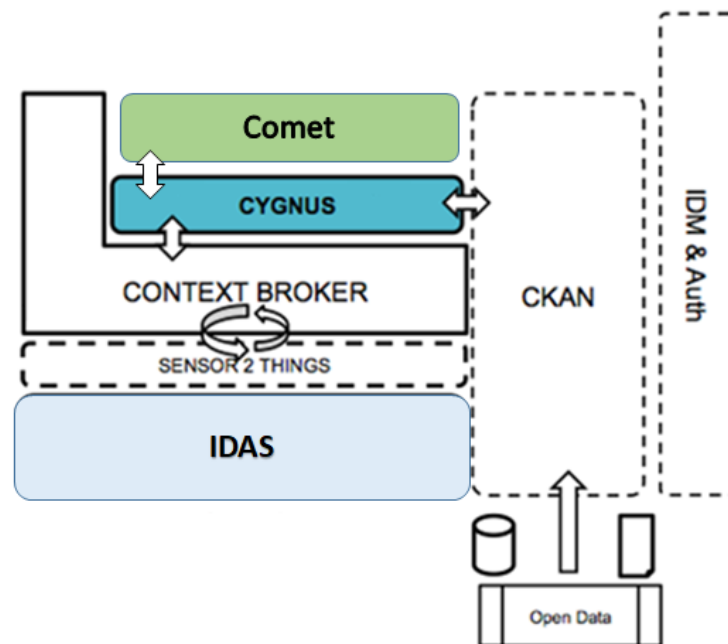


Figure 3.2: ISABELA's FIWARE architecture

3.2.1 ORION

The ORION or context broken, as its name indicates, allows us to create context. More precisely it allows us to create virtual entities to represent objects in the real world or even people [33]. These entities are like classes they have a type, a specific id and attributes. However unlike classes these entities can have different attributes even if they are of the same type, for example in our case study some entities of the type student can have the attribute heart rate while other may not have it, depending if the student in case has a Smartwatch or not.

This is of course a must have capability on an IoT architecture where we want to create a connection between the sensors and the applications that consume the information. For this purpose this module implements a Representational State Transfer (REST) API, that allow us to create, update and delete entities or attributes.

Although, this module allow us to create entities and save their attributes, it is only able to store the last instance of that data. That is if we update an attribute value, the module only retains the last value. To save data for historic context we need to implement other modules as the CYGNUS and the COMET.

3.2.2 IDAS

The IDAS module is the part of our system that handles the communication with the embedded devices. Unlike the smartphone that communicates directly with the ORION, the embedded systems communicate with the IDAS, that then registers a new device and creates a subscription from that device to an entity [37]. For example we can connect an Arduino with a light sensor to the IDAS and subscribe that sensor to a entity type room in the ORION, and every time that the sensor sends a new value that value is updated to the entity.

But as stated before the main task of this module is handling the communication with embedded devices that do not use Hypertext Transfer Protocol (HTTP) protocols, allowing the use of constrained devices, which allows our system to integrate more devices thus increasing the possible heterogeneity of the same. Although in our specific case we only use Raspberry Pi 3 to communicate with FIWARE, and this embedded system supports HTTP so we are using it as means of communication. The IDAS also supports Ultra Light 2.0 (UL2.0) a JavaScript Object Notation (JSON) representative protocol that allow us to send shorter messages [37].

3.2.3 CYGNUS

The CYGNUS module is responsible for coordinating the storage of the data. That is, in this module we can create subscriptions from the entities to a specific third party storage system, such as MongoDB[38] or MySQL[39], creating this way a historic view of that data [40]. These subscriptions are made by type, and after the subscription is made, for a specific type, whenever an entity of that type is created, updated or altered the CYGNUS automatically saves those changes to the third party database to which the subscription was made.

3.2.4 KEYROCK

The KEYROCK is an authentication module that manages all the other modules accesses, that is by creating accounts we can restrict the access of certain information and functionalities to certain accounts. This allow us to solve a number of issues with users' access to network, applications and services, also this way we can secure the data and assure privacy[41]. This module implements a single sign-on service, that is the users' credentials (email and password) are the same to all modules and are hosted inside of this module. When the user signs on he is given an Access Token that is then used in the requests he makes to the remain modules. Those modules query then the KEYROCK, to validate the user request, before allowing the user to change or access any data.

This a very important module since the privacy of the data is one of the main requirements to this project and with this module we can prevent personal data from leaking to third parties, or even from being access by users which shouldn't have access to them.

3.2.5 COMET

This module is a short time historic, in charge of managing historical context information, storing and retrieving it[42].This module implements REST APIs to communicate with the ORION and with third parties (like the ISABELA application). The API used to communicate with external applications was developed to facilitate the retrieving of data with time context, that it this API allows us to aggregate the data by time or even query specific time intervals. The API also allows us to aggregate data with sums or by occurrence (to discrete values, like strings).

This module is from where we get most of the data in the application, and as such that makes it a very important component to keep the platform working. This module also addresses the requirements for storing information in the project as well as the requirements to have information contextualized in time.

3.2.6 CKAN

The FIWARE also offers support to initialize a CKAN dependency on our architecture. However the CKAN is not a FIWARE module. The CKAN is the world leading open-source data portal platform, that is a powerful data management system that makes data accessible [43].

This platform much like COMET allows us to store and retrieve data, but this module does not offer an API so "powerful" as the one from the COMET to retrieve data with historic context. However this platform has its advantages, for instance this platform provides us with a Web interface that we can use to see the data. Another big factor for the use of this platform is that this project is a case study for the project SOCIALITE, and in this project the data that we collect is used by another group of investigators, that try to create new models for data analytics and artificial intelligence.

3.3 Mobile application architecture

This project's mobile applications follows the design patterns of a common mobile application, where we have a thread that handles the graphic display and Users Interactions (UI) (UI thread) and an another group of threads that run in the background to make everything work properly. This architecture allows us to have a smooth application where the display is not affected by tasks that required a lot of computation time.

The UI thread is in general responsible for small tasks like show a chart or change the settings. This thread needs to be as free as possible to interact with the user and this prevents malfunctions that otherwise occur. This thread is in charge of displaying Activities and implementing interface function like sending data, or changing a text when a button is pressed. Because of this, this thread is responsible for starting the background threads and is also linked to the background threads to exchange information. This communications is explained in further detail in the

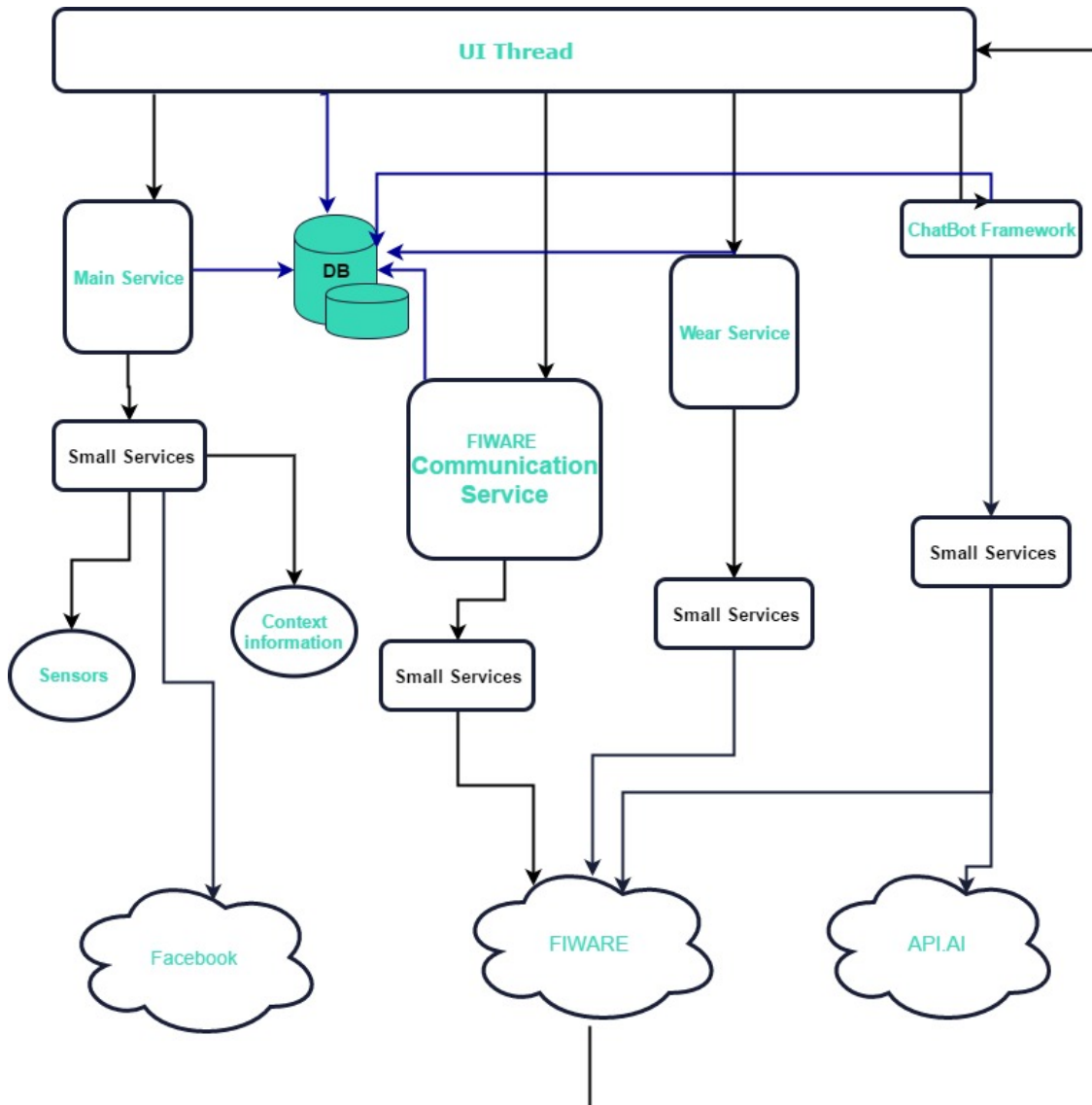


Figure 3.3: Mobile application architecture

fourth chapter.

The background threads or background services handle the more complicated tasks like business logic, HTTP communication, data storage or even repetitive tasks like sensor data retrieval. In our application there are two main background services that are running continuously: the Main service and the FIWARE communication service. The Main service as can be seen from figure 3.3 is responsible for obtaining the sensors and context data, and the FIWARE communication Service is responsible for sending data to the FIWARE.

In figure 3.3 the small services blocks represent a set of specific services that were made to do some short-term tasks. These services are an extension of the

IntentService Class [44], that class implements her own thread and request pool, and execute the requests one at a time. These implementations are further explained in detail in the fourth Chapter.

The Wear Service is responsible by the communication with the Smartwatch, as well as sending the data that is received from the Smartwatch to the FIWARE. This process is also explained in detailed in chapter 4.

The ChatBot framework is made to process the user messages, send them to the API.AI, communicate with the FIWARE to fetch information that the user request and to give Alarms. These implementations are explained in detailed in the fourth chapter.

On figure 3.3 is also visible a Database (DB) that is used by the main services as well as the UI Thread. Although all the data we collect is sent to the FIWARE for safeguard, we dimmed necessary to have a DB in the mobile application to store the data when no Internet connection is available.

3.4 Wear application architecture

The Smartwatch is not as powerful as a Smartphone, and as such the architecture of the application on the Smartwatch has to be smaller. As can be seen from figure 3.4, the wear application architecture like the mobile application, has an UI Thread to handle the UI and background services to handle the more complex tasks.

From figure 3.4, we can see that the architecture of the Smartwatch's application is more minimal with just two background services. The Main service, like on the Smartphone's application, takes care of the sensors' and context's information acquisition and the Wear Service handles the communication with the Smartphone.

Like in the Smartphone application, we also felt it was necessary to have a DB on the Smartwatch to ensure that no data is lost before being sent to the Smartphone. This architecture is explained in more detailed in Chapter four.

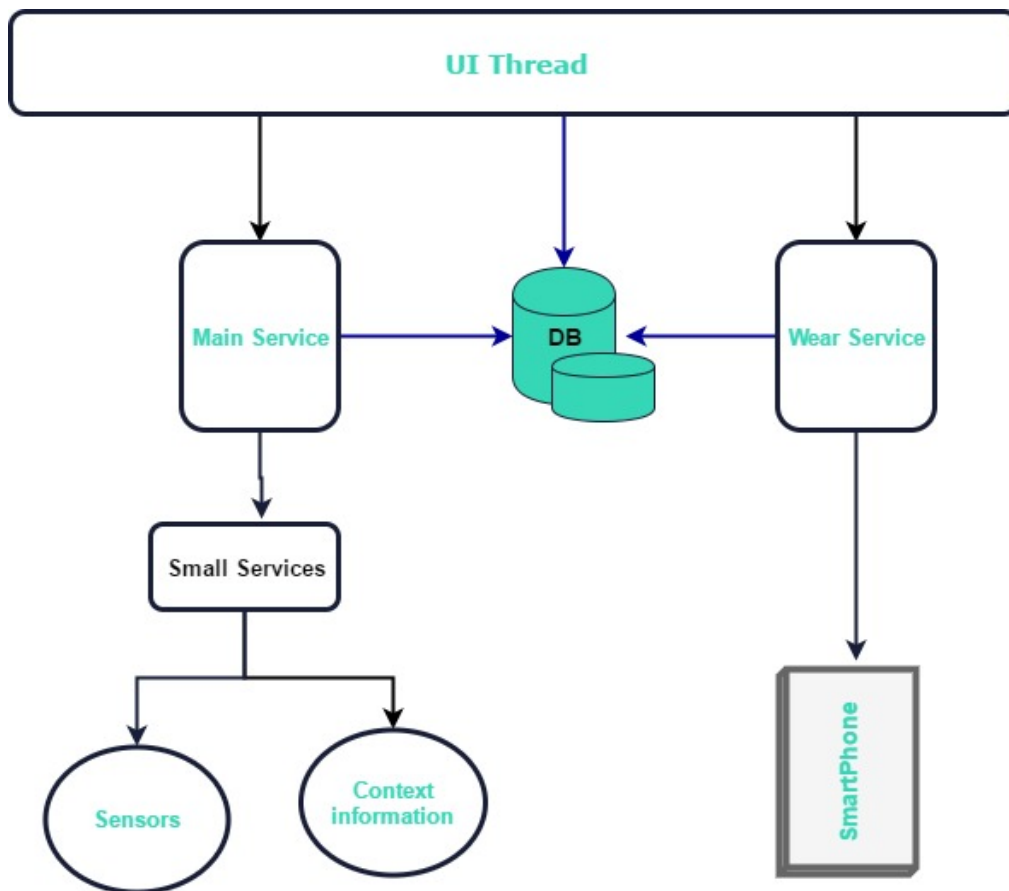


Figure 3.4: Wear application architecture

3.5 IoT kits

As was stated before we developed a kit composed by Arduino and Raspberry Pi and some environmental sensors, namely a Temperature sensor, a Light sensor and a Sound level sensor. The kit final aspect can be seen in the figure 3.5. The position of the 3 sensors we used can also be seen from figure 3.5, the Light sensor is on the top of the box, the sound (left) and temperature (right) sensors are on the front side of the box.

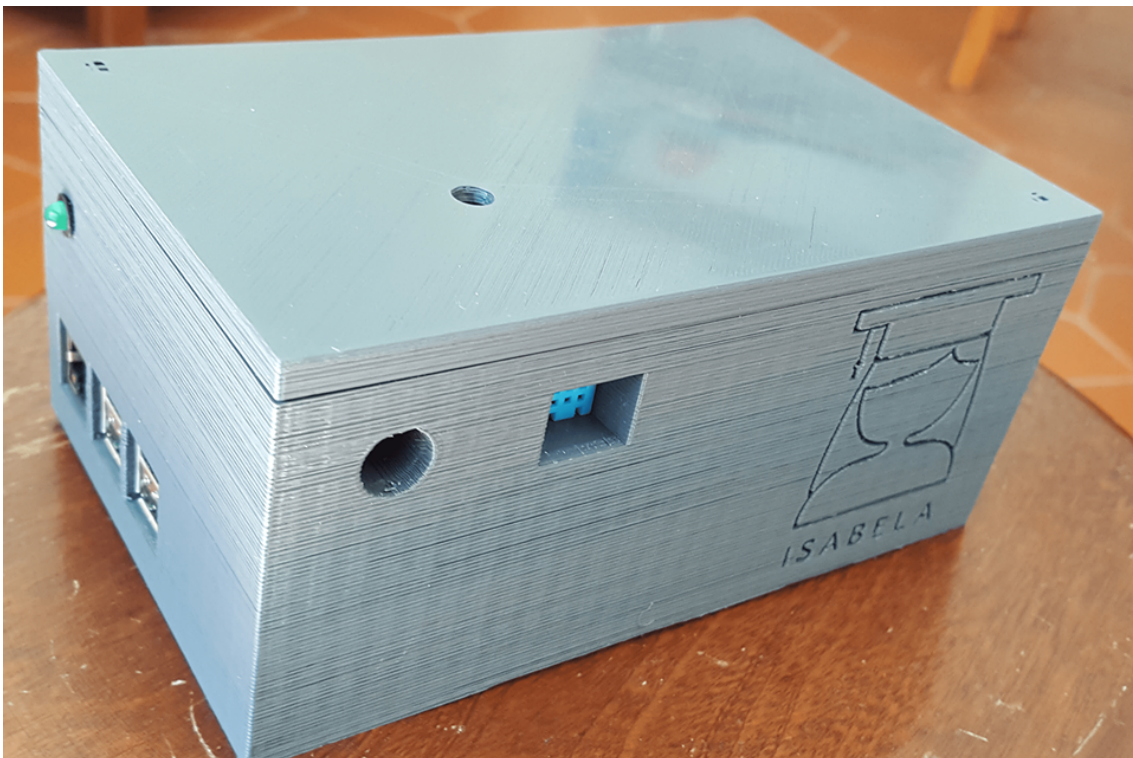


Figure 3.5: IoT kit design

As it was explained before we used the Arduino to process the analog data from the sensors. And the architecture can be seen in figure 3.6, where we can see how the sensors are connected. All the sensors are connected to a power source represented as the red wire, to the ground represented as the black wire and to each sensor as a wire that is used to read the sensor values. The Sound sensor is an analog sensor and as such needs to be connected to the analog input pins (left side pins) on the Arduino, the Light and Temperature sensors are digital and are connected to the digital pins (right side pins).

As it was also stated the Arduino by itself is not able to handle the Internet communication. It needed additional hardware or a Raspberry Pi like on our case.

3. Overall System

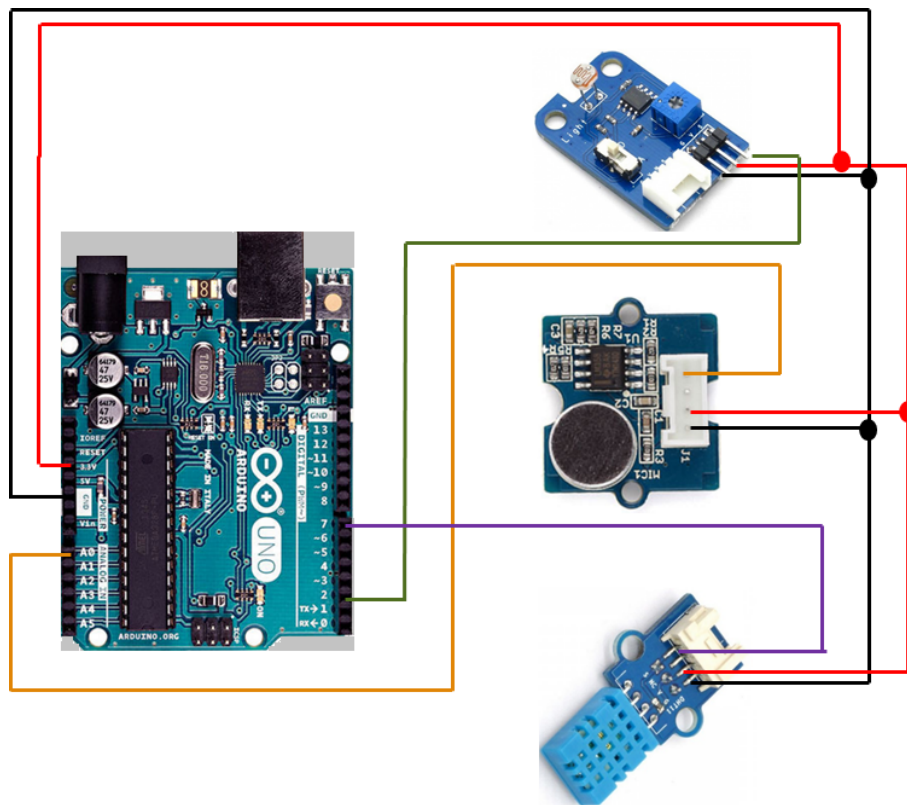


Figure 3.6: Arduino and sensors architecture

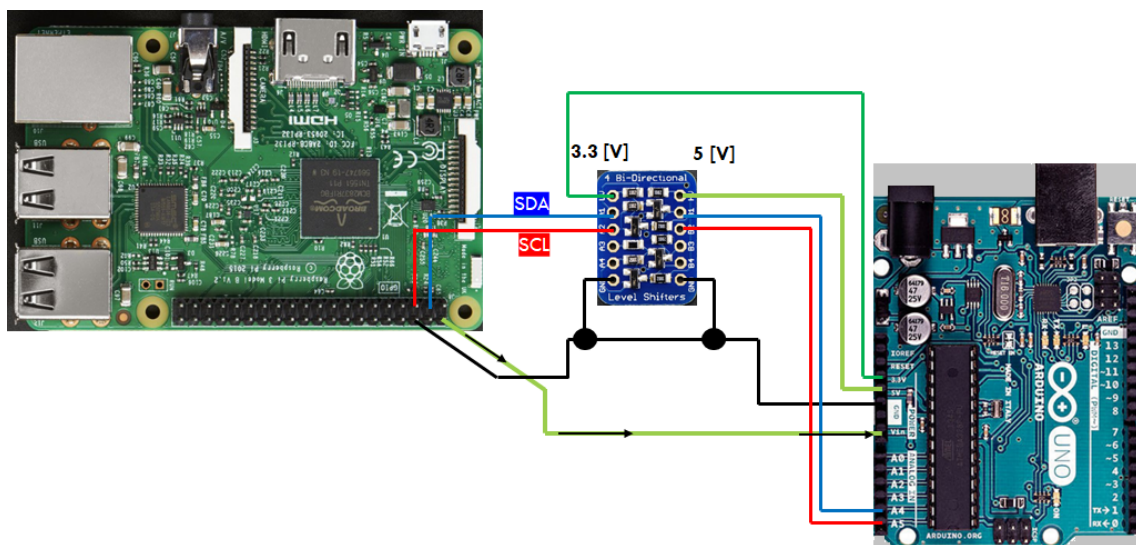


Figure 3.7: Raspberry and Arduino connection

We choose to use Raspberry Pi because it also gives more processing power and memory, what may come in hand if we decide to make our project more distributed in the future. As we can see from figure 3.7 the Arduino and the Raspberry Pi work with different voltages values and as such we need to connect them through a converter.

3.6 ISABELA application

In this section we present the overview of the mobile application, the context in which it is inserted, the description of the application and its requirements.

3.6.1 Context

As was stated before this mobile application was developed to improve students' academic outcome. With this application we want to get a better scope of the students' day-to-day routine and try to prevent behaviors that can lead to poor academic results or even academic dropout.

One of the leading causes for students to dropout of college is depression[10]. There are many factors that can lead to depression among college students, and among those there some physical factors like the lack of sleep, not doing enough exercise or social isolation [10]. If these behaviors could be monitor and prevented we can promote better academic outcomes and reduce, even if slightly, the number of students that drop out from college.

In this mobile application we also try to implement a HITLCPS architecture, where we include the students in the process by making them the first actuators to change their behaviors. Is important that the students are involved in the process because in the majority of the cases they are the ones that have to change these "bad" behaviors.

3.6.2 Description

ISABELA is a HITLCPS application that uses the Smartphone's sensors, as well as other sensors, to detect behaviors that can affect students' academic outcomes. The application is also part of a larger platform that implements an IoT

architecture that allows it to collaborate data from different sources and contextualize the environment.

This application main goal is to improve students' outcomes, and as such is meant to be used by college students, in particular the students of the University of Coimbra. This application monitor the students continuously, twenty four hours a day, to get Smartphone's sensor data and analyses the student day-to-day activities. The sensors data is retrieved and sent to the FIWARE in real-time or when a connection to the Internet is available. The data is then retrieved from the FIWARE with specific time intervals and is processed and showed to the User, so he can see by himself how he is doing. When some irregular behavior is detected, for example if the user needs to do more exercise, the application sends the user a notification showing him he has a new message form ISABELA(the ChatBot). Those messages are customized for the type of problem at hand and the user can ask following up questions, like for example "what type of activity can I do?" and get suggestions from the ChatBot like going for a run or a bicycle ride. If the user implements the behavioral changes suggested the application no longer will detect the specific problem and the control loop is closed by the user implementing this way a HITLCPS architecture.

In this application we use the students activity, sleep, social engagement and location to infer the student behavior. The flow of the application is explained in more detailed in chapter 4. As well as the processing of the information on the mobile application, what data is retrieved form the Smartphone and how we made the data acquisition.

3.6.3 Requirements

In this section we present the mobile application requirements, functional and non-functional requirements. The parameters for every requirement are exposed in table 3.1 and as can be seen the parameters used to describe the requirements are priority, description, actors, pre-conditions, event flow and expected outcome.

Table 3.1: Requirements' parameters

Parameters	Parameters' Description
Priority	<p>Must: requirement is essential to the project.</p> <p>Should: requirement is important to the project, but the system should work without it.</p> <p>Could: not implemented requirements that wont affect the implementation of the important requirements.</p> <p>Will not: not implemented requirements for the project, considered as future work.</p>
Description	Overall exposure of the requirement.
Actors	Systems that intervene with the application.
Pre-conditions	Conditions necessary to the requirement functionality
Events Flow	Description of the actions that an actor needs to fulfill in order to get the expected outcome.
Expected outcome	The expect result from the action.

In the next pages we present the application requirements. Starting for the functional requirements that are presented in more detail and in the end we make a subtle overview of the non-functional requirements.

3.6.3.1 Functional Requirements

Intro

- **Priority:** Should.
- **Description:** the application should present a intro with a reference to the University of Coimbra.
- **Actors:** the user.
- **Pre-conditions:** None.
- **Event-flow:** The user clicks the application's icon from the smartphone's menu, the application starts and shows a new screen.
- **Expected outcome:** The screen shows a new activity with the text "Made by University of Coimbra". The screen then changes to the Login screen after 5 seconds.

Login

- **Priority:** Must.

- **Description:** the application must have an authentication mechanism to ensure data security and privacy.
- **Actors:** the user.
- **Pre-conditions:** the user must be register in the system and the Smartphone needs an Internet connection.
- **Event-flow:** The user must insert a valid username and password and press the "Sign in" button.
- **Expected outcome:** If the Login is successful the application grants the user access to all functionalities. If the Login is unsuccessful the user needs to try again and make sure he meets all the pre-conditions.

Log out

- **Priority:** Must.
- **Description:** the application must have a mechanism to Log out the user.
- **Actors:** the user.
- **Pre-conditions:** the user must be Logged in.
- **Event-flow:** The user presses the Logout button from the Navigation drawer. The application changes to the Login activity.
- **Expected outcome:** If the Logout is successful the application cancels all services and the screen changes to the Login activity.

Sign Up

- **Priority:** Must.
- **Description:** the application must allow the user to register a new account in the system.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone needs an Internet Connection.
- **Event-flow:** From the Login screen the user presses the "don't have an account button" and a new screen appears. The user must then insert a valid username and password and press the Register button.

- **Expected outcome:** If the Username and password are valid the registration is successful and the app returns to the Login screen.If the registration fails an error message appears and the user can try again.

First configuration

- **Priority:** Must.
- **Description:** the application must ask the user for some information at the first login.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone needs an Internet Connection.
- **Event-flow:** If it's the first time that the user Logs in the application opens the first configuration screen. The user must then indicate the configurations and press the "Finnish" button.
- **Expected outcome:** If the configurations are saved with success the application goes to the Main screen.If the configurations cannot be saved a error message is showed and the user must try again.

Settings

- **Priority:** Must.
- **Description:** the application must have an activity that allows the user to change his configurations.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone needs an Internet connection.
- **Event-flow:** If the user presses the Settings button from the Navigation drawer. The application changes to the Settings screen. Where the User can change the configurations made in the First Login and also Login to Facebook.
- **Expected outcome:** If the configurations are saved with success the application gives a message "saved with success".If the configurations cannot be saved a error message is showed and the user must try again.

Login with Facebook

- **Priority:** Should.
- **Description:** the application should allow the user to Login to Facebook. This Login is not used as authentication is just a mean to get more information.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone needs an Internet Connection and the User must have a Facebook account.
- **Event-flow:** From the Settings Screen the User must press the "Login with Facebook" button, a pop-up appear prompting the user for his credentials to Facebook. After the Login the pop-up disappears and the text on the button changes to "Log out from Facebook".
- **Expected outcome:** If the Login is successful the pop-up notifies the user that the Login was successful and the button changes to "Log out from facebook", that button can then be used to Log out.

History screen

- **Priority:** Should.
- **Description:** the application should allow the user to browse though his information.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone needs an Internet connection.
- **Event-flow:** from the navigation drawer the user presses the History button, then he can see the information of his activity, location and sociability.
- **Expected outcome:** If the information is retrieved with success from the Fiware a chart with bars or a line chart appear indicating the values of the selected information in the picked time window.

ChatBot

- **Priority:** Should.
- **Description:** the application should have a ChatBot that can provide information to the user.
- **Actors:** the user.

- **Pre-conditions:** the Smartphone needs an Internet Connection.
- **Event-flow:** from the navigation drawer the user presses the ChatBot button, then he can see the messages that he exchanged with ChatBot. The User can then ask questions to the ChatBot.
- **Expected outcome:** If the smartphone has an Internet connection every message sent receives a response in a few seconds.

Events

- **Priority:** Should.
- **Description:** the application should have an Activity where the User could see his next events, for example classes or workouts.
- **Actors:** the user.
- **Pre-conditions:** the user must give permission to access his calendar.
- **Event-flow:** From the navigation drawer the user presses the Events button, then he can see a list of that week's events. If he presses any event he can see more detailed information about the events.
- **Expected outcome:** When the button is pressed the screen changes and the list should be populated with the events in a few seconds. If there are no events a message is showed.

Charts with information

- **Priority:** Should.
- **Description:** the application should have a main activity that shows a resume of the information.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone must have Internet connection.
- **Event-flow:** from the navigation drawer the user presses the Home button, then he can see a screen with 3 fragments that can be changed with horizontal slide.
- **Expected outcome:** if the information is retrieved with success from the Fiware the charts should refresh every few seconds.

Activity, Location and Sociability features

- **Priority:** Must.
- **Description:** the application must collect all the information needed from the sensors and process it to infer Activity, Location and Sociability.
- **Actors:** none.
- **Pre-conditions:** the user must give all the requested permissions.
- **Event-flow:** After the Login the application starts the services necessary to collect data.
- **Expected outcome:** if all the permissions are granted the application can gather the information necessary to infer Activity, Location and Sociability.

Store information

- **Priority:** Must.
- **Description:** the application must be able to store information until a Internet connection is available.
- **Actors:** none.
- **Pre-conditions:** none.
- **Event-flow:** when the information is gathered from the sensors it is stored on local database on the Smartphone, and remains there until it is sent to the FIWARE.
- **Expected outcome:** if no Internet connection is available the application stores the sensors information on a database in the Smartphone. When the information is sent to the FIWARE the information is deleted from the Smartphone.

Send information to the FiWARE

- **Priority:** Must.
- **Description:** the application must be able to communicate with the FIWARE to send all the data detected from the sensors.

- **Actors:** none.
- **Pre-conditions:** the Smartphone must have an Internet connection.
- **Event-flow:** when the User Login the application starts a service, that sends the information from the Smartphone to the FIWARE. The information is deleted from the Smartphone after the confirmation is received.
- **Expected outcome:** if the information is sent with success to the FIWARE a response code is received and the information is deleted from the local database, if the request fails the service tries to send the information again after a few seconds.

Detect behaviors that lead to bad performances

- **Priority:** Should.
- **Description:** the application should detect behaviors that can be bad to the user's academic performance.
- **Actors:** none.
- **Pre-conditions:** Internet connection.
- **Event-flow:** every time that the charts are refresh on the main screen the application runs services that detect behaviors detrimental to the user's academic performance.
- **Expected outcome:** if a harmful behavior is detected the application should change the chart colors to shades of red to indicate the problem. The ChatBot should also send a new message indicating the problem and the application should send a notification to that new message.

Give the user suggestions

- **Priority:** Should.
- **Description:** the application should give the user suggestion through new messages from the ChatBot.
- **Actors:** the user.
- **Pre-conditions:** the Smartphone must have an Internet connection.

- **Event-flow:** when a problem is detected a message indicating that problem should be received from the ChatBot. The ChatBot should also be able to answer the user's questions regarding that problem.
- **Expected outcome:** when a new message indicating a problem appears the user can ask for example "what can i do about it?" and the ChatBot should be able to answer that question with meaningful information.

3.6.3.2 Non-functional requirements

These requirements are common to the majority of the mobile application, and most of them can be solved by implementing good design patterns. The non-functional requirements are very important in the development of a mobile application, because some times a failed to implement one of this requirements is enough to make the user delete the application. The non- functional requirements for the ISABELA mobile application are:

- **Accessibility:** determines if all users can utilize the application, the application should not suffer from limitations to the number of connect users.
- **Performance:** the application should not affect the battery life too much, the memory or performance of the Smartphone.
- **Privacy:** all data collected must remain private and secure.
- **Support:** the application should be supported by the majority of the Androids devices currently on the market.
- **Usability:** the application must have a pleasing UI and also features that are important to the users.

The accessibility is a requirement of the system and not of the application. The FIWARE itself handles the accessibility. The good design of the FIWARE ensures that the system is always accessible independently of the number of connect users.

The Performance is handle by the use of background services to perform the more complex tasks, like it was explained on section 3.3. By implementing background services we are freeing the UI thread and ensuring a more responsive application.

The privacy of the data collect from the application is ensure by the authentication and by encrypting the mobile database. Furthermore the data is only store

on the mobile phone for short periods of time.

We define Support as the number of devices that support our application. Our application is supported by all devices with an Android version greater than 4.0.3. The current market share for versions lower than that is 1.8%, and as such we ensure that our application runs on 98.2% of the devices currently on the market. To ensure this compatibility is necessary to use some older API and classes that Android has deprecated.

Usability of the application is also a big requirement now-a-days, because user are becoming more demanding about the aesthetics of the applications. We took that in account when we were developing the application and tried to implement an interesting and usable UI. Some examples of that are the animations on the charts, the navigation drawer and the theme of the application.

4

Development

In this chapter we explain the development of the system. We start by explaining the Smartphone development followed by the Smartwatch application development.

4.1 Smartphone application

In this section we explain in detail the development of the mobile application. We can divide the development of the application on 4 sections the acquisition, the communication, the storage and the processing and in the end we present the application flow with screenshots of the application.

4.1.1 Data acquisition

The acquisition of data by the Smartphone is either made by direct acquisition of the sensor values, or by context acquisition, that is we use on or more sensors to acquire a virtual sensor value (for example class attendance) or we get context information from the Smartphone's SMS and calls.

4.1.1.1 Physical sensors

The physical sensors are sensors that are implemented in the Smartphone by hardware. To obtain the values of the Smartphone's sensor the Android OS provides an API, the android.hardware API [45]. This API has classes dedicated to manage the sensor acquisition, namely the SensorManager class [46]. This class allow us to retrieve the values of all the sensors of the Smartphone. The steps to do this acquisition are:

1. Get the sensor manager instance from the application context.
2. Get the sensor from the sensor manger, for example the light sensor.
3. Create a listener that implements the code that needs to run when that sensor value changes.
4. Register the listener to that sensor with the sensor manager instance.

These steps need to be replicated to every sensor we want to use, except for step 3 where the listener can be the same. For 2 different sensors we only need to implement a condition that verifies what sensor triggered the event and implement a different code on the listener. As it was stated before these listeners are triggered by events, namely a change on a sensor value. This event returns a object from the `SensorEvent` class[47]. The `SensorEvent` objects have 4 attributes that can be retrieved. These attributes are: the accuracy, the sensor type, the timestamp, and the values from the sensor. Using these 4 attributes we can gather a lot of information, for example the sensor type attribute lets us see what sensor triggered the event and the accuracy allow us to disregard events that may not really happen. But the values attribute is the more relevant attribute, because the value that the sensor returns is in that array. Generally the value of the sensor is returned by the value on the index zero of the array. But there are some sensors that return more than one value like for example the Accelerometer.

In our case we implement the `SensorEventListener` interface[48] on our `MainService`. As can be seen from figure 3.3 that service is responsible for the data acquisition. In this service we created a variable for every sensor we read. Those variables are then updated with the latest value of the sensor whenever an event triggers the listener. This service runs as well some Timers that are responsible for trigging the storage of the data. We set the timers to store the data every 5 seconds, because we wanted to have the best resolution possible when we pos-process the data. The architecture described can be seen on figure 4.1.

The physical sensors that we take values from in this project are indicated in the list below:

- proximity sensor.
- light sensor.

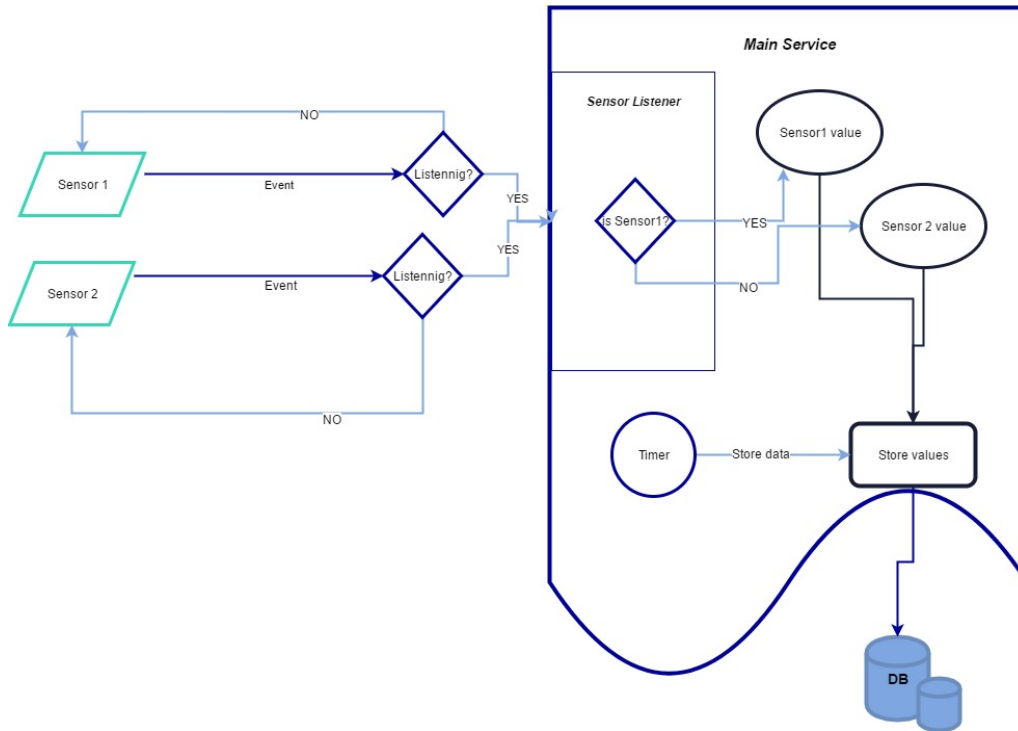


Figure 4.1: MainService flow diagram

Apart from these 2 values we also measure other parameters that rely on physical sensors, but are not taken directly from them. We measure the sound amplitude and to measure it we need a different approach that uses Android MediaRecorder API [49]. That API records the microphone response and stores it on a file or in a buffer. In our case as we only need the amplitude in real time we opted for the buffer approach. This approach also serves as a privacy measure since we are not storing any microphone data on the phone. The flow of the data from this "sensor" is the same as the other physical sensors. We store the value of the amplitude on a variable and that value is store when a timer ends.

Another parameter that we take from physical sensor is the Activity. To measure the Activity we use the Google Activity Recognition API [50] that is part of the Google location API. Although they don't specify what sensors the API uses to infer the Activity, they only say that it uses short busts of low power sensors data[50]. We firmly believe that this API relies primarily on the Accelerometer data. To implement this API we need to implement a listener and we set a minimum time period in which we get updates from the API, in our case 1 second. This Listener is implemented on another service that extends the IntentService class[44],and as such run on its own thread. This service communicates with the MainService through Local Broadcasts[51] (the use of Local broadcasts is explained in more detail in the

communication section of this chapter). When the `MainService` receives a broadcast from the Activity Recognition service it stores the value of the activity on a `String`. The value of that variable is then stored when a timer ends. The API also returns the value of confidence that it has on the result, and we can use that confidence to post-process the data.

The possible results from the Activity Recognition API are:

- In vehicle
- On Bicycle
- On foot
- Running
- Walking
- Still
- Tilting
- Unknown

In our project we consider that there is no need to differentiate the types of exercise and as such the `On Bicycle`, `On foot` and `Running` result were changed to `Exercise`.

We also took the Location based on Global Positioning System (GPS) from the Smartphone. This also required a different approach from the one explained before. We implemented the Fused Location Provider API[52]. This API was developed by Google specially to the Android OS. The Fused Location API is designed to improve the precision and battery life, but none the less we have to take some measures to improve battery life, like a bigger interval between data acquisition.

The method we used to get the precise Location consists of creating a `Location Listener`[53] that implements a client to the Fused Location API, and receives location updates from time to time. The rest of the architecture is equal to all the other physical sensor. The value of a variable on the `MainService` is updated by sending a Local Broadcast and that variable's value is stored on the Database when a timer ends.

It should also be noted that to obtain this precise Location in an accurate way we need the user permission to obtain the course location and the fine location. A

list of all the permissions used on the application and the reason they are used are given in the appendix B.

4.1.1.2 Context acquisition

Some of the data we collect from the Smartphone does not depend on hardware and as such we call it context data, because it gives us a better insight of the student's life. In this section we explore this kind of data and how we process its acquisition. The data we obtain by context is:

- Calendar information.
- SMS information.
- Alarms information.
- Calls information.
- Discrete Location information.
- Facebook information.

By calendar information we are referring to the student's academical calendar, classes, exams, deadlines and events that are scheduled. To obtain this information we need that the student register the Inforestudante[54] calendar on the Smartphone. To do so the Inforestudante already provides the student's with a link that they can use to export their calendars to their Google account. We also need the user permission to access the calendar information, as it is considered sensitive information by the Android OS.

To detect the classes, exams or deadlines we created a service that checks all the events on the Inforestudante's calendar. The service check the description of the event, if it's a class, an exam or other, and check if they are set for the current time. This service then updates a variable on the MainService through a Local Broadcast and the MainService takes care of storing this info. This service unlike the other it's started by a timer, with time intervals of 30 seconds.

The SMS detection work the same way as the calendar detection, we created a service that goes through the SMS. Although this service only checks the SMS in the last 5 seconds, because we want to obtain a history of the SMS sent and received. For privacy reasons we don't see the content of the SMS. We only see the timestamp to make sure the SMS was sent or received on the last 5 seconds. We

check the type of SMS to separate the received SMS from the sent ones and we also see the destination or source of the SMS.

We use the SMS destination and source to check the number of different people that the user contacts. To keep the user privacy we encrypt the number by making it a unique string without meaning. These strings are never decrypt and we only used them to obtain the number of different person the user contacts. This service works in the same way as the calendar check service, it is triggered by a timer and when it ends it sends the result to the MainService through a LocalBroadcast.

The Alarms information is also obtained through a service that is triggered periodically, in time intervals of 30 seconds. This service checks if there is any Alarm set and how much time before the alarm goes off. It then returns these values to the MainService that stores them.

The Calls information work in a different way: this service is not based on a IntentService but on a BroadcastReceiver[55]. The broadcast Receiver class work in a different way, the receiver must be declared on the android manifest, and we must create a filter for the intent we want to detect. Tn this case we are setting a receiver for a intent that is sent by the OS whenever the phone makes or receives a call. In this service we check the type of call, made, received or missed, the duration of the call and the destination or source (this one is encrypt in the same way as in the SMS detection). These values are then sent to the MainService through Local Broadcast and process in the same way as the other services.

The discrete Location information is obtained through the Wifi networks available. To be more concrete we consider 3 possible locations: Home, University and Other. We scan the available networks through a Wifi scan that returns a list with the Service Set IDentifier (SSID) and the strength of the signal for each network. We then check if any network corresponds to the the network of the University (Eduroam network) or to the User's House network (the user picks this network from the list of configured networks in the phone, when he does the first Login). If none of the networks correspond to either the House's or the University's networks we consider that the user is on Other location.

On Our mobile application we also interact with the Facebook, to get information about the user social network. To do so we are working on a API that is being developed by Helder Senna, a Master's student of Informatics Engineering. Presently the application only uses the number of likes in the last hour, and to get this information we created a service that implements the Facebook's Graph API[56]. The

Graph API is an API developed by Facebook for developer and uses HTTP/REST requests to get this information, and because of that the service also extends the IntentService class, it needs to be a background service. When the request gets a response this service sends the processed information through a broadcast intent to the MainService. Since this service only runs once per hour, we stored the value on a different entity of the DB (where the other Facebook values will be store in the future as well), and only update the value to the Fiware once per hour as well.

4.1.2 Communication

The application has 3 external communication modules: Fiware's communication module, the ChatBot communication module and the Facebook communication module. Apart from these 3 external communication modules we also have the internal communication between activities and services. In this section we start by explaining the internal communication and then we present the 3 external communication modules.

4.1.2.1 Internal Communication

When we refer to internal communication in this document we are referring to communication between the elements (activities and services) of the mobile application.

As it was already stated in other section of this document, the communication between Activities and Services is made with the the use of Local Broadcasts. To do so we use the LocalBroadcastManager class[51], that as the name indicates is a class created to manage the local communications. This class allow us to send broadcasts through intents[57]. These broadcasts can then be received by Broadcast Receivers[55] with the specific intent filter[58].

The Intent class, like the name says, is taken by the Android OS as an intent to do something. Intents normally are used to swich between Activities or to start services, but can also be used to communicate with Local Broadcasts. Intents work like bundles of information. We can store values or even objects in them and used them to send this information. To use a intent as a means of communication we need first to give them a component name, that will work as an identifier to this specific type of message. Then we use the LocalBroadcastManager class to send this intent to any service or activity of the application's process. And to

receive this information, on the destination, we have to register a Broadcast Receiver. Once again the LocalBroadcastManager manages this registration and besides the Broadcast receiver we also need to create an instance of the IntentFilter class. This IntentFilter accepts one argument that has to be a String equal to the component name given to the Intent that was sent. This way we are filtering all the intent until we find the one we want. Then we can define the action we want the system to do on the Broadcast Receiver. A service can have multiple BroadcastReceivers instances registered to different IntentFilter like for example the MainService that receives information form multiple services. An diagram with an example of the usage of this implementation is given on figure 4.2.

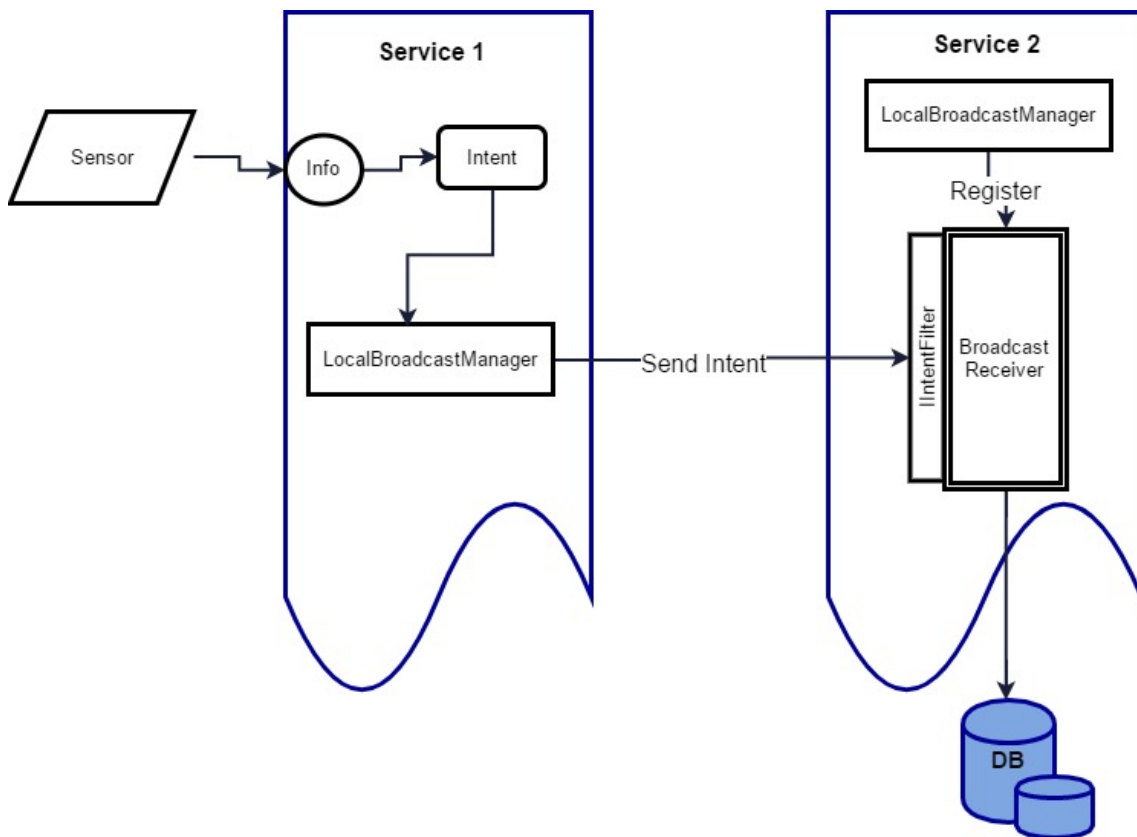


Figure 4.2: Example of the communication between 2 distinct services using Local broadcast Manager and intents.

With the implementation we described above we can create "channels" of communication between the services or between a Activity and the services. These "channels" are our main means of communication, for real time data sharing. Although we also use the DB as a mean of sharing information between the services, that is, we store the information in one service and retrieve it on other service by querying the DB. This topic is explored in the next section.

4.1.2.2 FIWARE communication

The FIWARE has multiple modules as it has been stated before, but we only communicate with 4 of them: the ORION, the IDAS, the COMET and the IDM. All of these communications are made using HTTP/REST requests, and to implement them we use services that extend the IntentService class, because all the requests need to be asynchronous. The communication with the ORION is made with 3 objectives: create entities, update or add attributes values and retrieve attribute values.

If there isn't any Access Token, object return from the Oauth 2.0 authentication, in the application the application asks the user for the Log in. When the user Log in the application makes an POST request to the IDM, this post request creates a new Access Token for the user. If the user chooses to create a new account the application also stats a service that implements a POST request to create a new user account. After a successful login, the FIWARE returns a Access Token that is used in the requests to the other modules. Those modules then validate the Access Token with the IDM before returning a response.

The first interaction that the application does with the ORION, is when the user Logs in. The application needs to do a GET request (an REST request to retrieve information) to checks if the entity with the user IDentification (ID) already exists. If not the application does an POST request (an REST request to create an entity) to create a new entity for the user.

Also if there isn't a entity with the user ID on the ORION the application considers this as the user first Login and ask for the first configurations. In these first configurations the user needs to set the ID of the device (Kit with Rapsberry and Arduino) that is on his room. To do this configuration the user chooses between a drop-down list options. To get theses devices ID the user application needs to make a GET Request to the IDAS (devices management module) to get all the available devices on the system. Then the application filters the devices that aren't already associated to some entity. After the user finishes these configurations the application needs to link the the user's room entity to the selected device through a PUT Request (HTTP request to replace an entity with the sent one). This request replaces all the entity with a new one that is sent as the payload of the request. And the application also sends the configuration information to the user's entity through a PATCH Request (HTTP Post Request where the payload id and type are not included). Unlikely the PUT request this request only updates values for the

sent attributes or creates a new attribute if there is no attribute with the specific attribute name. This procedure repeats itself for the Settings activity where the user can change this configurations.

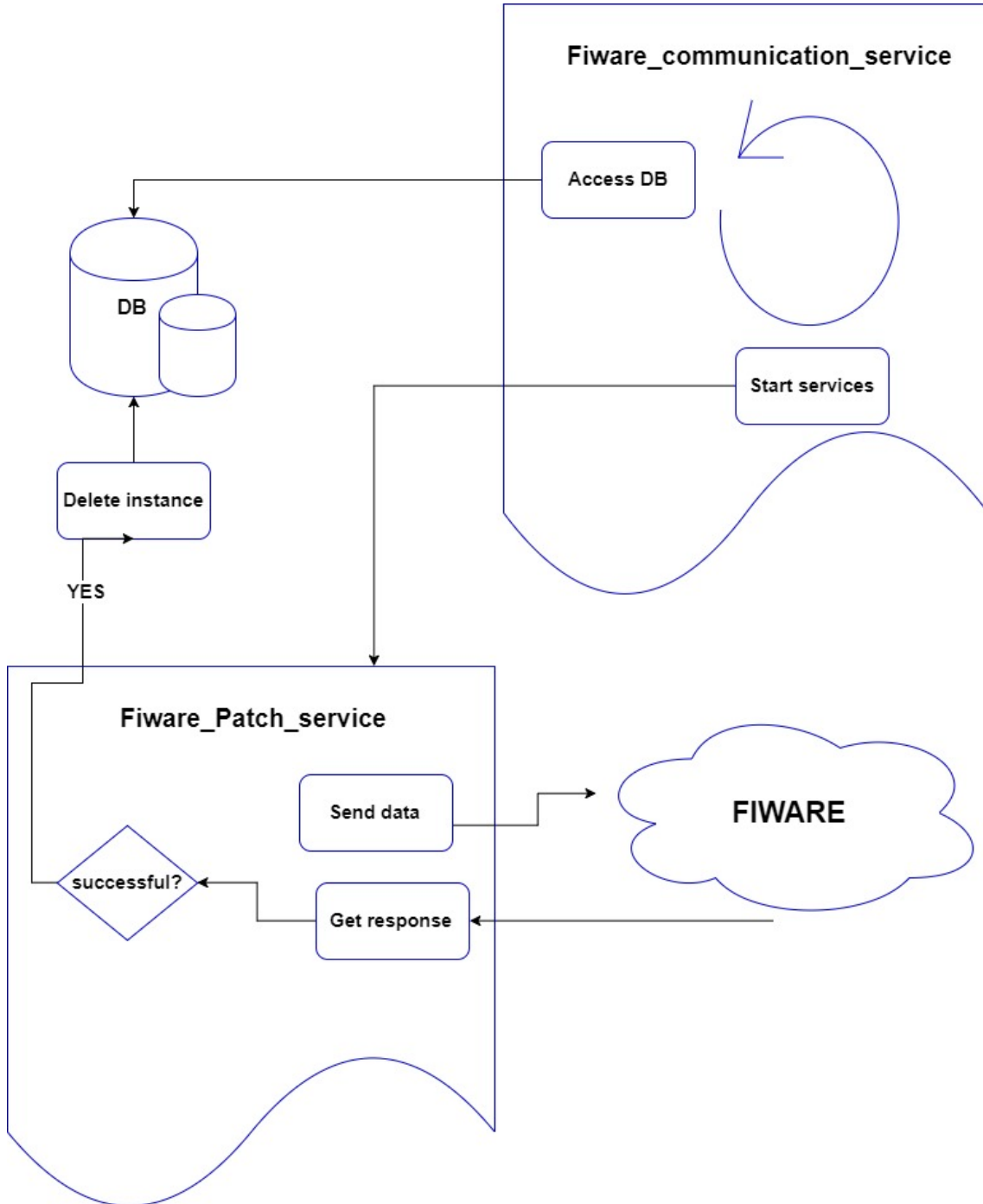


Figure 4.3: diagram of the operation of the Fiware_communication_service and Fiware_patch_service services

After these first steps (Login/Registration and Configuration) the application starts the Fiware_Communication_Service service. Like the name indicates this ser-

vice is responsible for coordinating the communication with the FIWARE. This service executes its own thread and runs on the background so as not to affect the UI thread. This service has 3 main functions: read the stored data from the database, iterate the returned values one by one and start the services that execute the PATCH Request services (as explained before this request updates the values of the attributes of a entity). To make sure no data is lost the entity that represents the information sent on the database is only deleted when the PATCH service receives the confirmation code from the FIWARE. This service sends the data of three entities: the data from the Smartphone that corresponds to the user, the data for the sleep classification and the data from the social sensors (Facebook data). A diagram that represents this procedure can be seen in the figure 4.3.

The other module that the application communicates with is the COMET. The COMET implements a REST API that allow us to make GET requests (make queries for information). These requests allows us to configure what information we want, in which time interval and how we want it aggregated. The communication with this module is made in several parts of the application, but the majority of these requests are made to get the display information for the main screen. The application also request information from the COMET to make the historic screen and for the sleep classification. These services that communicate with FIWARE, unlike the previous ones, are started by the UI thread and return the values by local broadcasts.

There are some other services that communicate with the FIWARE, like the WearListener service and the ChatBot service. But those services and communications implement the same logic as the one described above. These communications are also explained in the next subsections.

4.1.2.3 Wear communication

For the communication between the Smartphone and the Smartwatch we implement the Android Wear Message API[59], an API developed by Google to facilitate the communication between these devices. This API allows us to set a listener for incoming messages on the Smartphone by implementing the WearableListenerService interface on a class. For this class to work we also have to define it as a Service on the Android Manifest and define the Intent-Filters that we want to detect (this filters are also define by the API).

Also on the Smartphone we have to define a eXtensible Markup Language

(XML) file with the Capabilities of that Smartphone. As the name indicates Capabilities serves as an indication of what a device can or can't do, and this is used as way to find the available devices that have the desire capabilities. For example we could define a capability for being able of sending SMS. In this specific case a Smartphone would have this capability but a Tablet wouldn't. It is easy to see how this implementation can help us to develop applications that don't waste resources. But in our case we don't need any specific capability and as such we define a simple capability just for the implementation.

As we stated before we need to define capabilities for a device so we can find them on the Smartwatch, and the Smartwatch does this through the Capabilities API[60]. This API lets us find the Nodes[61], the devices on the Android wear network. These nodes are represented by a opaque singular ID, and we can also see if the Node is nearby (that is if it is available through the Bluetooth connection or through the cloud API).

These APIs allow us to share information between the Smartwatch and the Smartphone, and also delegate some tasks that require more computation power to the Smartphone. But, in our case, we implement these APIs because it is a recommendation by Google that we use the Message API when trying to send information to the cloud. This recommendation is due to the fact that although Smartwatch can connect to networks over Wifi when it is away from the Smartphone (to stay connected to the Smartphone through a cloud system), this connection is not possible when the Smartphone is nearby once that the connection between the 2 is made by Bluetooth and the Wifi is turned off to save battery. So the only way to ensure reliability is by using the Message API.

On a overhaul we implement a listener, that waits for new messages from the smartwatch. We make a search for available Nodes on the Smartwath and send the information for the closest one. The information is received on the Listener and processed. On the this Listener the information is stored on a DB, and this service also tries to send the information to the FIWARE. The wear information is sent to the FIWARE through a PATCH request. This request updates the values if they already exist or create these attributes on the student's entity in case they don't exist.

4.1.2.4 ChatBot communication

The ISABELA's ChatBot was created using the API.ai[29] as it was stated before. To obtain responses from this platform we need to implement a Android Software Development Kit (SDK) that the API.ai provides to developers. This SDK implements an synchronous REST API and as such it needs to be called on a background thread.

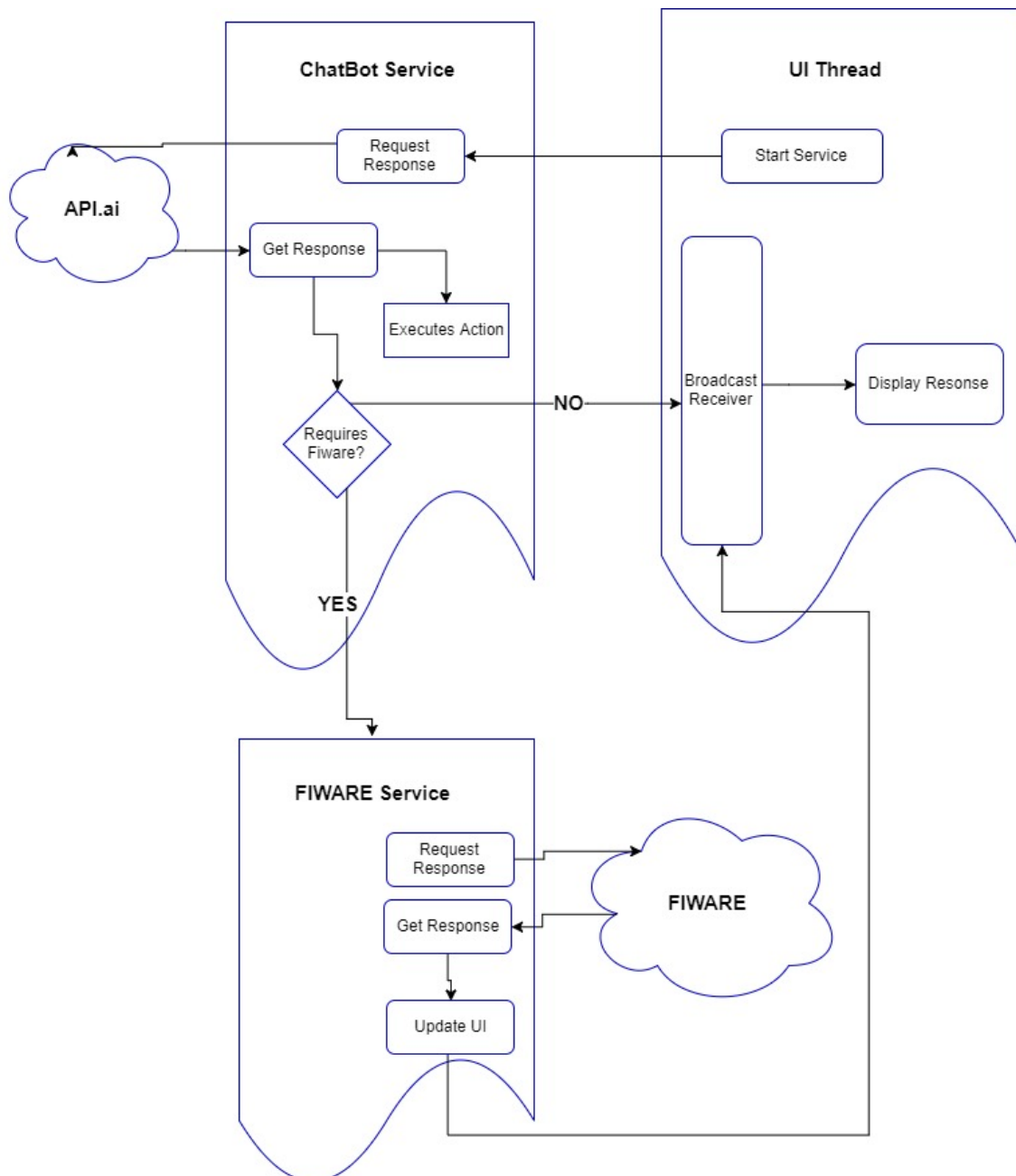


Figure 4.4: Flow of events when a user sends a message to the ChatBot

We created a service that uses this SDK, handles the responses and also inter-

acts with the FIWARE when needed. Since this service doesn't need to be always running and the user's messages have temporal context, that is the user wants the responses in the same order as the messages he sent, the service we created extends the IntentService class. As it was stated before many times this class runs on its own thread and creates a queue for the instances that are called, that is, if the user sends two messages the second message is going to wait for the service to process the first one. This help us maintain the temporal context we previously refer.

As we can see from figure 4.4 every message that the user chooses to send starts a new instance of the ChatBot service,.The SDK then processes the message and waits for a response. When a response arrives the service processes it and executes the intended action. If application needs additional information, it starts one or more service that request that information from the FIWARE. The actions that are currently implemented on the ChatBot are:

- Show alarm messages (Bad behavior notifications).
- Follow-up response to an alarm.
- Request info from the environmental sensors.
- Set an Alarm Clock.

The first 2 actions where the intended actions when the ChatBot was proposed on this project, because we wanted to use a more user friendly approach to close the loop (give user information about is behavior). The first action (show alarm messages) doesn't require user interaction. The application sends a predefined Message to get the ChatBot intended alarm. Then the user can ask questions about what we can do about that particular behavior (follow-up questions about the alarm). For example if an alarm is raised about the user exercise levels, a possible conversation between the user and the ChatBot can be:

ChatBot: " Hey we should do more exercise."

User: " Any suggestions?"

ChatBot: "Let's go for a run."

The third action is used to obtain info from the FIWARE. That is information from the IoT kit that the student has at home or from the devices at the University. Like it is represented on figure 4.4 the ChatBot Service verifies if the response, taken from Api.ai, needs FIWARE information and, if so, initiates a new service that deals with the FIWARE communication and also delegates the UI update to that service.

The last action is just an example of functionalities that the ChatBot can have in the future that will make it more useful to the user. Right now it only has this functionality to set alarms, but as future work it could be able to suggest user spots to socialize, set events on the user agenda or even call someone for the user (More about future implementations on chapter 6). Although this is a simple implementation, it already allows the ChatBot to create new follow-up answers to the alarm, more specifically the alarm that it's raised when the user needs to sleep more. For example the dialogue can be:

ChatBot: "You should sleep more if you want to be healthier."

User: "Ok, set up my alarm for tomorrow at 8."

Chatbot: "I set your alarm to 08:00:00 of 2017-06-29."

We believe that the ChatBot should be intuitive and if it tells some information to the user, it also should be able to give him the details and perform simple action related to that information. Other example of a simple implementation is for example: if the ChatBot is able to warn the user that he is about to be late to a class, the user should also be able to ask where the class takes place. This way he doesn't have to switch between applications or search by information.

4.1.3 Storage

First it is important to note that we did not create the database on the Smartphone with the intention of having a set of structured data. Our only intention was to have a mean to store data when there was no connection available to the FIWARE. And as such the database on the Smartphone is used only to prevent data loss. Nevertheless in this section we present the structure of the entities we used in this database as well as the means we used to create and access the database.

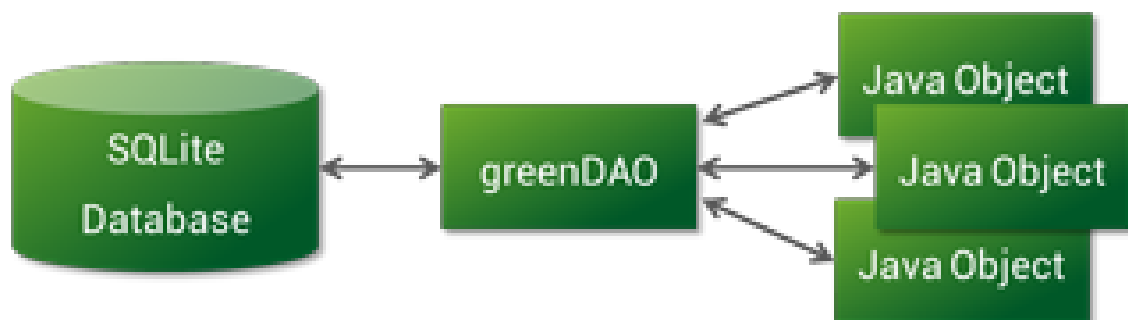


Figure 4.5: greenDAO working flow[4]

4. Development

We used an Android ORM library, greenDAO[4], to create and access the database. This means we also used the code first approach. The code first approach means we write the code first, that is, classes that represent the database entities. Then we create the database with those classes. And as we can see from figure 4.5 the greenDAO library works as an intermediary between the DB and those classes.

As was stated before this database works as safe storage to the acquire data and as such our database entities represent that data. We tried to simplify the DB architecture, and as such there are no relations between the data. The entities that we used can be seen on figure 4.6.

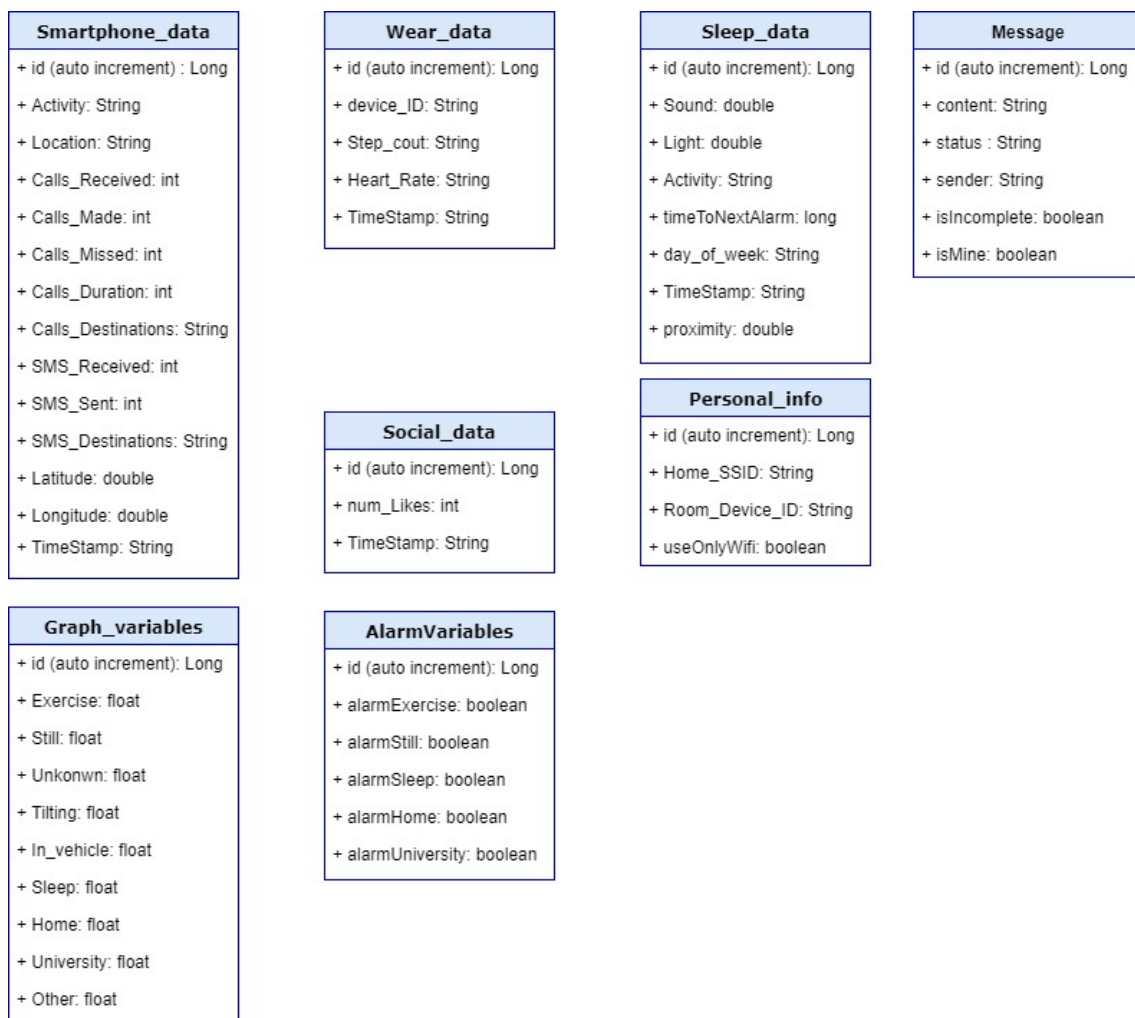


Figure 4.6: Database entities/ORM classes

As can also be seen from figure 4.6 we also used the DB to save some configurations like the values of the graphs from the Home screen (Graph_variables), the Personal configurations for the application (Personal_info) and the status for the alarms from the user behavior (AlarmVariables). On these three entities we only

save a instance at a time with the ID 0. When we want to change any attribute we replace the existing entity for a new one. There was other approach that could be used, but since we already were using the ORM architecture this was the easiest way for us.

From figure 4.6 we can also see that the Social_data only has two attributes. It may look unnecessary to use a DB to hold such a small entity, but as we stated before this module is under development by another student and we wanted to leave the architecture ready for implementation of this Social information API.

The Smartphone_data, Sleep_data and Wear_data are the entities that hold the sensors information. The Smartphone_data holds the majority of the information gather in the Smartphone. The Sleep_data entity keeps the data that is used for the sleep classification. And the Wear_data stores the information that is received from the Smartwatch. All these entities have a Time stamp attribute, that represents the time of acquisition of the data (in the Wear_data the Time stamp is also send from the SmartWatch). This attribute is important to maintain time series when the data can't be sent immediately to the FIWARE.

The Message entity stores the messages that are sent from the user to the ChatBot, as well as the one the user receives. This information is only store in the Smartphone for privacy proposes, and is only used to display a chat history to the user.

4.1.4 Processing

As we stated before we obtain 3 main context information from our data the Activity, the Location and the Sociability. But in this section we divide the data processing in 4 sections: Activity, Sleep data, Location and Sociability. Although we consider Sleeping as an activity the sleeping data classification is processed in an different way from the activity classification and as such deserves a more detailed explanation. We also describe the way the alarms are performed.

4.1.4.1 Activity

As it was already stated on section 4.1.1.1 we used the Google Activity Recognition API[50] to infer the user Activity. Although this facilitate our work, it also limits our processing options. But we still can make some operations over the results of the API to make the classification more suitable to our needs.

This API returns an list of possible activities and the confidence of each one of them (the reliability of that result). We consider the most provable activity (activity with the most confidence) as the current user's activity. If none of the returned activities has a confidence superior to 0.3 (the confidence goes from 0 to 1 and indicates the percentage of confidence) we consider the user activity as "Unknown", that is we couldn't classify it. We can also verify all the activities on the list returned from the classification. For example if the last activity was "In vehicle" and on the new classification we receive a list with "Exercise" on the first position of the classification and "In vehicle" on the second position. If the confidence on the first classification is not to high we can consider the second classification as more accurate because of the previous classification.

We also aggregate some of the classifications that refer to user doing physical activities, that is we discriminate the type of physical activity and replace it by "Exercise". The activities that are replaced by "Exercise" are the "On Bicycle", "On foot" and "Running".

4.1.4.2 Location

We used the available Wifi networks to define the User discrete location, if it is at Home, University or Other. As was referred before on the main service we start a new service that scan the available Wifi networks. This scan returns a list with the networks information and with the strength of each signal.

The Network information, returned by the Wifi scan, includes the SSID and the Media Access Control (MAC) Address (the Access point singular identifier). The SSID allow us to identify the network and if we know in which location one specific network is available we can pin point the user Location. However there are networks that can be available in more than one location. The Eduroam network, for example is available in all the University facilities, and this makes our task more difficult. In this case study we only wanted to determine if the student was at the university but in future implementations it can also be important to verify if he is in the right class, and this can be achieved by mapping the MAC addresses of every Access Point (AP) in the University. If in the future we map the AP and the strength of the signal in every class room we can obtain a more precise location and, of course, more context.

We also used the location to check class attendance. If the user has a class on the schedule and he is at the University, then he is attending the class. This however

is not 100% accurate since some student's may prefer to stay at the bar or even frequent the library instead of attending the class. By mapping the Access points of the University we would eliminate this uncertainty. Consequently we consider this task as an important future work and we explore this possibility on chapter 6.

4.1.4.3 Sociability

The StudentLife study[3], as we previously explore on the stat-of-the-art, also took sociability in consideration to evaluate the student's lifestyle. They did this by recording the student's Smartphone microphone audio and classifying it as conversation or not. However we consider this approach a bit intrusive to the students and with a lot of privacy issues. And as such we searched for other approaches to measure the student's sociability. We found some other works and between them a Master thesis developed in the University of Utrecht[62]: a student from the Master in Business Informatics created a model that calculated the sociability. In this model he used the SMS, calls, Social Networks activity and some other "non-invasive" factors to infer the sociability with a mobile application. We aimed to developed a similar model to infer the sociability.

However in this first approach we lacked the knowledge, to obtain the ground truth about the student's sociability. And as such we had several meetings with a investigation group from the Faculty of Psychology and Education sciences, from the University of Coimbra, lead by Professor Dr. Carlos Barreira. However, these partnerships have a long adaptation period and as such we started to develop a more simple model. This partnership and future work related with it is explore in more detail in the Chapter 6.

The model we created is based on a simple formula, which makes a sum with weights for each defined factors. We define the 5 factors to calculate the sociability, the SMS's frequency, the Calls' frequency, the SMS's ratio, the Calls' ratio and the diversity. These factors are defined by the formulas 4.1,4.2,4.3,4.2 and 4.5 respectively.

$$\text{SMS frequency}(SMS_{freq}) = (SMS_{received} + SMS_{sent}) / \text{(total time of acquisition in min)} \quad (4.1)$$

$$\text{Calls frequency}(Calls_{freq}) = (Calls_{received} + Calls_{made}) / \text{(total time of acquisition in min)} \quad (4.2)$$

$$\text{SMS Ratio}(SMS_{ratio}) = SMS_{received} / SMS_{sent} \quad (4.3)$$

$$\text{Calls Ratio}(Calls_{ratio}) = Calls_{received} / Calls_{made} \quad (4.4)$$

$$\text{Diversity} = \text{(number of different people contacted)} / \text{(total number of people contacted)} \quad (4.5)$$

Taking these features in consideration, we needed to remove the units from the SMS_{freq} and from the $Calls_{freq}$ and make them dimensionless. In order to do that we search for statistics for the average number of SMS and Calls exchange by a regular person each day.

We found a study[63] from 2014 that reported that the average of SMS exchange by a person between the ages of 18 and 24 where 128 SMS per day. We found that number a bit to high and as such we consider 90 SMS a day (3/4 of the number on that study). This decreasing is do to the fact that we believe that using the phone to much is not healthy. And also because now-a-days there a lot of other alternatives to the SMS and we need to take in account that if the users use a different type of communication technology the number of SMS will decrease.

We also found some information dating 2010[64], that indicates that the number of average calls per day is 5. Although the study is a bit old is a good starting point, and as we stated before this is just the first approach to the model.

We had to change the values for the average number of SMS and Calls per day to SMS and Calls per minutes to use them on the equation.

It is important to refer that we don't consider the overuse of the Smartphone functionalities as a bad habit but we don't endorse it. That is, we only consider the maximum of the weight given to each feature. So in equation 4.6 if one of the parcels of the sum is bigger than it is weight we consider the weight instead. This way we ensure that the score is between 0 and 100 and that we don't favor overusing

behaviors.

$$\begin{aligned} \text{Sociability Score} = & \frac{SMS_{freq} * Weight}{Average SMS_{min}} + \frac{Calls_{freq} * Weight}{Average Calls_{min}} + SMS_{Ratio} * Weight \\ & + Calls_{Ratio} * Weight + Diversity * Weight \end{aligned} \quad (4.6)$$

The last step of the model was to choose the weights for each one of the features. As we stated before we only aimed to make the first approach to the final model and as such we choose the values of the weight by our own perception of what is more important in the sociability point of view. We prioritized the values for the SMS_{freq} , $Calls_{freq}$ and the diversity instead of the ratios. The weights we choose are described in table 4.1. By replacing the values from table 4.1 in equation 4.6 we obtain the final equation to the Sociability score (equation 4.7).

Table 4.1: Weight for the Sociability score (equation 4.6)

SMS_{freq}	$Calls_{freq}$	SMS_{ratio}	$Calls_{ratio}$	Diversity
30	30	10	10	20

$$\begin{aligned} \text{Sociability Score} = & \frac{SMS_{freq} * 30}{Average SMS_{min}} + \frac{Calls_{freq} * 30}{Average Calls_{min}} + SMS_{Ratio} * 10 \\ & + Calls_{Ratio} * 10 + Diversity * 20 \end{aligned} \quad (4.7)$$

Once again it is important to refer that this is just the first approach to the final model, and that we are considering to use neural networks and more features once we are able to obtain a ground truth.

4.1.4.4 Sleep detection

There are several works about sleep detection, through classification, over the years like Chen Z et al[65] and Min J et al[66]. Although these studies have the same goal, applying machine learning to perform sleep detection, they have different approaches. The first one focus on non obstructive sleep detection, that is they

don't require any device on the user's body, while the second study does. The use of complementary devices, with accelerometers attached to the user's body, can be more accurate to perform sleep detection. But this kind of device requires that the user changes his routines and performs bothersome tasks every night. So, in our case, and as we were already focusing our efforts on detecting data through the Smartphone, we decided to make an approach that resembles the one done by Chen Z et al[65].

In the study done by Chen Z et al[65], they used 6 features to perform sleep detection and they were:

- Light feature (amount of light in the environment) .
- Phone lock.
- Phone recharging.
- Phone off.
- Stationary feature (amount of time the phone is still).
- Silence feature (amount of sound/noise in the environment).

All of these features can be easily explained by the factors that normally are attached to the sleeping activity. Commonly people tend to sleep in quiet environments and with less light intensity. The other 3 features focus on the Smartphone usage patterns but can also correlate with sleep patterns, since commonly Smartphone's users tend to charge the phone at night or even turn it off to don't be disturbed. The phone lock and the stationary features are self-explanatory since if the user is sleeping he can not be using the phone and the phone remains still. Is important to refer that these feature individually are weak indicators. In this study by Chen Z et al[65] they got good results with ± 42 minutes accuracy to the sleep time [65].

In our study we used 7 features, some are equal to the study described above but some are different. The features that we choose are:

- Light feature (amount of light in the environment) .
- Phone lock.
- Day of the week.
- Time to the next alarm.
- Pressure sensor value.

- Activity.
- Sound amplitude taken from the microphone.

The light, sound amplitude and the phone lock features were chosen by the same reasons that were described above. The Activity feature is equivalent to the Stationary feature. We are taking the mobile phone motion into account to detect if the Smartphone is being moved or not.

The pressure feature is included in our classifier because of the time that the phone remains in the user's pocket, that is in the pocket the Smartphone as no light and the sound amplitude of the environment is decreased even on loud environments. Because of that the values taken from the Smartphone while in the pocket might look like values from a sleep pattern. Since most people carry their Smartphones on their pockets, for long periods of time, without this feature we probably would have a lot of misclassifications.

The phone now-a-days replaces our alarm clock, and as such people set the alarm every day to the hour they intend to wake up and except some extraordinary cases they do wake up when the alarm goes of. This makes the time to the next alarm a good feature with a strong correlation to our sleep duration. When the alarm is set in the phone we check the time to which the alarm is set and subtract the current time to it (in epochs). If no alarm is set we define the value -1 to this feature (is a value that can't appear in this feature when the alarm is set). The classifier then can be adjusted to these values.

We also used the week day because we believe that the user's patterns are highly influenced by this feature. For example normally people (not just students) tend to sleep more in the week-ends than in the week days. And with this feature the classifier can become more tailored to each user, that is it can adjust better to the different user's sleep patterns. For example if a student has no classes on Wednesday he may wake up later that day or for example some user might not even set an alarm clock on the week-end. These are just some examples of how this feature can be useful to get a different scope of the user or even better adjust the classifier to the other features.

Some of these features, day of the week and activity, were given by strings but luckily they were already a discrete number of strings and we could easily normalize those features to numbers. The phone lock was also a string, that represented a boolean. But it was converted to 0 or 1, 0 when the phone is not locked and 1 when the phone is. The other features were already numeric values and had only to be

normalized.

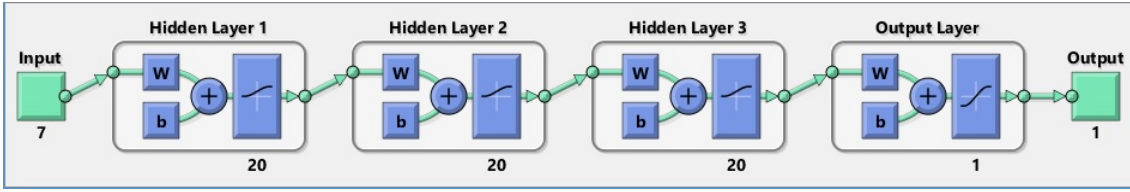


Figure 4.7: Sleep Detection neural network structure

As we stated before we are using Encog[26], an Java machine learning framework, to perform the sleep classification. We are using a Basic Neural Network[67] with 1 input layer, 3 layers of neurons (each with 20 neurons) and 1 layer of outputs as it can be seen from figure 4.7. Every Hidden layer has a Logarithmic activation function and the same happens to the output layer.

Table 4.2: Sleep classification network performance

Sensibility	Specificity	Accuracy
87.99	94.06	92.71

$$\text{Sensibility} = \text{TruePositives} / (\text{TruePositives} + \text{FalseNegatives}) \quad (4.8)$$

$$\text{Specificity} = \text{TrueNegatives} / (\text{TrueNegatives} + \text{FalsePositives}) \quad (4.9)$$

$$\text{Accuracy} = (\text{TrueNegatives} + \text{TruePositives}) / (\text{TrueNegatives} + \text{FalsePositives} + \text{TruePositives} + \text{FalseNegatives}) \quad (4.10)$$

The network sensibility, specificity and accuracy can be seen in table 4.2. The formulas that we use to calculate these values for the network are on the formulas 4.8, 4.9 and 4.10. This is the best network we could get with only these 7 features, but by implementing more features and increasing the number of samples we may get a better result. Also by putting the network on the Smartphone application, we can make the network adjust for each User, by performing additional training, making it more precise for that specific user. We test some other structures to the network and the information about those tests is on chapter 5.

4.1.4.5 Alarms

We already implemented a system of alarms. As we can see from figure 4.12 when a alarm is raised we change the color from green to red indicating that something is wrong with that behavior. The ChatBot also sends a message and a notification, if the user is not on the ChatBot screen. The Alarms, that the application can currently trigger are:

- Too much time Still.
- Not enough Exercise.
- Should spend more time at the University.
- Too much time at Home.
- Need more sleep.

The alarms are trigged by simple conditions and we define these conditions ourselves by using common sense. The conditions we defined are:

- Spend more that 12 hours per day still, triggers the "Too much time Still" .
- Less than 8 hours of Exercise per week, triggers the "Not enough Exercise".
- Less than 4 hours per day at the University, Triggers "Should spend more time at the University".
- Too much time at Home.
- Less than 6 hours of sleep per day, triggers the "Need more sleep".

4.1.5 Application Flow and Design

In this section we explore the application flow (the way the application interacts with the user) and design choices, UI and User experience (UX). The application was all designed with the same colors' patterns of a dark blue, on the Background, and a Light green for the main details. The graphics and charts that we use on the application also use the same colors' pallet.

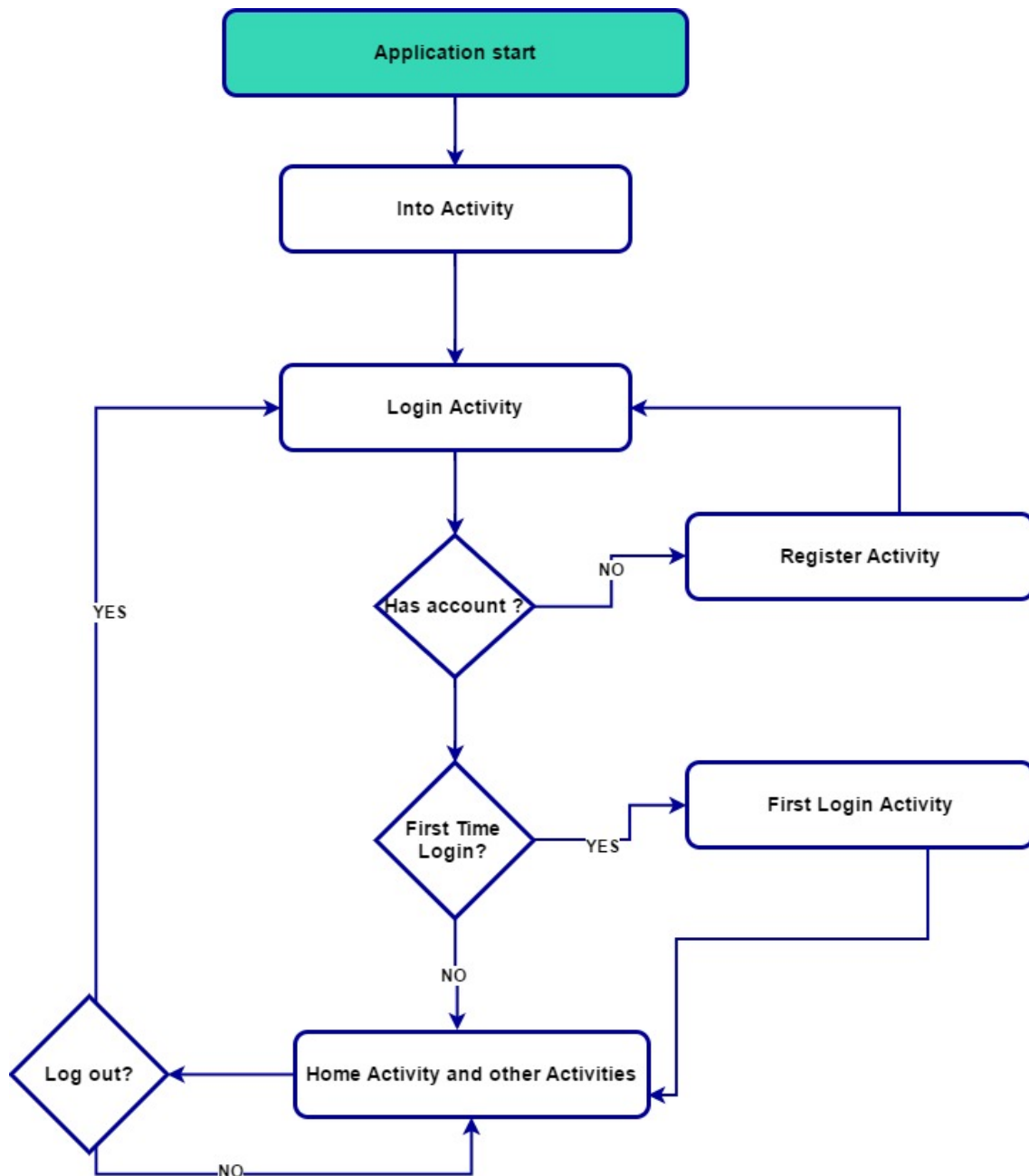


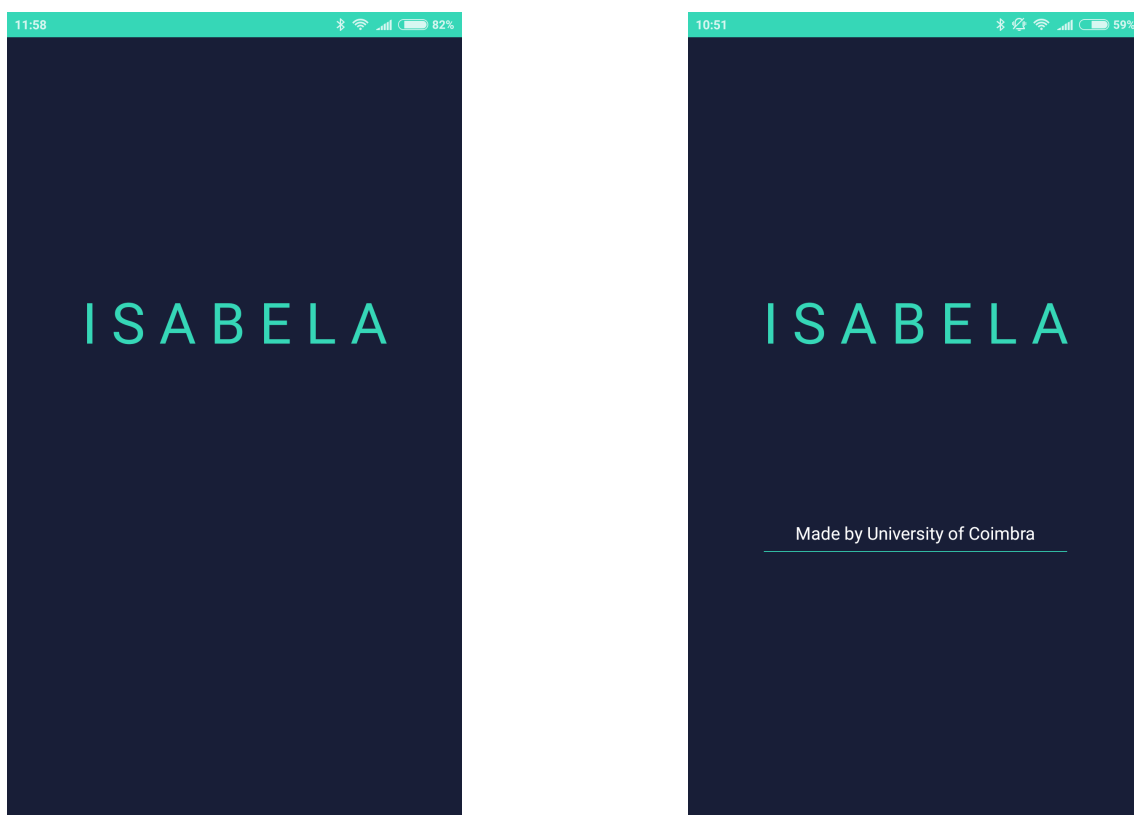
Figure 4.8: Mobile application flow

For the application to start the user has to click on the application's icon from the Smartphone's menu, then the application follows the diagram represented on the figure 4.8.

The Application starts with the Intro screen, figure 4.9, then after 5 seconds (or less if the user taps the screen twice), it changes to the Login screen. In the Login screen the application checks if there is any Access Token for the user stored in the Smartphone. If there is one the application advances to the Home screen without any User interaction needed. If there is not an Access Token the user is

prompted for is Login information. If the user does not have an account we can choose to go to the registration screen and create one.

After a successful login, the application starts the services and checks if this is the user first Login, by checking the FIWARE existing entities for the user entity. If this is the user first login the application changes to the First Configurations screen. When the user finishes these configurations we can move to the the Home screen and use the application functionalities. As it can also be seen from the diagram on figure 4.8 the user can also choose to Log out at any moment, stopping all the services and returning to the Login screen.



(a) Intro screen before the animation

(b) Intro screen after the animation

Figure 4.9: Introduction screen

The Intro screen purpose, figure 4.9, is just to display the name of the application and that this application is property of the University of Coimbra. As stated before this screen is changed automatically after 5 seconds. We choose to made a animation in this screen to improve UX. When the screen appears it only displays the application name, after 2 seconds the phrase "Made by University of Coimbra" appears and after 3 seconds the screens changes automatically to the Login or the Home screen. As stated before if the user wants he can also skip the animation by

double tapping the screen.

The layout from the Login screen and from the Sign up screen are identical as can be seen from figure 4.10. The only difference between the design of the two screens is the existence of an extra text box on the Sign Up screen to confirm the Password. The Login screen, figure 4.10 (left), has two buttons, one that does the Login using the information on the text boxes and another button that changes to the Sign up screen. The Sign up Screen, figure 4.10 (right), also has two buttons, one that allows the creation of a new user account with the information that the user inserts and a second button to return to the Login Screen. If the registration is successful the application returns automatically to the Login Screen and asks the user to insert is account.

It is important to refer, that do the nature of these actions, the Smartphone needs a stable connection to the Internet in order to perform these actions.



Figure 4.10: Login screen (left) and New Account Registration screen (right)

After the first successful Login the application starts the First Sign in screen, and asks the user for some configurations that are essential to the application's

proper operation. That screen that is prompted to the user on the first Login can be seen on figure 4.11.

On that screen the user must configure 3 options in order for the application to work properly: the Wifi SSID from the user House, the ID from the IoT kit that the user has at home, and if the user wants to only send data through Wifi connections. The Wifi SSID configuration is used to define the user location at home. The user should choose between all the configured networks in the Smartphone, and that network SSID will be saved and used as reference to define the user location. The IoT kit ID serves to create the link between the device and the entity that represents the room in the FIWARE. We also allow the user to choose if he only wants to send data through Wifi connections, because some user may not want to spend their mobile data plans. From figure 4.11 it is also visible that exists a fourth configuration: the user can indicate the Department to which he belongs. But this configuration isn't used on the application for now (is intended to be used once we map the AP and start using them to define location inside buildings).

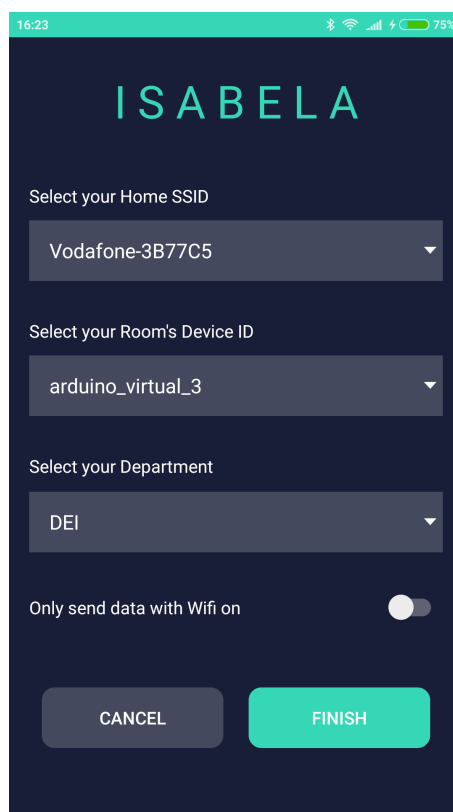
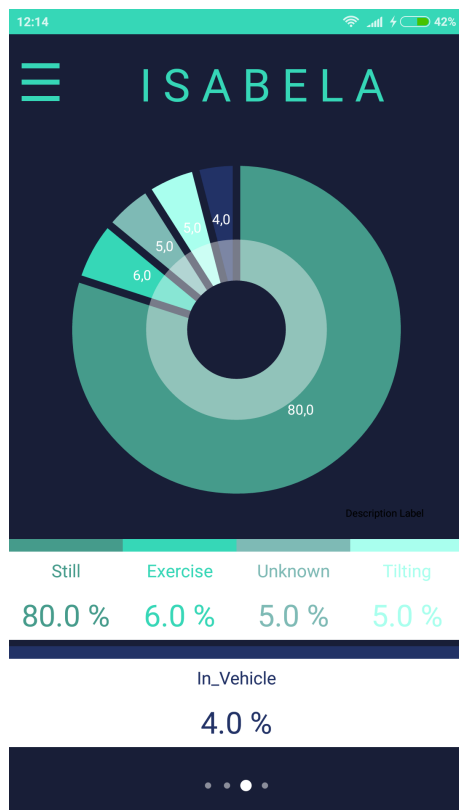
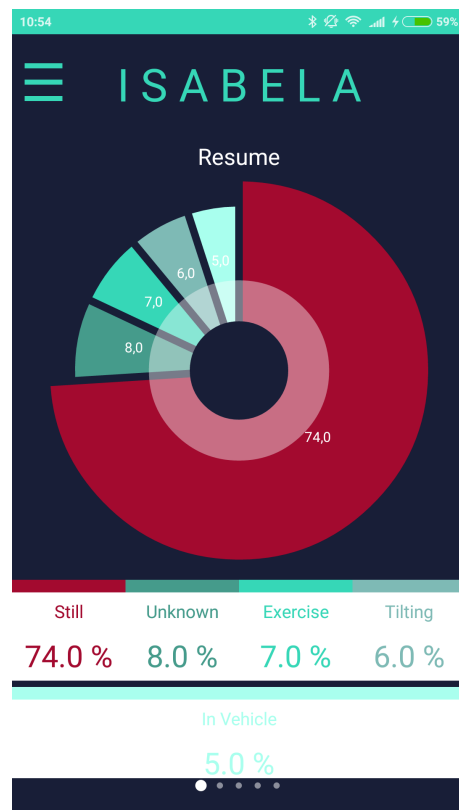


Figure 4.11: First configurations screen

On the Home screen we have a horizontal slide design, where the user can navigate between the screens by swiping right or left. The first screen is the resume of the user activity in a donut shape chart, with the amount of time the user spend in each activity in percentage. The figure 4.12 shows the activity chart, as well as the legend of the chart. We can also see that the display changes the chart colors when a alarm is raised. In this case the user spends too much time sill, the application changed the color of the chart and highlighted the slice of the chart that represented the "Still" activity.



(a) Activity screen



(b) Activity screen with alarm

Figure 4.12: Activity screen before (left) and after an alarm is raised (right)

The second home screen is the Location screen where we can see the distribution of the user time between the 3 places (Home, University and Other). As can be seen from figure 4.13 we also choose to create a different chart for each of the three features, the Activity, the Location and the Sociability. This way we improve the UX because the user can associate the Chart shape with the specific feature. The inverted pyramid chart style was chosen to represent the Location, because it resembles the pins that the navigation applications (for example Google Maps) use to represent specific positions. As can also be seen from figure 4.13 we use the same color palette, and we keep the style of the chart's legend.

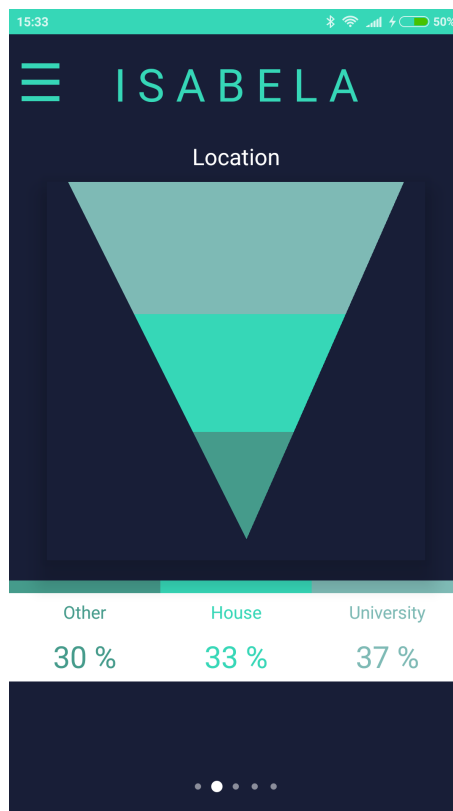


Figure 4.13: Location chart screen

As we explained in section 4.1.4.3, the sociability is given by a score and as such it can not be represented in a chart. So we opted to use a progress bar that goes from 0 to 100 percent. We also opted to use a circular progress bar because we find it more appealing than a regular progress bar and gives a feeling of interactivity. In this chart legend we show the total number of SMS, Calls and other factors that affect the sociability. These numbers are relative to a week. We believe that the total number is a more pertinent information, for the user, than the frequency or the ratio of SMS and Calls.

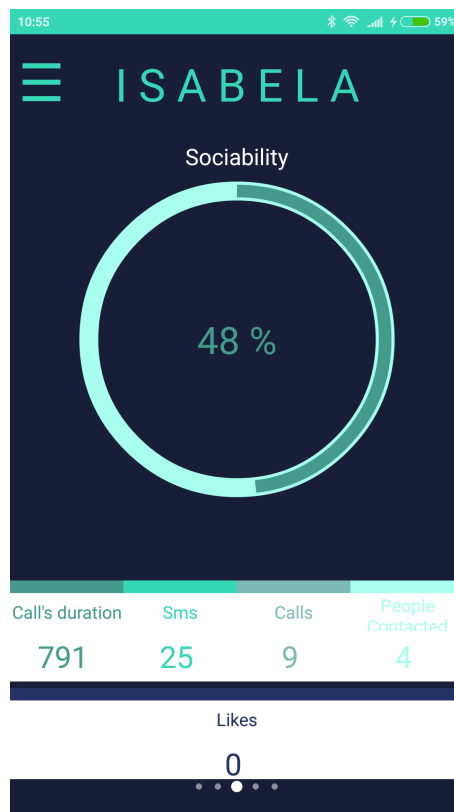


Figure 4.14: Sociability chart screen

Also to improve the UX and the UI appearance we created a navigation drawer, figure 4.15(a), that can be accessed through the button in the top left corner of the application (figure 4.15(b)). This navigation drawer is used to navigate through the activities or to Logout. The user can close this navigation drawer by pressing the "X" button on the top left corner or by sliding left over the navigation drawer. On the bottom of the navigation drawer we can also see the current version of the application.

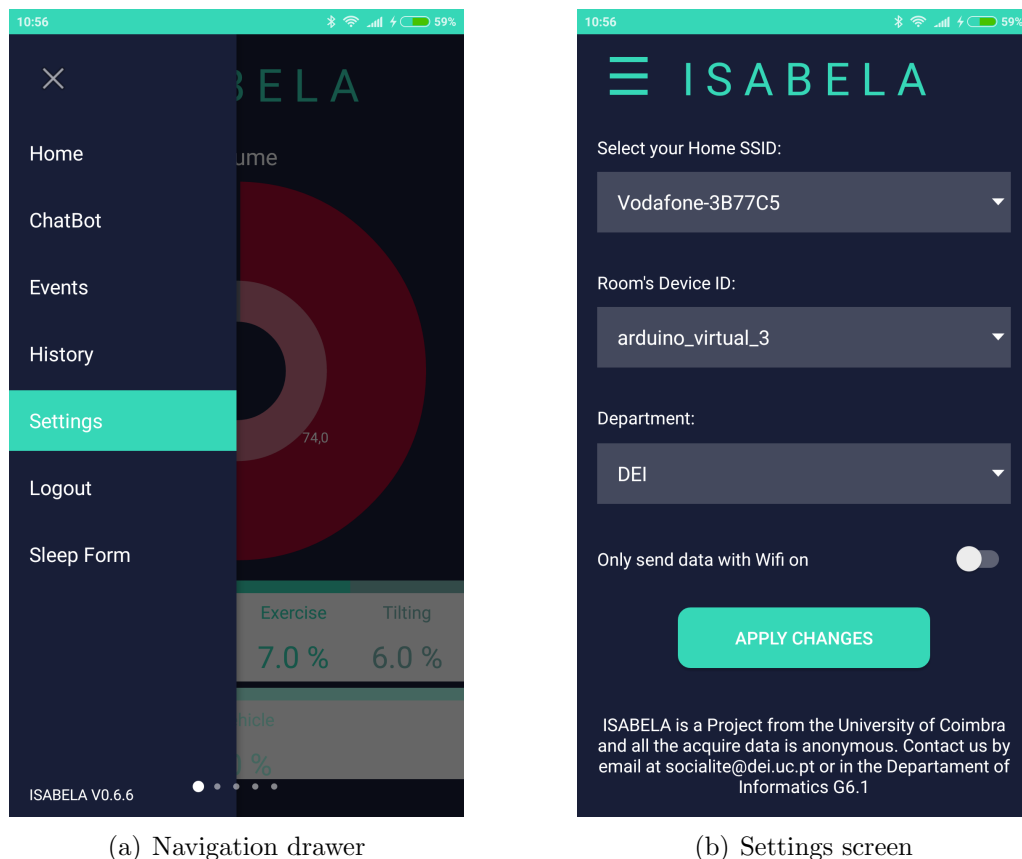


Figure 4.15: Navigation drawer (left) and Settings screen (right)

On figure 4.15(b) we can see the settings screen, where we can change the configurations made on the first configuration screen. This screen is almost identical to the First configurations screen (figure 4.11); the only difference between these two screens is that this screen only possesses a button to save the changes and as access to the navigation drawer. This screen allows us to change exactly the same 4 configurations: the Wifi SSID, the IoT kit ID, the option to choose how data is sent and to choose the Department (although this configuration is not used as explained before). Additionally on this screen is also the information on how to reach the development team in case of any fails during the tests.

By using the navigation drawer we can also navigate to the Events screen that can be seen in figure 4.16. This screen is made exclusively to improve the UX; that is if our application is going to warn the user about his classes the user should not be forced to switch applications just to have more information. As can be seen from figure 4.16, this screen also follows the same design as the previous ones. We used the same color pallet and the screen also has the navigation drawer button and the application name on the top of the screen. We show the events information with little cards that have the title and an image that changes depending on the type of event. As can be seen in figure 4.16(b) by pressing a card we can see the details of that event, that is the starting time, the ending time and the description of that event.

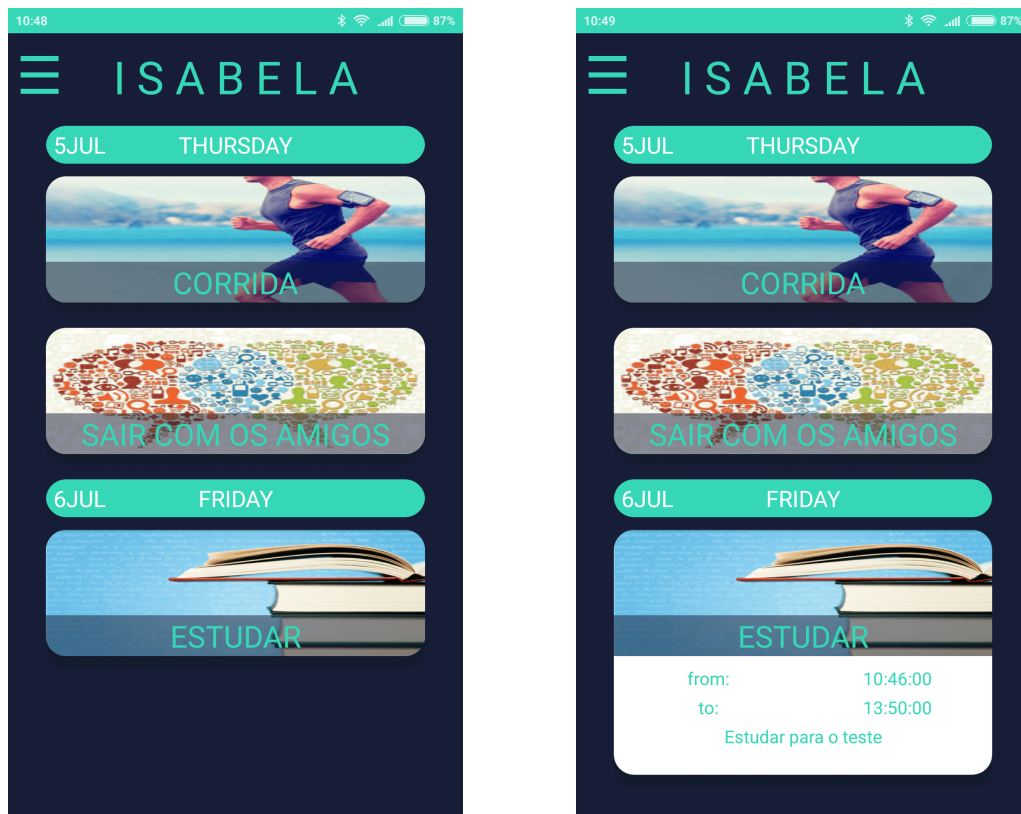
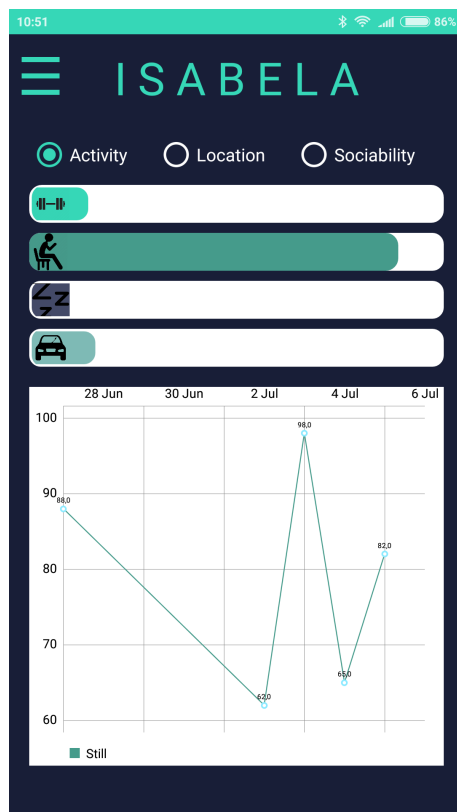


Figure 4.16: Events screen, with event details showing(right)

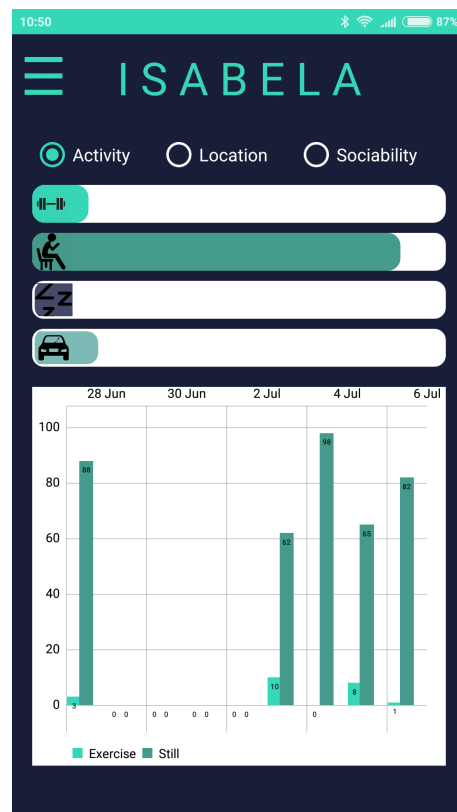
Once again, by using the Navigation drawer we can navigate to the historic screen, that can be seen in figure 4.17. As we can also see from figure 4.17 this screen allows us to see the information from the Activity, Location and Sociability features. We can change between these features by changing the selected radio button on the top of the screen.

Bellow the radio buttons we can see, in case of the Activity feature, 4 progress bars with an icon representing the four activities: exercise, still, sleep and in vehicle (from top to bottom). These bars represent the amount of time spent doing each activity, in percentage and each bar has a different color that is used to represent that activity on the chart.

In figure 4.17(a) we can see the historic screen when only the activity is selected. In this case the still activity (the bar for the still activity is bigger than the others). When we only select one type of activity the application shows the information in a line chart, with the x-axis representing the time in days and the y-axis showing the amount of time that the user spent doing that activity in percentage.



(a) Historic Screen



(b) Historic screen grouped bar style

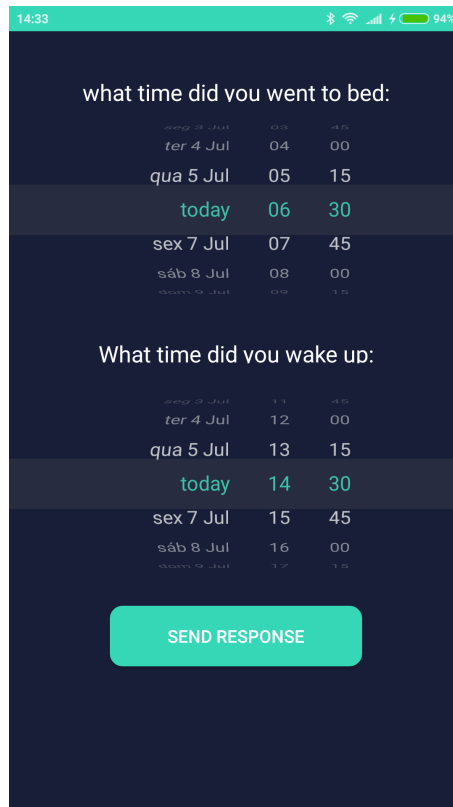
Figure 4.17: Historic screen

In figure 4.17(b) there are two types of activities selected, Still and Exercise,

and as we can see the display of the chart switches from a line chart to a grouped bar chart. The grouped bar chart can show 2, 3 or 4 activities depending on how many activities we select. This type of chart is better to compare the values of the different activities. We can also see from figure 4.17 that the chart has an legend with the type of activities and the color in which they are represented.

The representation of the Location and Sociability features in this screen is the same as for the Activity. In the Location we can see the amount of time, in percentage, that the user spend at Home, at the University or at Other place and in the Sociability we can see the number of SMS received, the number of SMS sent, the number of Calls received, number of Calls made and the duration of the Calls. It is important to refer that for the Activity and for the Sociability we are not showing all of the types of the data we collected because we believe that some of them may not interest the user, like in the case of the Activity tilting or the number of calls missed.

We also created a screen that serves to collect the ground truth for the sleep classifier, and we call it sleep form. The user can access it from the last option of the navigation drawer as can be seen in figure 4.15(a). But the application is also programmed to deploy a notification each morning between 10 am and 11 am, remembering the user to fill the sleep form and by clicking that notification the user is sent to the sleep form screen (figure 4.18). We can see from the figure the user only needs to indicate what time we went to sleep and what time he waked up.



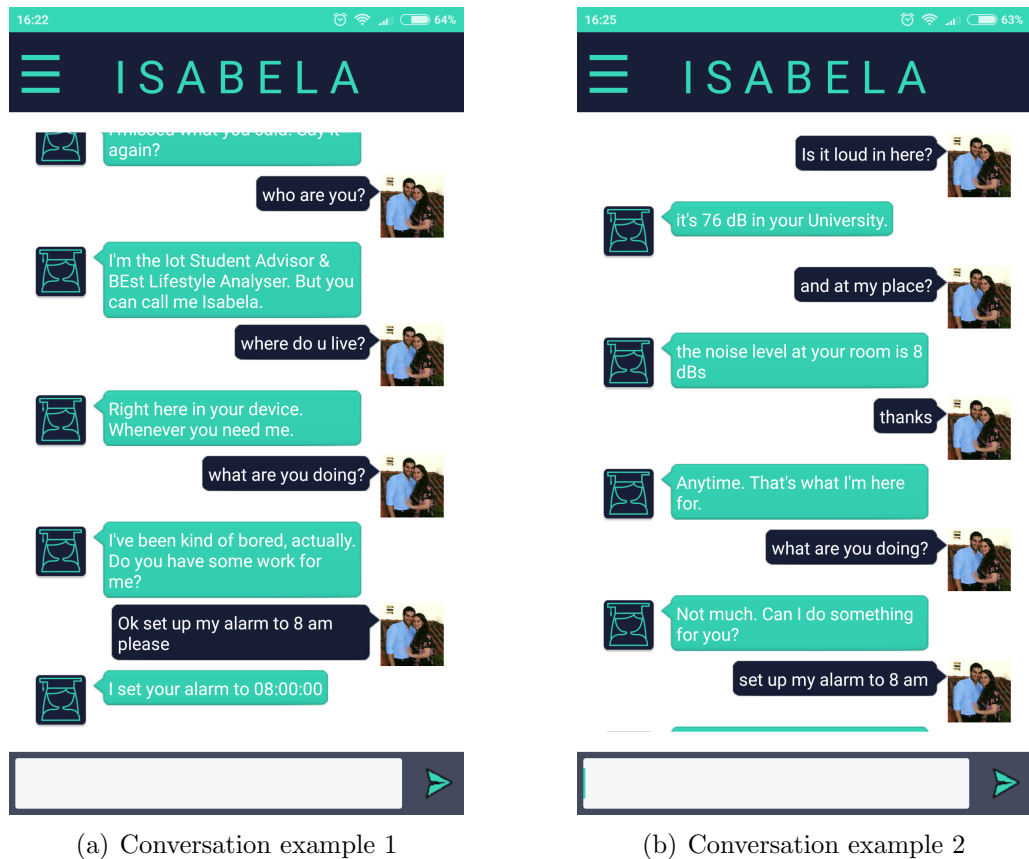
The screenshot shows a mobile application interface for recording sleep data. The screen is dark-themed with a teal header bar at the top displaying the time 14:33, signal strength, Wi-Fi, and a 94% battery level. The main content is divided into two sections: 'what time did you went to bed:' and 'What time did you wake up:'. Each section contains a table with columns for the day of the week, the hour, and the minute. The 'today' row in both tables is highlighted in a lighter shade. At the bottom of the screen, there is a teal button labeled 'SEND RESPONSE'.

Day	Hour	Minute
ter 4 Jul	04	00
qua 5 Jul	05	15
today	06	30
sex 7 Jul	07	45
sáb 8 Jul	08	00
domingo 9 Jul	09	15

Day	Hour	Minute
ter 4 Jul	12	00
qua 5 Jul	13	15
today	14	30
sex 7 Jul	15	45
sáb 8 Jul	16	00
domingo 9 Jul	17	15

Figure 4.18: Sleep form screen

The application has one more screen, the ChatBot screen that can be seen in figure 4.19. As we stated before, in this document, the main purpose of the ChatBot is to serve as a way to interact with the students and tell them what is wrong. But the ChatBot has other functionalities, that can be seen in the figure 4.19.



(a) Conversation example 1

(b) Conversation example 2

Figure 4.19: ChatBot screen

On the figure 4.19(a) we can see the Small talk functionalities, where the user can ask simple question like "Who are you?" or "where do you live?". And on the end of the figure 4.19(a) we can also see the ChatBot setting an alarm for the user. On figure 4.19(b) we can see the functionalities that interact with the FIWARE, that is the student can ask the values of the sensors from the IoT kit that in his house or from the kit that is at the University. On that figure we can also see the context awareness of the ChatBot when we ask "It's loud in here?" it queries the FIWARE for the user location first, then queries it for the intended information in this case the sound level at the University. In that same figure we also can see that is also aware of the context of the conversation. By asking "And at my place?" the ChatBot can understand that we where previously talking about the sound level and returns the response for the sound level at the students home. This of course

is just the tip of the iceberg of what can be achieved by implementing a ChatBot as mean of actuation. And some possible future implementations are addressed in chapter 6.

Lastly we also show the information from the Smartwatch on the Smartphone. As we can see in figure 4.20, that information is showed on the end of the Activity Screen. We indicate the current step count for the day and the last heart rate value available. We also receive the result of the Activity Recognition from the Smart-Watch but that information overrides the Activity Recognition of the Smartphone and as such it is already represented on the donut chart.

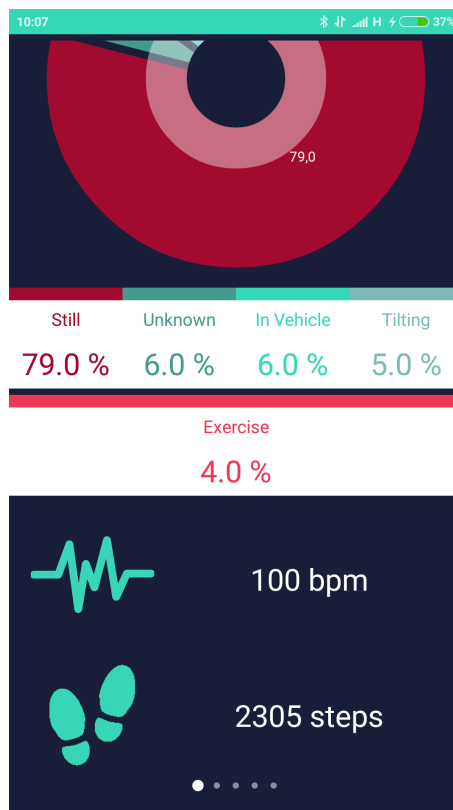


Figure 4.20: Wear data on the Smartphone

4.2 SmartWatch application

The Smartwatch application serves as an companion for the mobile application and as such is much simpler. In this section we explain in detail the development of the wear application and we can divide the development of the application in 4 sections the acquisition, the communication, the storage and the display of the application.

4.2.1 Data acquisition

The data acquisition on the Smartwatch application is much simpler than on the Smartphone application. The Smartwatch application only makes the acquisition of the 3 values from the sensors:

- The Heart Rate.
- The Step Count.
- The Activity

To make this acquisition we created a service called MainService that runs on the background at all times, after the user opens the application. This service works in the same way as the MainService on the Smartphone. It implements the `SensorEventListener`[48] interface that allows us to know when a value of a sensor is changed. By using the `SensorManager` class[46] we can register this listener to the desired sensors, in our case the heart rate sensor and the step count sensor.

The architecture of the MainService from the SmartWatch can also be represented by the figure 4.1 from section 4.1.1.1. We have a listener that is triggered every time a registered sensor is changed. The listener saves the sensor's updated value on a variable and when a timer goes of that variable value is stored on a DB. Although the architecture is the same we had to implement a second timer in the MainService in the Smartwatch. That timer function is to unregister the heart rate sensor after it runs for 10 seconds and register it again every 30 seconds. We do this because the heart rate sensor consumes a lot of battery do to the hardware, this of course is a big issue on a device with low battery power like the SmartWatch. Just as we do on the Smartphone on the wear application we save the values every 5 seconds.

The Activity once again is taken from the Google Activity Recognition API

and as such the acquisition process is the same as in the Smartphone application. The information about how this acquisition is made can be found in sections 4.1.1.1 and 4.1.4.1.

4.2.2 Communication

As we stated before on section 4.1.2.3 the intended architecture for the android wear doesn't allow us to connect the device directly to the Internet. This happens because the device manages the Wifi and the Bluetooth connections automatically, to improve battery life. Once the device is connected to the Smartphone by Bluetooth the Smartwatch disconnects the Wifi to save battery. This of course would not allow us to send data when the Smartwatch and the Smartphone are close to each other (most of the time).

For this reason we have to use Android Wear Message API. We already explained how we implement this API on the Smartphone in section 4.1.2.3. To implement this API on the SmartWatch we use a timer on the MainService, that every 30 seconds reads the DB and try to send the data to an available device. To send the data we start a new service, that extends the IntentService class[44]. This service uses the Capabilities API[60] to make a scan for available Nodes (devices connected to the SmartWatch) that have the desired capability. Once the API finds one, the service starts sending the saved instances of data to the Smartphone. When a instance of the data is sent to the Smartphone it is also deleted from the DB on the Smartwatch. Once the data is received by the Smartphone it is processed as explained on section 4.1.2.3.

4.2.3 Storage

Since the android wear application is simpler than the Smartphone application, the DB is also much simpler. The DB on the SmartWatch only has one entity that works like a log for the sensed data. The entity structure can be seen in figure 4.21.

As can be seen from figure 4.21 we save the Activity, the heart rate and the step count. We also save a time stamp, because the Smartwatch data can be saved in the Smartwatch for a while and we would lost the time context of the data. This way we ensure that we can relate the wear data with the Smartphone data. The entity has also the device ID attribute so we can identify where the data came from on the Smartphone. And as all the other DB entities this one has an ID attribute that is a single identifier for each instance of saved data.

Wear_data
+ id (auto increment): Long
+ device_ID: String
+ Step_cout: String
+ Heart_Rate: String
+ Activity: String
+ TimeStamp: String

Figure 4.21: DB on the Smartwatch

4.2.4 Display

The Display of the Smartwatch application, like the other parts of the application, in comparison to the Smartphone application, is less elaborated. The display on the Watch is small and as such we can't show a lot of information at once. So we opted for a simple layout where we can see the name of the application and the 3 sensors values in real time. As we can see from figure 4.22 the display of the heart rate and step count is similar to the one on the Smartphone application UI, with a symbol representing each of the two values and the actual value in front of it. The Activity recognition value appears in simple text under the application name.

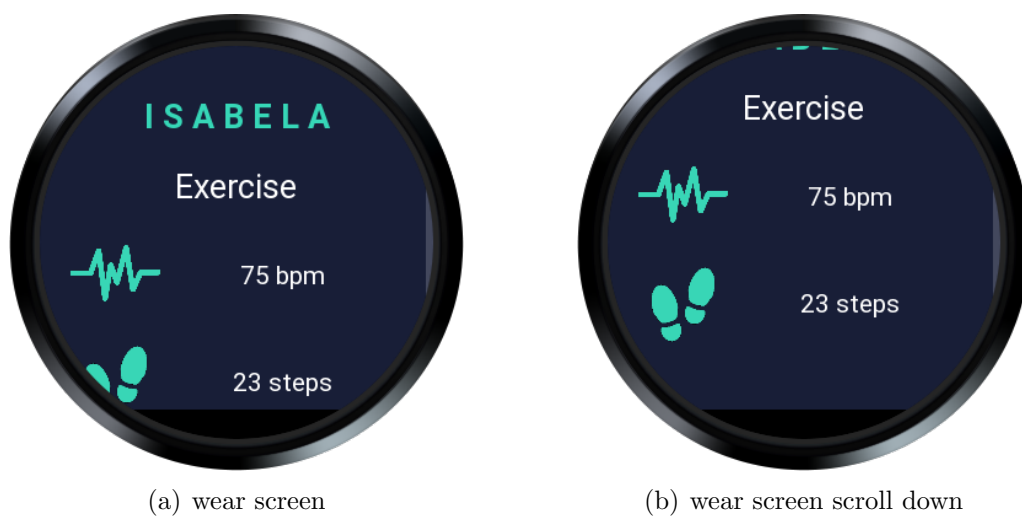


Figure 4.22: Smartwatch screen

Tests and Results

The majority of this work was focused on the construction of the system architecture, but we have also done some tests on battery life, network traffic and on the sleep classification. Those tests are explained in the next subsections.

5.1 Battery life

Low battery consumption is one of the biggest requirements in mobile applications, because if a application drains the battery life of the phone to quickly, no matter at application it is the user is going to uninstall it. So since the beginning of the project we took that in account and tried to developed the application in a way that did not affect the normal use of the phone.

On figure 5.1 we can see a battery consumption report for the Smartphone application. On the delimited red area, of figure 5.1, we can read "Energy consumption level 8,35 mAh" and that means that the application consumes an average of 8,35 mA(miliAmperes) per hour. This value of course depends on how much the user opens the application, because the display and the rendering of the visuals consume energy as well. The value of 8,35 mAh is obtain for minimal use pattern, that is the user doesn't spend to much time with the application open and only the background services run continuously. This value serves as an indication of the background task consumption. Now-a-days Smartphones have become larger and their batteries have increased as well. The regular Smartphone now-a-days has a battery of 3000-4000 mAh, that means that the application, by itself consumes less than 1% of the battery of the Smartphone.

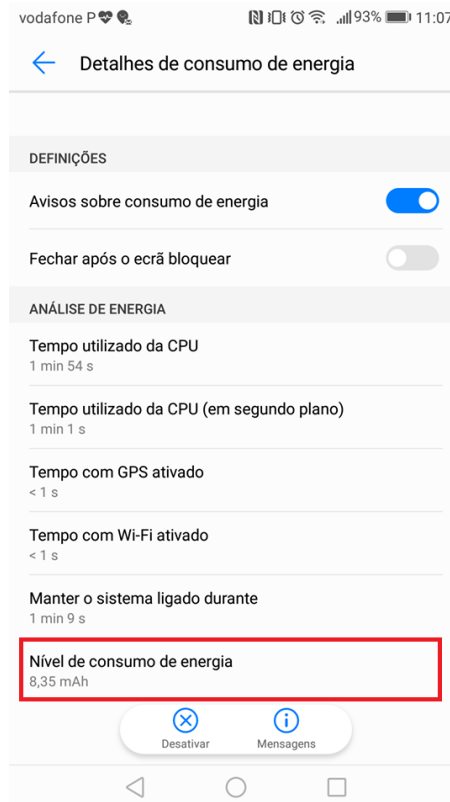


Figure 5.1: Battery consumption information on the

However there are other factors that affect battery consumption, like for example the use of the Google Play Services[68] and the use of the Wifi. Although we can see how these external components affect the Smartphone battery life we can not track the actual contribution of the application on each external component. So in order to test the application to see how it would affect a real device we installed the application on an Xiami Redmi Note 3, that has a battery of 4050 mAh. The Smartphone lasts in average 24 hours on a regular use, with GPS enabled, and after we installed the application the battery life was decreased to roughly 23 hours. Although testing with only one Smartphone does not guarantee viability of the results, we can conclude that the application does not affect, significantly, the battery life of the Smartphones.

5.2 Network traffic

Now-a-days we use our phones even more than the computers to access the Internet[69]. People need to always be able to connect to the Internet and as such they don't want to spend enormous amounts of data from their mobile data plans

on applications. So this creates a new requirement for all applications, including ours, the management of network traffic. As we stated before the users can choose to only sent data when they are connect to a Wifi network and that is the easiest way to reduce traffic.

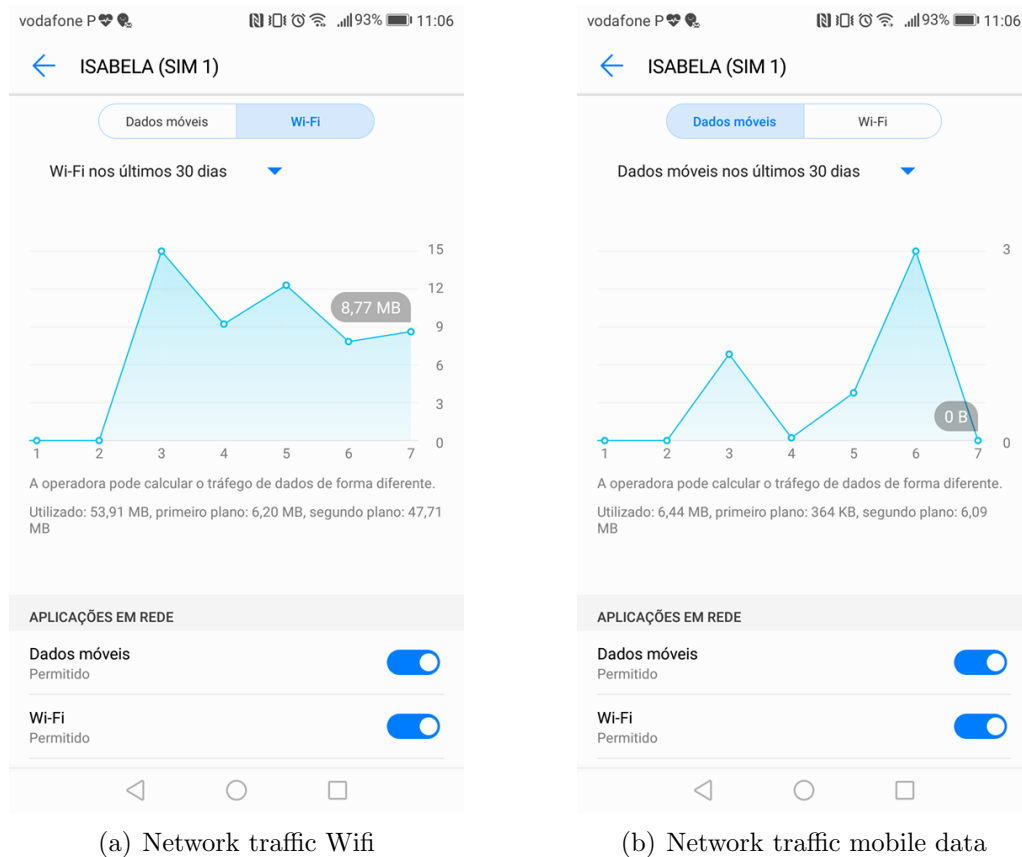


Figure 5.2: Network traffic reports

Although we could prevent unwanted traffic, like we stated before, we wanted to verify how much network traffic the application was generating. In order to do so we installed the application on an Smartphone and leave it running for 4 days, as can be seen from figure 5.2. On figure 5.2(a) we have a chart with the amount of traffic generated by the application in Wifi connections and on figure 5.2(b) we have the amount of traffic generated by the mobile data. We can also see from figure 5.2 that the average network traffic generated per day is 12,5 megabytes (Mb) (50Mb in 4 days). The discrepancy between the days can be explain by the fact that some days the test subject used the application more than others, that is by opening the application's UI we generate more traffic by requesting new data from the FIWARE. Although if we consider these values as the regular scenario, that is students have a Wifi connection at the University's facilities, at home and in some public spaces

(shoppings, etc), then only 10% of the data is sent via an mobile data connection (roughly 5Mb out of 50Mb). If we consider the 5Mb per day at the end of the month the application consumes 150Mb.

The numbers that we obtained in these tests show, that even if the user allows the application to transfer data via the mobile network, the amount of traffic that the application generates is small.

5.3 Sleep Classification tests

On this section we present the results obtain by using different structures of networks.

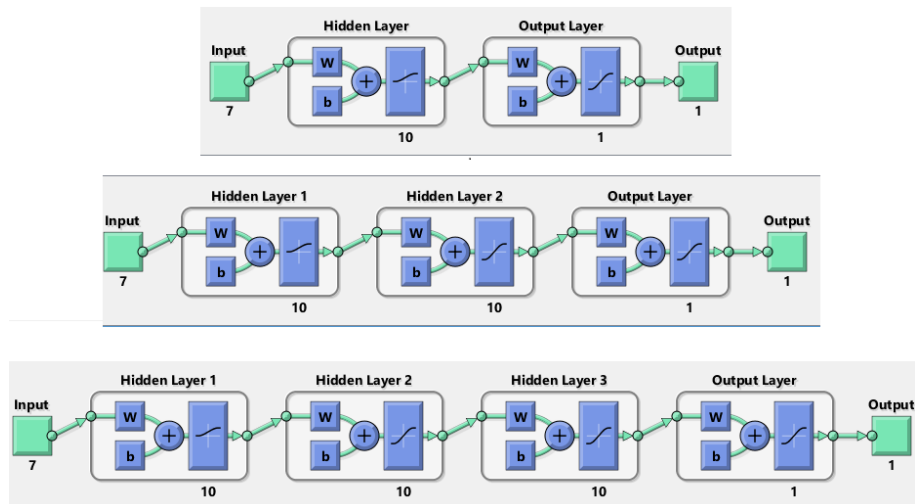


Figure 5.3: Structure of the tested networks

The structure of the networks follow the patterns from figure 5.3. As we can see from the figure we tried 3 structures of networks: the first with 1 hidden layer, then 2 hidden layers and the last structure with 3 hidden layers. All the structures have a input layer with 7 features, and as stated before the features are:

- the activity
- the day of the week
- the light intensity
- the sound amplitude
- the proximity

- the time to the next alarm
- the phone lock status

The output layer of all structures has only one neuron that classifies the output as sleeping or not at sleep. We also vary the number of neurons in the 3 structures between 1, 2,5,10, 15 and 20 neurons and the results for the Sensibility, Specificity and Accuracy can be seen in the tables 5.4, 5.5 and 5.6. We can also see from the tables that we did 3 runs, for each configuration of the network, because we were randomizing the initial states of the networks and just one run could be misleading as can be seen by comparing the different runs for the same structure, on table 5.1 for example.

Table 5.1: Results for network with 1 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity(%)	Accuracy (%)
1	1	0,00	100,00	63,65
	2	88,54	83,66	86,03
	3	78,85	95,89	89,71
1-1	1	0,00	100,00	63,71
	2	80,33	95,78	90,18
	3	80,18	96,04	90,29
1-1-1	1	81,66	94,41	89,79
	2	75,84	96,57	89,04
	3	0,00	100,00	63,71

Table 5.2: Results for network with 2 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity(%)	Accuracy (%)
2	1	58,89	97,70	83,61
	2	09,20	98,69	66,21
	3	67,67	96,94	86,32
2-2	1	08,12	98,98	65,97
	2	71,76	96,36	87,43
	3	74,26	92,38	85,80
2-2-2	1	06,64	99,50	65,80
	2	81,10	96,54	90,93
	3	05,57	99,59	65,47

Table 5.3: Results for network with 5 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity(%)	Accuracy (%)
5	1	09,14	98,63	66,15
	2	54,95	97,81	82,26
	3	29,47	96,60	72,22
5-5	1	74,72	96,77	88,77
	2	71,70	96,16	87,28
	3	57,49	98,71	83,61
5-5-5	1	79,01	96,04	89,84
	2	33,70	94,88	72,67
	3	60,93	95,61	83,02

As can be seen from tables 5.1,5.2 and 5.3, we can obtain good results with small networks and just a few neurons. However we can notice from tables 5.1 and 5.2 that these networks are very unstable, that is, they give good results but depending on the initiations conditions they can also starting to diverge. From table 5.1 to 5.2 we see a slight increase in stability, there no more networks with 0% sensibility. In table 5.3 we can also see that the stability increased in comparison to table 5.2. So to obtain a more stable network we decided to increase the neuron count and do some additional testing. The results of those tests can be seen in tables 5.4, 5.5 and 5.6.

Table 5.4: Results for network with 10 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity(%)	Accuracy (%)
10	1	61,61	98,35	85,38
	2	29,98	96,70	73,15
	3	63,64	97,50	85,55
10-10	1	53,59	99,13	83,50
	2	70,81	95,85	87,01
	3	69,27	96,75	87,01
10-10-10	1	29,13	96,58	72,76
	2	80,06	94,86	89,63
	3	66,40	96,72	86,01

Table 5.5: Results for network with 15 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity(%)	Accuracy (%)
15	1	52,68	97,68	81,79
	2	33,06	95,21	73,27
	3	62,31	98,11	85,47
15-15	1	60,23	98,40	84,93
	2	69,90	96,23	86,93
	3	70,00	95,50	86,50
15-15-15	1	60,55	92,97	81,53
	2	74,64	94,51	87,50
	3	47,69	98,72	80,70

Table 5.6: Results for network with 20 neurons on the hidden layers

Structure	Run	Sensibility (%)	Specificity (%)	Accuracy (%)
20	1	55,87	98,11	83,20
	2	67,96	94,80	74,73
	3	78,31	94,37	88,70
20-20	1	77,83	94,86	88,85
	2	63,00	94,20	83,18
	3	58,16	97,12	83,37
20-20-20	1	61,56	97,91	85,08
	2	87,99	94,06	92,71
	3	85,27	95,15	91,67

All values from tables 5.1 to 5.6 were obtained for networks with an activation function Log-Sigmoid on every hidden layer and as such we decided to compare the results of the best structure with different activation functions. And from that comparison we obtained the table 5.7. In this case we also decided to make 3 runs for each configuration.

Table 5.7: Results for different activation function on a network with a 20-20-20 structure

Activation function	Sensibility (%)	Specificity (%)	Accuracy (%)
Log-Sigmoid	81,42	95,31	90,29
Sigmoid	0,00	100,00	63,84
Linear	0,00	100,0	63,84
BiPolar	0,00	100,0	63,84
Gauss	0,00	100,0	63,84
TangH	05,75	99,91	66,54

From the table 5.7 we can see that the Log-Sigmoid Activation function is the only one that gives viable results.

From these results we can conclude that the best network is the network with 3 hidden layers and 20 neurons in each layer and with an Log-Sigmoid Activation function in each layer. However 3 layers networks take some time to train specially on the Smartphone, and the use of the CPU can affect the battery life. But there are ways to work around this problem like, for example by only training the network when the Smartphone is charging. This way we will not drain the battery and normally the user will not use the phone while it is charging. As such we don't need to worry about affecting the user experience by slowing the Smartphone (we can also control when the phone is locked and pause the train when it is not).

All the results in the tables above were made for a validation dataset that was taken from the original dataset and not used on the training. So we decided to also compare the results of the training dataset with the results from the validation dataset. So we trained the best network (3 layers, 20 neurons and activation function Log-Sigmoid), then compute the results for the same training dataset and the validation dataset. We obtained the results of table 5.8. As we can see the results are very similar and with that we can conclude that the dataset is very homogeneous, as all of the examples are similar. This also tells us that we need a larger dataset to obtain more reliable results.

Table 5.8: Results training data vs validation data

Dataset	Sensibility (%)	Specificity(%)	Accuracy (%)
Training	83,58	95,43	93,03
Validation	83,34	93,40	91,12

Conclusions and Future work

6.1 Conclusions and Future work

With this project we aimed to create an application, that could monitor and improve the students' performance and, in doing so, we wanted to also use the concept of HITLCPS. With this in mind we explored the concepts of IoT, CPS and mobile sensing. We also evaluated how we could use these concepts to create such an application. And because of all of the concepts that this project encompasses, it was a very ambitious project from the start.

By implementing all these concepts we were able to develop an IoT platform using FIWARE, an Android application for both the Smartphone and SmartWatch and also IoT kits with Raspberry Pi and Arduino. And although mobile motorization is still a relatively new concept and with many drawbacks, such as privacy concerns and lower computation power, we believe that this kind of project will be very important in the near future.

We can conclude that at the end the main objectives of this project were all accomplished, and we believe that we made some advances toward making people centering applications more common. But we also believe that there is still a lot of work to be done to improve the system. In the next section we explore some ideas for future work.

We also concluded that the results we obtained from our classification/ inferences are not very reliable. Because we needed to improve the techniques we use, to classify/infer, and above all we need to test the system in a real situation. Using the application in real scenarios was also one of the objectives of the project in the beginning. But real test were impossible to achieve during the first year of the project due to the project nature. To infer the students' performance we need to collect data through out an entire semester and that requires a lot of previous

preparation. However we did small tests between the LCT-group members, that proved to be promising.

This project was also of great importance to my personal development, since it helped me to improve many of the skills I gained during my academic course, acquire new ones and further my knowledge in software development. Apart from this project I am also writing a scientific paper and I was also co-author in two other publications.

6.2 Open challenges

Through out this thesis we mentioned some times the possibilities for further improvement in the some features of the application. In this section we summarize all of the open challenges we identified.

As we stated we define the discrete the Location of the student through the SSID of the Wifi networks and this way we can detect if the student is at home or in the University. But we lack the ability to know in which facility of the University the student is. By mapping all the AP of the University we could locate the student with more precision inside the buildings. This of course could be used to improve the class attendance detection, by pin pointing the location of a student to a classroom and by checking his schedule we can check if we is attending a class or not. Although we have already done this to some level by checking the schedule and checking if the student is at the University, this assumption can be wrong since the student can be at the University and not at the intended classroom.

Also as future work we need to implement a different type of classification on the Sociability. As we stated before we have already had a reunion with a investigation group from the Psychology Faculty to create a partnership between the two groups. This group is coordinated by professor Dr. Carlos Barreira and their main investigation focus is students' depression and learning assessment. We believe that this partnership will give us a new insight on what steps to take next in order to detect sociability and other emotional factors.

The sleep detection module also needs to be improved by adding new features and increasing the dataset. We also need to perform more tests and use different network architectures.

The ChatBot can also be improved in the future by adding new functionalities,

and extending its context awareness.

Another open challenge is the improvement of the alarm system. Now the alarm system is only based on conditions, that verifies if the amount of time a student performs a certain activity exceeds or is less than a desired amount. In the future we would like to apply machine learning techniques to perform pattern detection. This of course requires that we do a large study and collect data from students through out an semester.

In conclusion the challenges described above represent some good opportunities for new research work.

Bibliography

- [1] <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>, “Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions),” accessed in 30/05/2017.
- [2] F. B. David Nunes, Jorge Sá Silva, *A Practical Introduction to Human-in-the-Loop Cyber-Physical Systems*. Wiley-IEEE Press, December 2016.
- [3] R. Wang, F. Chen, Z. Chen, T. Li, G. Harari, S. Tignor, X. Zhou, D. Ben-Zeev, and A. T. Campbell, “StudentLife,” in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '14 Adjunct*, 2014.
- [4] t. <http://greenrobot.org/greendao/>.
- [5] S. Li, L. D. Xu, and S. Zhao, “The internet of things: a survey,” *Information Systems Frontiers*, 2015.
- [6] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Communications Surveys and Tutorials*, 2014.
- [7] S. K. Khaitan and J. D. McCalley, “Design techniques and applications of cyberphysical systems: A survey,” *IEEE Systems Journal*, 2015.
- [8] D. S. Sousa Nunes, P. Zhang, and J. Sa Silva, “A Survey on human-in-The-loop applications towards an internet of all,” *IEEE Communications Surveys and Tutorials*, 2015.
- [9] http://www.jornaldenegocios.pt/economia/educacao/detalhe/taxas_de_abandono_e_desemprego_por_cursos_podem_ajudar_indecisos, “Ensino superior: taxas de abandono e desemprego por cursos.”

- [10] <http://www.healthline.com/health/depression/college-students>, “Depression and college students.”
- [11] <https://www.fiware.org/>, “Fiware.”
- [12] <https://www.scrumalliance.org/why-scrum>, “Scrum alliance.”
- [13] <https://about.gitlab.com/>, “Gitlab.”
- [14] <https://slack.com/>, “Slack.”
- [15] <https://www.skype.com/pt/>, “Skype.”
- [16] <https://developer.android.com/studio/index.html>, “Android studio.”
- [17] <https://www.linuxfoundation.org/>, “The linux foundation.”
- [18] <https://www.android.com/wear/>, “Android wear.”
- [19] <http://expandedramblings.com/index.php/android-statistics/>, “70 amazing android statistics.”
- [20] https://www.android.com/intl/pt_pt/, “Google android.”
- [21] https://ec.europa.eu/research/fp7/index_en.cfm, “7th framework programme.”
- [22] N. Armando, D. Raposo, M. Fernandes, A. Rodrigues, J. Sá Silva, and F. Boavida, “WSNs in FIWARE – Towards the Development of People-centric Applications,”
- [23] <https://www.fiware.org/foundation/>, “Fiware foundation.”
- [24] <https://catalogue.fiware.org/>, “Fiware catalogue.”
- [25] K. Hinsén, K. Läufer, and G. K. Thiruvathukal, “Essential tools: Version control systems,” *Computing in Science and Engineering*, 2009.
- [26] t. <http://www.heatonresearch.com/encog/>.
- [27] <https://www.arduino.cc/>, “Arduino.”
- [28] <https://www.raspberrypi.org/>, “Raspberry pi.”
- [29] <https://api.ai/>, “api.ai.”
- [30] K. Ashton, “In the real world, things matter more than ideas,”
- [31] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, “Cyber-Physical Systems: The Next Computing Revolution,”

-
- [32] V. Issarny, M. Caporuscio, and N. Georgantas, “A perspective on the future of middleware-based software engineering,” *Future of Software Engineering*, vol. 00, pp. 244–258, 2007.
- [33] t. <https://fiware-orion.readthedocs.io/en/master/>.
- [34] R. Wang, G. Harari, P. Hao, X. Zhou, and A. T. Campbell, “SmartGPA,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*, 2015.
- [35] A. Zenonos, A. Khan, G. Kalogridis, S. Vatsikas, T. Lewis, and M. Sooriyabandara, “HealthyOffice: Mood recognition at work using smartphones and wearable sensors,” in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2016*, 2016.
- [36] N. D. Lane, M. Mohammad, M. Lin, X. Yang, H. Lu, S. Ali, A. Doryab, E. Berke, T. Choudhury, and A. T. Campbell, “BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing,”
- [37] <https://catalogue.fiware.org/enablers/backend-device-management-idas>, “Backend device management - idas.”
- [38] <https://www.mongodb.com/>, “Mongodb.”
- [39] <https://www.mysql.com/>, “Mysql.”
- [40] <https://github.com/telefonicaid/fiware-cygnus>, “Fiware- cygnus.”
- [41] <https://catalogue.fiware.org/enablers/identity-management-keyrock>, “Identity management keyrock- fiware.”
- [42] <https://github.com/telefonicaid/fiware-sth-comet>, “Comet- short time historic.”
- [43] <https://github.com/ckan/ckan>, “Ckan.”
- [44] <https://developer.android.com/reference/android/app/IntentService.html>, “Intentservice class.”
- [45] <https://developer.android.com/reference/android/hardware/package-summary.html>, “android.hardware.”
- [46] <https://developer.android.com/reference/android/hardware/SensorManager.html>, “Sensormanager.”

- [47] <https://developer.android.com/reference/android/hardware/SensorEvent.html>, “Sensorevent.”
- [48] <https://developer.android.com/reference/android/hardware/SensorEventListener.html>, “Sensoreventlistener.”
- [49] <https://developer.android.com/reference/android/media/MediaRecorder.html>, “Mediarecorder api.”
- [50] <https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi>, “Google activity recognition api.”
- [51] <https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html>, “Localbroadcastmanager.”
- [52] <https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi>, “Google’s fused location provider api.”
- [53] <https://developer.android.com/reference/android/location/LocationListener.html>, “Location listener.”
- [54] <https://inforestudante.uc.pt/nonio/security/login.do>, “Inforestudante.”
- [55] <https://developer.android.com/reference/android/content/BroadcastReceiver.html>, “Broadcastreceiver.”
- [56] <https://developers.facebook.com/docs/graph-api>, “Facebook’s graph api.”
- [57] <https://developer.android.com/reference/android/content/Intent.html>, “Intent class.”
- [58] <https://developer.android.com/reference/android/content/IntentFilter.html>, “Intentfilter class.”
- [59] <https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageApi>, “Android wear message api.”
- [60] <https://developers.google.com/android/reference/com/google/android/gms/wearable/CapabilityApi>, “Android capabilities api.”
- [61] <https://developers.google.com/android/reference/com/google/android/gms/wearable/Node>, “Node android wear api.”

-
- [62] “THE SOCIABILITY SCORE: App-Based Social Profiling of Students from a Health-care Perspective,” 2014.
- [63] <https://www.textrequest.com/blog/many-texts-people-send-per-day/>, “Average number of sms per day accessed in 04/07/17.”
- [64] <http://www.pewinternet.org/2010/09/02/cell-phones-and-american-adults/>, “Average number of calls per day usa adults in 04/07/17.”
- [65] Z. Chen, M. Lin, F. Chen, N. Lane, G. Cardone, R. Wang, T. Li, Y. Chen, T. Choudhury, and A. Cambell, “Unobtrusive Sleep Monitoring using Smartphones,” in *Proceedings of the ICTs for improving Patients Rehabilitation Research Techniques*, 2013.
- [66] J.-K. Min, A. Doryab, J. Wiese, S. Amini, J. Zimmerman, and J. I. Hong, “Toss ‘n’ turn,” in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI ’14*, 2014.
- [67] <http://heatonresearch-site.s3-website-us-east-1.amazonaws.com/javadoc/encog-3.3/org/encog/neural/networks/BasicNetwork.html>, “Encog basicnetwork class.”
- [68] G. P. Services, “https://play.google.com/store/apps/details?id=com.google.android.gms&hl=pt_PT.”
- [69] M. web usage overtakes desktop for the first time, “<http://www.telegraph.co.uk/technology/2016/11/01/mobile-web-usage-overtakes-desktop-for-first-time/>.”
- [70] “<https://developer.android.com/guide/topics/permissions/requesting.html>.”

Appendices

A

Articles

WSNs in FIWARE – Towards the Development of People-centric Applications

Ngombo Armando^{1,3}, Duarte Raposo¹, Marcelo Fernandes¹, André Rodrigues^{1,2},
Jorge Sá Silva¹ and Fernando Boavida¹

¹ Centre for Informatics and Systems of the University of Coimbra (CISUC)
3030-290 Coimbra, Portugal

² Polytechnic Institute of Coimbra, ISCAC
3040-316 Coimbra, Portugal

³ Universidade Kimpa Vita, Escola Superior Politécnica do Uíge,
Uíge, Angola

{narmando, sasilva, boavida}@dei.uc.pt, {draposo, jmfernandes}@student.dei.uc.pt,
andre@iscac.pt

Abstract. Wireless Sensors Networks (WSNs) form the founding block of the Internet of Things (IoT) Sensing Layer. In IoT scenarios, programmers will make the most of IoT if they focus on the development of context-aware applications rather than managing low-level aspects of the network they run on. To this end, providing harmonized data models eases the development of applications. Focusing on crossbow-micaZ sensors, we present an approach to integrate a heterogeneous WSN platform into FIWARE, towards the development of people-centric applications.

Keywords: WSN, FIWARE, micaZ, Heterogeneity, People-IoT interactions, Context-aware application.

1 Introduction

IoT is an extension of computer networks and the Internet to connected devices labeled “things” [1]. Currently, there are more than 22 billion connected devices, and this figure should reach over 50 billion by 2020¹. The exact definition of IoT is still evolving since it was first proposed in 1999 [2]. However, the concept of IoT has rapidly grown to the idea of *Internet of All* where users are humans, machines, and devices, and the network conveys the sensed features of the physical world. In the end, the Internet will provide a larger scale computing system to carry any measurable state of the physical world like emotions, psychological states, physical states, and actions [1]. The neuronal infrastructure enabling both the acquisition and raw processing of the environment’s features is called WSN. For this reason, WSNs form the founding block of IoT/A Sensing Layer, which the core functions are in fact sense the environment and actuate on it [3].

¹ statista.com, Accessed at December 4th, 2016

A typical WSN is a set of wirelessly connected devices called sensor nodes. In general, sensor nodes are micro or nanosystems with power, storage, computational and communication limitations. However, some sensor nodes are personal communication devices with augmented hardware capabilities [4]. Sensor nodes form a network that once deployed commence sensing the environment and report its features to a central node called sink. The sink aggregates the values communicated by the sensor nodes and forwards the packets to a management station. Once in management station, the sensor data is processed to produce information about the state of the target environment [5]. WSNs have application in both monitoring and tracking activities [6]–[9].

The natural heterogeneity of hardware, software, and protocols technologies under is a critical factor to enhance both resilience and lifetime of WSNs [3], [10]. To this end, heterogeneity must be driven by common specifications so that sensor nodes will be one computing platform at the disposal of smart applications developers. Indeed, there is a diversity of solutions for designing and implementing a WSN which is not always easy for stakeholders to choose and integrate them. These equations become more complex in cases users are not in touch but need each other to mount a virtual common infrastructure [3], [11]. Among the specifications providing standardized API (Application Programming Interface) and harmonized data models there is a European solution labeled FIWARE.

Our work aims at presenting the integration of WSNs in FIWARE, towards the development of people-centric, smart applications. The rest of the article is organized as follows. In next section, an in-depth presentation of FIWARE will be done. In Section III we will discuss the main object of this work, which is the integration of a heterogeneous WSN platform WSN in FIWARE, towards the development of people-centric, smart applications. We will conclude our work in Section IV, by tackling some open issues and challenges.

2 The FIWARE Platform

FIWARE or FI-Ware (where *FI* stands for Future Internet, and the prefix *Ware* stands as analogy referring to a kind of software) is a European set of specifications to provide APIs that ease the development of Smart Applications in multiple vertical sectors. FIWARE is open, public and royalty-free².

2.1. Background

The FIWARE project started in 2011 within the Seventh Framework Programme (FP7) of the European Commission as part the Future Internet Public-Private Partnership Programme (FI-PPP). On the one hand, the primary goal of FI-PPP was to advance a process for a harmonized European technology platforms and their implementation. On the other hand, the aim was to provide integration and harmonization of relevant

² fiware.org, accessed on 12/12/2016

policies frameworks. The process was divided into three phases. The first phase (from May 2011 to April 2014) was aimed at creating the technological core called FIWARE, while the second phase (from April 2013 to March 2015) was mainly aimed at the implementation of FIWARE nodes. Finally, the last phase covered from September 2014 to September 2016 and aimed primarily at creating a sustainable ecosystem for Small Medium Enterprises, through the selection of sixteen business accelerators³. The ecosystem of developers and entrepreneurs met for the first time at in Malaga, from 13th to 15th December 2016. This meeting was the first in a series of six-monthly events that will be organized by the FIWARE Foundation to surround all those who work to promote the adoption of the technology.

2.2. Architecture

FIWARE is based on a library of components called Generic Enablers (GEs) that are meant to implement APIs. GEs offer reusable and commonly shared functions “as a Service”. Through APIs, GEs allow developers to put into effect functionalities making programming much easier by combining them. GEs are classified into seven technical chapters⁴, namely:

- *Cloud Hosting*. A set of components to provide computation, storage and network resources on top of which services are provisioned and managed.
- *Data/Context Management*. A set of components to facilitate the access, gathering, processing, publication and analysis of context information at large scale. The enablers here also transform the data into valuable knowledge available to applications.
- *Architecture of Applications / Services Ecosystem and Delivery Framework*. A set of enablers to co-create, publish, cross-sell and consume applications or services, addressing all business aspects.)
- *Interface to Networks and Devices*. A set of components to make the most of the network infrastructure capabilities, building communication-efficient distributed applications, exploiting advanced network capabilities and easily managing robotic devices.)
- *Security*. A set of components to provide mechanisms to make delivery and usage of services trustworthy by meeting security and privacy requirements.
- *Internet of Things Services Enablement*. A set of components to leverage the ubiquity of different devices, making connected things available, searchable, accessible, and usable.
- *Advanced Web-based User Interface*. A set of components to facilitate the use of 3D and Augmented Reality capabilities in web-based user interfaces.

Based on the FIWARE for developers⁵ and [12], in Fig. 1 we display an instantiation of FIWARE architecture⁶ tailored to approach a generic IoT platform.

³ <https://www.fi-ppp.eu>, accessed on 12/12/2016

⁴ <https://catalogue.fiware.org/>, accessed on March 18th, 2017

⁵ <https://www.fiware.org/2015/03/27/build-your-own-iot-platform-with-fiware-enablers/>, accessed on December 17th, 2016.

⁶ https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_Architecture_R3, accessed on December 17th, 2016.

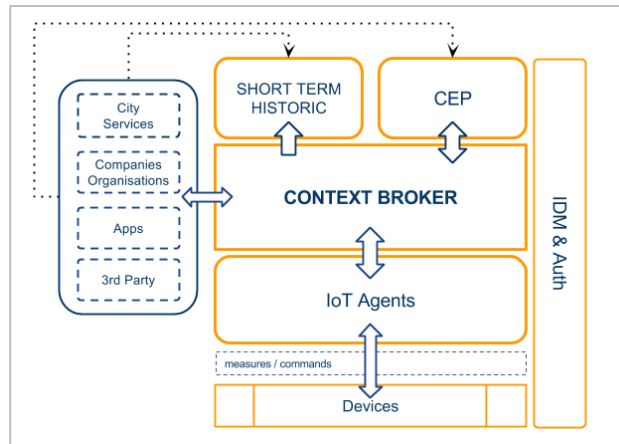


Fig. 1. The FIWARE IoT Platform.⁷

City services/companies & organizations /Apps / 3rd Parties are stakeholders that consume, transform and provide an end-utility to the data collected by the Devices.

The Devices refer mainly to the GE Chapter of *Interface to Networks and Devices*. They represent the network of sensors and actuators. Devices are context producers as they provide data to attributes and upload attributes for the entities. Here, a context producers are seen in a broad context where data can be collected through stationary sensors, mobile sensors, personal communication sensors, participatory sensing and Open Data Management Platform like CKAN (Comprehensive Knowledge Archive Network). Among the technical chapters, CKAN refers to *Data/Context Management*.

The IoT Agents refer to the GE Chapter of *Internet of Things Services Enablement*. They are the platform backend to manage Devices. That is, IoT Agents are gateways for the devices to communicate with the rest of the components in FIWARE. Further, IoT Agents act as translators between the communication protocols used by devices, and both a common language and data model across the FIWARE platform called NGSI (for Next Generation Service Interfaces).

The Context broker refers to the GE Chapter of *Data/Context Management*. It is the core and control piece of the platform, in charge of providing virtual representations of devices by developing a data/context scenario.

The Short Term Historic refers to the GE Chapter of *Cloud Hosting*. It is a database to manage the on-the-fly information and quick computational operations by third applications. These operations are performed using a time series database without an intermediate step to process the whole set of data to obtain the desired information. For instance, calculate the minimum, maximum, mean, bias or deviation of a set of data.

The CEP (Complex Event Processing) refers to the GE Chapter of *Data/Context Management*. It is intended to analyze and react to events, generating an immediate response and triggering actions due to changing conditions. It makes possible instant

⁷ <https://www.fiware.org/tag/iot/>, accessed on March 20th, 2017

response or reaction to a series of events occurring within a time window (SMS delivery and electronic mail notifications). CEP can be considered as the way to create new data from the arriving data.

Both *IdM* (The Identity Manager) & *Auth* (for authentication) are related to the mechanisms that enable privacy, security, and dependability in delivering services. Security is specified to be a transversal chapter to other chapters. IdM and Auth refer to the GE Chapter of *Security* in charge of carrying the information about users, roles, profiles, sending and validating tokens, and authentication mechanisms. Enablers to identity management and access control policies are complemented by two other elements which are: A Policy Enforcement Point (PEP) and PAP/PDP (for Policy Administration Point / Policy Decision Point).

Some FIWARE-based applications can be found in [13], and [14]. In the next Section, we present an approach to integrate a heterogeneous WSN platform in FIWARE, towards the development of people-centric smart applications.

3 The Integration of WSNs in FIWARE

3.1. The SOCIALITE Project

SOCIALITE⁸ (for Social-Oriented Internet of Things Architecture, Solutions, and Environment) is an ongoing project within the Centre for Informatics and Systems of the University of Coimbra (CISUC). The main objectives of the project are:

- Define a first generic version of an IoT people-centric architecture by developing and exploring a generic Cyber-Physical System / IoT architecture that can be used to support both, People2People interaction and People2Thing interaction.
- Explore the developed middleware and services in people-centric context, with the aim of demonstrating their use in enhancing the autonomy and quality of life of citizens, by identifying the needs and means towards general monitoring, and interaction with several user types.

Among the groups involved in SOCIALITE, the Laboratory of Communications and Telematics (LCT) is responsible for the design and the implementation of the network infrastructure on top of which, people-centric applications will be developed. Concerning the developed middleware in people-centric context, the LCT chose to align its solution to FIWARE specifications. To this end, the group has been developing a pilot scenario to infer the academic success of the students by monitoring their lifestyle. The application to be developed is called *ISABELA*⁹ (*for Iot Student Advisor & BEst Lifestyle Analyzer*). Since the LCT group works on the infrastructure to feed the database, the data processing, Machine Learning and Artificial Intelligence is out of the scope of this paper.

⁸ <https://www.cisuc.uc.pt/projects/show/215>

⁹ Due to communication concern, the name *ISABELA* was recently adopted for the previous *HappyStudent*.

Based on FIWARE generic platform, the tailored IoT scenario developed by LCT is depicted in Fig. 2.

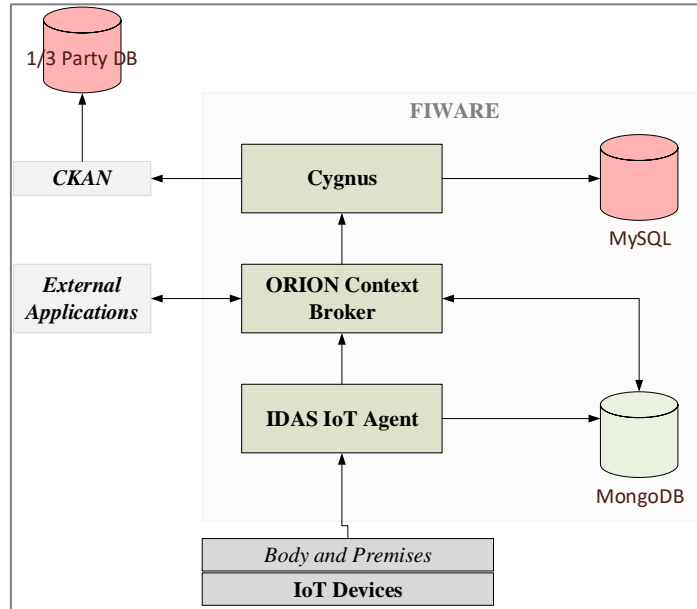


Fig. 2. ISABELA IoT Scenario platform. (adapted from [15])

In FIWARE, the context information is represented by values assigned to attributes that characterize entities in the application. Hence, for this pilot entities have been created to describe the student’s lifestyle environment. Some of the entities are a Student, its Rooms, the premises and entertainment facilities they use. Based on the inputs uploaded by sensors, through Machine Learning and Artificial Intelligence algorithms, it would be possible to infer the conditions that characterize the academic success of the students.

Even though the aim is to deploy sensors in the student’s environments, the current state of the pilot scenario is only performed in a testbed. We already provide real-time sensed data from both mobile and stationary sensors, integrated by the FIWARE runtime. All sensed data are stored in a MySQL and “CKAN”¹⁰ databases. These data are exploited to feed the primary version of ISABELA application allowing to simulate some aspects of the student’s environments. The setup for the current version of the application can download here (link). The values displayed in the current version of the application correspond to those uploaded by the remote sensors placed in the testbed i.e. in LCT Laboratory. In the following Subsection, we will describe the general architecture of the project and will better understand the limitations on the current version to analyze one’s Lifestyle.

¹⁰ <http://socialiteckan.dei.uc.pt/dataset/socialite>

3.1.1. General Network Architecture

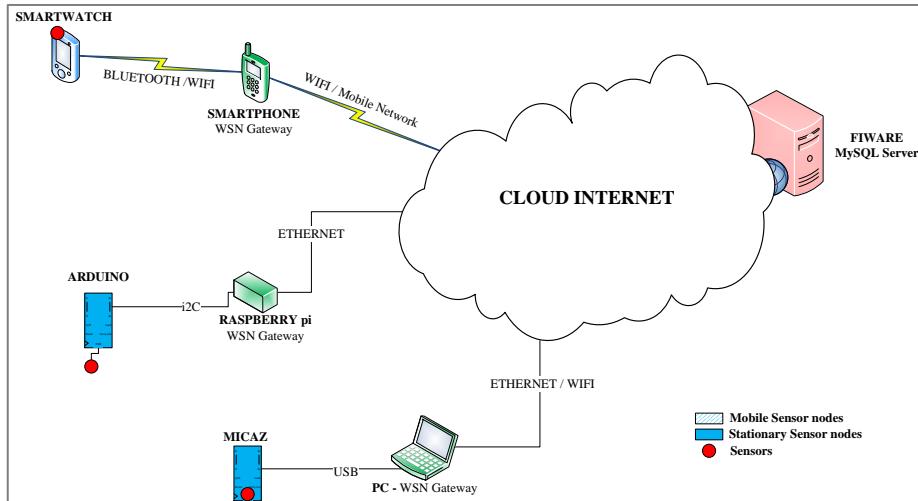


Fig. 3. LCT FIWARE Pilot network

The physical architecture of the network for the pilot integrated heterogeneous devices for data acquisition. As illustrated in Fig. 3, the sensor devices are mainly divided into two groups. The first group is set by mobile devices to ensure the student activities monitoring. The mobile devices are also used for the IoT-Student interaction through the smart application interface. The mobile devices form a mobile WSN composed by smartphones accoupled to Motorola Moto 360 smartwatches. This WSN is deployed to enable the acquisition of the student's activities e.g.: heart rate, communications counts, steps and location. Both smartphone and smartwatch run in Android environment. Hence, the applications in this WSN are developed in Android Studio IDE and uploaded to the watch via the smartphone. The immobile devices are intended to ensure the monitoring of the premises where students are supposed to develop some activities or rest such as classrooms or their personal rooms. Hence, the immobile devices form a stationary WSN currently composed by Arduino Uno sensor nodes and soon micaZ sensor nodes will integrate the stationary WSN. The Arduino's set uses a Raspberry Pi as gateway to the Internet and enables to acquire the temperature, the humidity and the noise level of the premises. However, additional data can be acquired depending on the nature of the transducers connected to the Arduino sensor node. The libraries on the Arduino are developed in C language while the requests in the Raspberry Pi were are made using command lines in Python. In Arduino, the application can be developed using other programming languages such as Java and C. Even though micaZ nodes offer expansion for a myriad of sensing activities¹¹, here it is only intended to the acquisition of both temperature and luminosity.

¹¹ <http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/MPR2400.html>

The dataset from the current testbed can be accessed at <http://socialiteckan.dei.uc.pt/dataset/socialite> (The *student resources* in the page correspond to data uploaded by mobile sensors).

The development environment of micaZ nodes is nesC-TinyOS and in the next subsection, we will be focusing its integration in the FIWARE within the scope of SOCIALITE project. The general integration of the three WSN in FIWARE towards the development of *ISABELA* is to be done according to the following steps [15]:

1. Creation IoT entities using Orion Context Broker;
2. Creation and registration of IoT devices in IDAS;
3. Sending of sensor data using the UltraLight IoT Agent;
4. Storage Persistent Data in a MySQL Database using Cygnus.

The broker enables to contextualize the information so that applications can interact with “things” instead of sensors. IDAS is the module in charge of handling all IoT devices, and Ultralight2.0 is the protocol used to provide IoT-Agents. To implement the communication client-server, HTTP (HyperText Transfer Protocol) and RESTful APIs are used for requests. It means, REST (Representational State Transfer) architectural style operations are adopted to reach resources (entities and attributes). JSON (JavaScript Object Notation) representation is the adopted format for the Payload. The on-the-fly data is stored in MongoDB database while Persistent data storage is done in a MySQL database through Cygnus Container. MongoDB is a snapshot database for the case of an interruption /DOS happens to Orion Context broker. The technical step-by-step guidelines can be found in [15], and all details concerning the API structures, experiments, and hands-on work can be found in *fiware.org* developers guide. As a case study, we will focus on the integration of micaZ in the FIWARE by describing a simple setup to upload sensor data.

In the next subsection, we will present the step-by-step guide to create an entity and send data directly to Orion context broker. Since the aim is only an introduction to some tools to enable the integration of WSNs built in micaZ, we will not tackle the creation of subscription to external description neither the store persistent data.

3.1.2. Focus on micaZ

The technical specification of micaZ sensor board is shown in Table 1 and the setup for our case study is illustrated in Fig. 4.

Table 1. Specifications of the MicaZ - MPR2400 [16]

MCU	Atmega 128L - Up to 16MIPS Throughput at 16MHz
RAM	4 KBytes
EPROM	4 KBytes
Flash memory	128 KBytes
Radio module	CC2420 – 2.4 to 2.48 GHz IEEE 802.15.4, AES-128
Transmission rate	250 Kbps
Extensions	Analog I, Digital I/O, I2C, SPI, UART

In the Laptop, we installed a turnkey Xubuntu OS via VM VirtualBox™. The turnkey solution includes a Linux Ubuntu distribution and the package TinyOS, libraries, and nesC compiler. We used a second Virtual Machine player (VMware Workstation 12™) for the FIWARE runtime machine. Given some technical issues, we had to install and run the Java playground in the OS of the Physical laptop (MS Windows™). Besides having knowledge of programming languages used in the case study, the training in [15] is a prerequisite to succeed the next instructions.

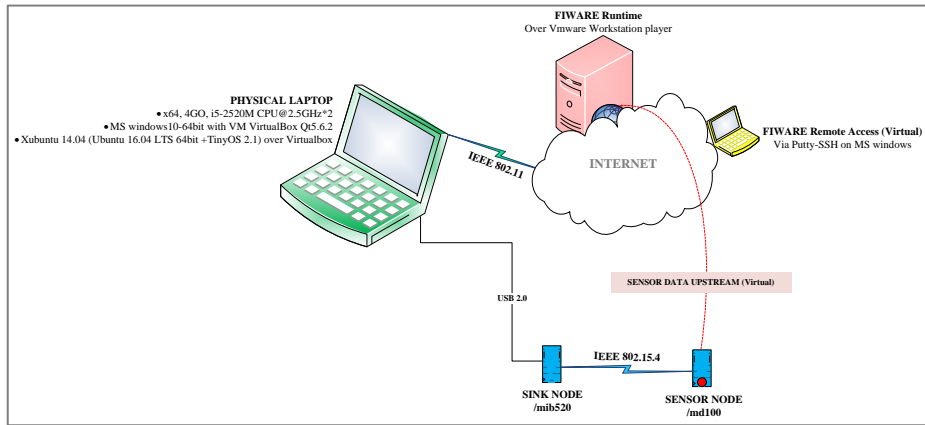


Fig. 4. The case study setup overview

The hands-on work to integrate MicaZ in FIWARE is done as follows:

- Step 1* - In TinyOS, write a sensing application, then couple the MicaZ hardware to the mda100 sensor board and the mib520 programming board. Finally, connect the whole set to a USB port of the Physical laptop and install the application in the sensor node using the command `~/Oscilloscope$ SENSORBOARD=basicsb make micaZ install,1 mib510,/dev/ttyUSB0`. Remark the specification of the sensor board at the beginning of the installation command. This is due to TinyOS protocol regarding the compilation of programs that run on the sensor nodes. As sensing application, we used the generic *Oscilloscope* program from TinyOS libraries (`~apps/Oscilloscope`). Disconnect the sensor node from the programming board and connect the Sink node.
- Step 2* - Staying in TinyOS, write the dispatcher program, then connect the Sink node to the mib510 board. Finally, install the dispatcher in the Sink node using the command `make micaZ install,0 mib510,/dev/ttyUSB0`. Do not disconnect the whole set (sink and programming board) from the Laptop. Remark that we changed the `TOS_ID_NODE` (node identification number) to 0. By convention, the number 0 is reserved to identify the Sink node in a WSN. As dispatcher program, we used the generic *BaseStation* program from TinyOS libraries (`~apps/BaseStation`).
- Step 3* - In TinyOS, write a program to fetch sensor data from the dispatcher and store it in a text file. We used a modified *seriallisten* program from TinyOS libraries (`~support/sdk/c/sf`). By default, *seriallisten* enables to displays the results on the terminal screen. We edited the *seriallisten.c* file and inserted a function to store on-the-fly, the last sensor data in a text file. To simplify, we only retrieved and

stored the last Byte of the sensed packets. In our case study, this Byte represents a value of the sensor data.

Step 4 - Write a middleware program to upload to sensor data to FIWARE database, from the text file created in the previous step. The program should enable to create an entity, create a service, register the device and send the sensor data. Our solution was developed in Java language.

Step 5 - Launch the FIWARE Virtual Machine, then run its containers.

Step 6 - Open a SSH session in puTTY and connect remotely to FIWARE.

Step 7 - Connect the Physical laptop to the Internet. This step serves to simulate the uploading of the sensor data to FIWARE via the Internet backbone.

Step 8 - Start the sensor node. It will run the application to collect and report the sensor data to the Sink. Using the generic application aforementioned, the green LED start blinking in both nodes.

Step 9 - Return to TinyOS and run the program to fetch sensor data from the dispatcher. Stop the application after retrieving at least one packet. If using the example in step 3, open a Terminal and run the modified *seriallisten* program with the command `~/c/sf$./seriallisten /dev/ttyUSB1 micaZ`. Remark that now we use the USB Port number 1. In fact, there are two ports associated with the mib520 programming board namely: port USB0 for configuration of the sensor nodes, e.g. to install applications in the nodes and port USB1 for the Application layer for instance to retrieve data from the sensor nodes [16].

Step 10 - Run the middleware program. In our case, the IDE is installed in MS Windows environment and the text file produced by *seriallisten* is a shared file to both MS Windows and Ubuntu over VM. If both the entity and its last attribute value have been uploaded to Orion context broker with success, a HTTP code is received in the IDE workstation added to the sensor value. Typical a 201 means the request has been fulfilled and the entity was created. A 20x code is a standard response for successful request. For our case study, we will assume that any other code translates an unsuccessful request.

Step 11 - Ultimately, return to the SSH session and verify the existence of the entity created with the value uploaded.

The setup is subject to several optimizations in both hardware and software environment. For the former, the Physical laptop can be replaced by a smaller computing system like Arduino, to make the setting more practical in deployment scenario. For the latter, one Virtual Machine is enough to run all the virtual systems in the Physical laptop. Moreover, the WSN runtime can be simplified by developing both the retrieving and the middleware program in only one language (Java or C). Indeed, both languages enable to develop the mechanism to fetch the sensor data from the dispatcher, then format it before uploading dynamically to the FIWARE.

4 Conclusion

Technological heterogeneity in WSNs enlarges the robustness for data acquisition and provision towards IoT applications. Dealing with diversity in both hardware and software solutions, as well as guaranteeing their transparency for the development of people-centric smart applications is a permanent issue. Smart applications are context-

aware applications. To this end, FIWARE offers powerful tools for having standardized API and harmonized data models for the development of applications, allowing People-IoT interactions. One of the main challenges of FIWARE is its consolidation as the open-source standard for IoT-enabled applications. Providing easy interoperability APIs to other IoT architectures such as OpenIoT, CitySDK, and IES Cities [17], may be an important step towards an international reference standard. On top of the services provided by WSNs and FIWARE, the SOCIALITE project aims at developing a people-centric, context-aware, cognitive framework that will ease the development of applications and, at the same time, bridge the gap between technology and its users.

The SOCIALITE Project began in July 2016 and is planned to end by 2019. The current paper focus on a technical aspect about how FIWARE can be leveraged to integrate an old sensor technology in a larger IoT computing system, towards the development a people-centric application. In future papers, more details describing the integration of all sensors deployed in the project will be done, as well as the evolution of functions in the smart application *ISABELA*.

5 Acknowledgments

The work presented in this paper was partly carried out in the scope of the SOCIALITE Project (PTDC/EEI-SCR/2072/2014), co-financed by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and Internationalization (POCI), European Union's ERDF (European Regional Development Fund), and the Portuguese Foundation for Science and Technology (FCT).

References

- [1] D. Nunes, P. Zhang, and J. Silva, "A survey on Human-in-the-Loop applications towards an Internet of All," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 1–1, 2015.
- [2] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, 2015.
- [3] S. M. A. Oteafy and H. Hassanein, "Resilient IoT Architectures over Dynamic Sensor Networks with Adaptive Components," *IEEE Internet Things J.*, vol. XX, no. X, pp. 1–1, 2016.
- [4] G. Savithri, P. Sujathamma, S. Padmavati, and M. Visvavidyalayam, "Android in WSN Applications – A Survey," *Int. J. Emerg. Technol. Comput. Appl. Sci. (IJETCAS)*, no. 1961, pp. 329–333, 2013.
- [5] W. S. and Y. S. and E. C. I. F. Akyildiz, "Wireless Sensor Networks : A Survey," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 38, pp. 393–422, 2002.
- [6] D. Wu, B. Yang, H. Wang, D. Wu, and R. Wang, "An Energy-efficient Data Forwarding Strategy for Heterogeneous WBANs," *IEEE Access*, vol. 4, pp. 1–1, 2016.
- [7] M. Rawidean, M. Kassim, and A. N. Harun, "Applications of WSN in Agricultural Environment Monitoring Systems," 2016.
- [8] T. O'donovan *et al.*, "The GINSENG system for wireless monitoring and control: Design and deployment experiences," *ACM Trans. Sens. Networks*, vol. 10, no. 1, p. 4,

- 2013.
- [9] M. A. Hussain, P. Khan, and Kwak kyung Sup, "WSN Research Activities for Military Application," *Adv. Commun. Technol. 2009. ICACT 2009. 11th Int. Conf.*, vol. 1, pp. 271–274, 2009.
 - [10] A. Giorgetti *et al.*, "A Robust Wireless Sensor Network for Landslide Risk Analysis : System Design , Deployment , and Field Testing," *IEEE Sens. J.*, vol. 16, no. 16, pp. 6374–6386, 2016.
 - [11] H. A. A. Figueira, D. Nunes, R. Barbosa, A. Reis, C. H. S. Sinche, A. Rodrigues, V. Pereira, H. Dias, and F. B. D. Raposo, J. Sá Silva, "WeDoCare: A Humanitarian People-centric Cyber-Physical System for the benefit of Refugees," in *IEEE Global Humanitarian Technology Conference (GHTC)*, 2016.
 - [12] K. Nájera, "Arquitectura básica para FIWARE," *Mooc INFOTEC*, 2016. [Online]. Available: <https://www.youtube.com/watch?v=j1XCjgeAW2c&t=62s>. [Accessed: 12-Dec-2016].
 - [13] M. Fazio, A. Celesti, F. G. Marquez, A. Glikson, and M. Villari, "Exploiting the FIWARE cloud platform to develop a remote patient monitoring system," *Proc. - IEEE Symp. Comput. Commun.*, vol. 2016–Febru, pp. 264–270, 2016.
 - [14] F. Ramparany, F. G. Marquez, J. Soriano, and T. Elsaleh, "Handling smart environment devices, data and services at the semantic level with the FI-WARE core platform," *Proc. - 2014 IEEE Int. Conf. Big Data, IEEE Big Data 2014*, pp. 14–20, 2015.
 - [15] D. Raposo, "Fiware basic operations." LCTSens, DEI-Universidade de Coimbra, Coimbra, 2016.
 - [16] F. B. Jorge Sá Silva, Ricardo Mendão Silva, *Redes de Sensores Sem Fios*, 1st ed. Lisboa: 978-972-722-830-0, 2016.
 - [17] M. S. Stefanos Vatsikas, Georgios Kalogridis, Tim Lewis, "The Experience of Using the IES Cities Citizen-centric IoT Platform," *IEEE Commun. Mag. Manuscr.*, 2016.

A Practical Assessment of Redundancy Models for IoT Reliability

Soraya Sinche, *Senior Member, IEEE*, Oswaldo Polo, Duarte Raposo, Marcelo Fernandes, Fernando Boavida, *Senior Member, IEEE*, André Rodrigues, and Jorge Sá Silva, *Senior Member, IEEE*

Abstract— Use of wireless sensor networks (WSNs) in conjunction with mobile devices is growing day by day. Mobile phones are not only personal devices that provide communication services, but also versatile and resourceful elements in the Internet of Things (IoT). Nevertheless, despite the referred growth, one key aspect for real-world IoT deployments is still largely overlooked: reliability. In this respect, there is a clear gap between theoretical models and real implementations, as no practical reliability mechanisms are in place for IoT. Proposing, studying, and providing IoT reliability solutions is, thus, fundamental for the massive deployment of IoT technology across all sectors of society.

In this context, the main purpose of this paper is to propose, explore, and assess reliability mechanisms for IoT scenarios, based on the use of redundant routes and devices that take advantage of mobile devices, as key elements of IoT. These mechanisms were evaluated in a variety of failure scenarios constructed with real, off-the-shelf equipment, showing their good capability to maintain effective communication even in the presence of link failures.

Index Terms— Mobile phone sensing, Wireless sensor networks, Reliability, FogPhone, Internet of Things.

I. INTRODUCTION

THE number of mobile devices is growing at a rapid pace. According to 4G Americas, in September 2016, there were 7.5-billion mobile subscriptions in the world [1]. In the near future, the number of connected mobile devices will be greater than the number of people on Earth. In fact, mobile devices, including mobile phones, are becoming an essential part of the 21st century Internet, and this trend will continue with the expected near-future roll-out of 5G.

Manuscript received April 29, 2017 This work was partially carried out in the scope of SOCIALITE Project (PTDC/EEI-SCR/2072/2014), co-financed by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and Internationalization (POCI), European Union's ERDF (European Regional Development Fund), and the Portuguese Foundation for Science and Technology (FCT), as well by SENESCYT – Secretaría Nacional de Educación Superior, Ciencia Tecnología e Innovación de Ecuador and by the Escuela Politécnica Nacional de Ecuador

The authors are with the Centre of Informatics and Systems of the University of Coimbra, 3000-214 Coimbra, Portugal, (e-mail: smaita@dei.uc.pt, oswado@dei.uc.pt, draposo@student.dei.uc.pt, marcelo14fernandes@hotmail.com, boavida@dei.uc.pt, arod@dei.uc.pt, sasilva@dei.uc.pt). S. Sinche is also with the Department of Electronic, Telecommunications and Networks, Escuela Politécnica Nacional, 170525 Quito, Ecuador (e-mail: soraya.sinche@epn.edu.ec) and A. Rodrigues is also with Polytechnic Institute of Coimbra, ISCAC, Coimbra, Portugal (e-mail: andre@iscac.pt).

Research and technology are now oriented to providing connectivity to people, things, and applications through the Internet. The Internet of Things (IoT) enables the development of all kinds of intelligent applications and services, combining both real-world and virtual world data. According to the World Economic Forum (WEF), IoT devices will grow from 22.9-billion in 2016 to 50.1-billion by 2020¹.

The proliferation of connected smart things or devices through the Internet for a variety of applications brings several challenges. When a number of smart things are involved, reliably sharing information and making collaborative decisions in critical applications, such as e-healthcare, home automation, or industrial automation, is crucial [2].

There are some pieces of work where the concept of reliability in IoT is treated. Kempf et. al. [3] discuss the role of reliability in the IoT, and provide some architectural guidelines to address reliability issues. Prasad et. al [2] present several approaches to reliability in IoT and discuss potential energy efficiency and reliability (EER) issues. They concluded that in the next generation of IoT, real-time monitoring applications are required and EER must be satisfied. Nevertheless, [2,3] just present requirements and make general considerations that allow improving reliability in IoT communications.

Maalel et. al [4] present a study of reliability for emergency applications in IoT, where they define an Adaptive Joint Protocol based on Implicit ACK (AJIA) for packet loss recovery and route quality evaluation. However, the solution is presented at the theoretical level only, without performing an experimental validation.

Zhu Q. [5] implements an extended SCALE (Safe Community Awareness and Alerting Network) platform as a testbed, and presents the key challenges and approaches in creating a reliable IoT community. SCALE is in an experimental development stage and the author does not provide an analysis of its reliability mechanisms.

On the other hand, there are already some pieces of work about designing and modelling IoT reliability. For example, Behera et. al. [6] propose a novel reliability-modeling scheme for a service-oriented IoT setup. The article is focused on service reliability of IoT, and defines a case study with two subsystems. Each subsystem receives temperature and smoke information in real time from several sensors, and when their

¹ <https://www.weforum.org/agenda/2015/11/is-this-future-of-the-internet-of-things/>

values are over a given threshold an actuator and alarm are activated. However, the system is not studied in failure scenarios.

Li et. al [7] present an IoT reliability evaluation model that considers perception reliability, transmission reliability, and processing reliability. Moreover, Ahmand et. al. [8] present a methodology for estimating hardware and software reliability. Nevertheless, these pieces of work are oriented to the modeling, evaluation, and estimation of IoT reliability, not to providing it. Despite this, they constitute a good basis for the ideas proposed in the current paper.

Although there are several theoretical studies on IoT reliability, like those presented in the previous paragraphs, they do not provide IoT reliability tests in real scenarios, with practical solutions. In the current paper, we propose and assess, in practical settings, reliability mechanisms for IoT systems, using mobile devices as key elements of the reliability architecture. In our work, we took an experimental approach, and considered the average Round Trip Time (RTT) as metric. This is defined as the average period between issuing a sensor data request and receiving the respective data. In a communications network, the RTT is one of several factors affecting latency. For this reason, RTT is important for identifying the efficacy of recovery mechanisms during failures.

The contributions of this paper are the following:

- 1) a model for reliability in IoT networks, based on device and link redundancy;
- 2) implementation of the proposed model with off-the-shelf equipment, thus proving its viability;
- 3) use of mobile devices, including mobile phones, as elements of the reliability strategy, taking advantage of the fog-of-things concept;
- 4) practical assessment of the proposed reliability model, by performing a set of tests under a variety of faults.

The remainder of the paper is structured as follows. In section II we provide background concepts, firstly explain the role of the Fog of Things in IoT, then describing the potential of personal mobile devices, especially mobile phones, as fog-enabling systems, and lastly addressing key concepts in IoT reliability. In section III, we present the proposed model for IoT reliability. This model will be subject to practical evaluation in a set of case studies that are presented in section IV. Section V presents and discusses the experimental evaluation results. Conclusions and guidelines for further work are provided in section VI.

II. BACKGROUND

A. Fog of Things

Cloud computing is responsible for the biggest change in the way people work, live and, mainly, do business in recent years. It involves changing the way technology is used from an ownership paradigm to a subscription based and pay-per-use methodology. Based on three main service levels – infrastructure, platform, and software – it leverages the know-

how on clustering and virtualization to provide elastic computing and storage services with high-availability assurance. In fact, cloud computing is nowadays the preferable platform for many business solutions. It requires low initial investment, resources are elastic, and services are quickly deployed and available. Organizations can focus on the business model, leaving the management of infrastructures, platforms, and software at the cloud provider's side.

However, today's devices, which make up IoT and Cyber-Physical Systems (CPS), are becoming increasingly more heterogeneous, mobile, and intelligent. In theory, everything from lightbulbs to fridges, microwaves, and coffee machines will soon be able to connect to the Internet. These devices represent an untapped resource that is available on site. So, it no longer makes sense to rely on distant, cloud-based service providers when we wish to communicate with neighbors for handling local tasks and information. Thus, an efficient cloud should come down to the network edge and become diffused among client devices, in both mobile and wired networks, whenever necessary. This means that the traditional cloud architectures are now migrating to the network edge and becoming a Fog of Things (FoT).

To date, to the best of our knowledge, no study has delved into investigating the integration of WSNs and mobile phone systems within the fog, with the aim of exploring IoT reliability, as this requires significant conceptual changes to the traditional cloud architecture, endowing it with new functionalities based on a fog paradigm. As far as extending the cloud to IoT and CPSs is concerned, new advancements in the state-of-the-art need to be considered. Most of the current sensing techniques should be investigated and adapted to the highly distributed and heterogenic environment of these embedded systems. The outputs of these techniques also need to "close the loop" and affect the way embedded systems collaborate, in order to achieve dynamic levels of security/privacy, reliability, and higher system performance. All of these aspects are highly challenging and, as a consequence, innovative solutions have to be found by promoting the use of this new generation of cloud through virtual environments that facilitate the construction of new applications and products.

The goal of this distributed edge-based cloud is to quickly assist local nodes by locally maintaining and using data, avoiding the cost and latency of data transmission to a centralized cloud. In this context, the integration of WSN and mobile phone sensing leads to clear benefits, as processing and data storage tasks are performed in local devices.

B. Mobile phone as a Fog system

Advances in sensor technologies have led not only to an increase in WSN-based solutions, but also to a proliferation of mobile phones with native sensors and, as a result, to a set of new applications. This proliferation has profoundly changed what we believe will be the future of IoT, where WSNs will work together with mobile phones and other personal mobile devices to offer reliable applications. These will provide the research community with a massive flood of data that enables

systems to leverage sensed information on structures, humans, environment, energy, and resources. The work presented in the current paper represents a further step along this vision, by combining WSNs, IoT, mobile devices, and the fog concept, for providing IoT reliability.

There are several people-centric or environment-centric sensing applications implemented in mobile phone systems. In the work of Macias et. al. [9] several such applications are presented. Research points towards a future of WSNs increasingly integrated with mobile phone sensors. The next generation of Mobile Phone Sensing (MPS) should enable us to relate a person-centered application with the effects caused by human behavior in the environment and, on the other hand, to develop mobile applications that help improving quality of life. Thus, there is the need for integrating more external sensors that allow the monitoring of environments together with MPSs. This converts the traditional mobile phone into a fog-like device: The FogPhone.

Using the FogPhone approach, processing will be carried out by edge devices, and only selected, processed information will be sent to the cloud. In addition to improving efficiency and effectiveness, this opens a whole range of new applications that can be developed on mobile phones and that interact directly with WSNs.

For instance, by using activity recognition algorithms (e.g., algorithms that can identify walking, cycling or driving), it is possible to infer if a person uses public or private transport, or if he/she prefers walking, when going to a specific destination. Additionally, if information on the level of pollution on the roads, it is possible to advise and raise awareness in users, encouraging them to use environmentally-friendly transportation.

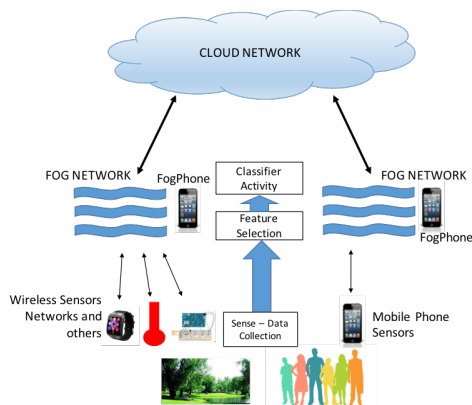


Fig. 1. Structure of Fog Network with a FogPhone

In Fig. 1, a general structure of a Fog Network using FogPhones is presented, where the data collected from a network of wireless sensors, a smart watch, or from the mobile phone sensors are processed in FogPhone systems. This processing can include feature selection and algorithms for activity recognition.

C. IoT reliability

In recent years, with the growth of IoT applications in many areas, reliability has gained in importance due to the impact

that failures can have on the performance of IoT networks and applications. In critical areas, the consequences could be disastrous. Faults in an IoT network may originate in the communication system, energy system, mechanical system, etc.

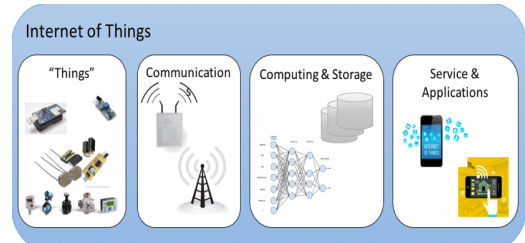


Fig. 2. General structure of IoT systems, based on [10]

IoT systems are made of several building blocks. Fig. 2 shows a general structure of an IoT system, in which there are four main blocks: “Things”, Communication, Computing & Storage, and Services & Applications. Our work is focused on increasing the reliability of the Communication block.

The term reliability can be used apply, for example in the case of application robustness, resistance to security problems, adaptability, self-configuration, long-term usability, or overall system reliability [3].

IoT network infrastructure reliability is crucial for the reliability of the whole system. For example, emergency applications require the reliable transmission of sensor data with minimum delay. Failure in delivering sensor data in a reliable and timely manner may end in high costs, user dissatisfaction, and physical damage to people or things².

There are two techniques typically used to achieve WSN reliability: retransmission and redundancy. Mahmood et al. [11] propose a three-dimensional reference model for WSN reliability (Fig. 3). In this reference model, reliability can be based on retransmission and/or redundancy, using hop-by-hop or end-to-end communication, and operating at packet level or event level.

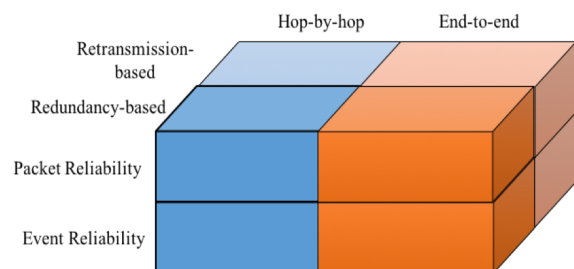


Fig. 3. Reference model for research in WSN reliability [11]

On the other hand, [12] presents some important techniques for improving network reliability, such as parts improvement method, effective and creative design, use of overrated components, and structural redundancy, in which the latter provides reliability through alternative routes.

² <http://iot.ieee.org/newsletter/january-2015/relyonit-dependability-for-the-internet-of-things.html>

The reliability model that we propose in the subsequent section is based on structural redundancy.

In an IoT system, a thing is a smart object where the data is collected through sensor nodes. The proper selection of sensors is crucial and it must be a function of the type of applications to be implemented. Furthermore, the system may combine information from several sensors, which may represent an opportunity to improve the reliability and accuracy of the data.

Many IoT applications use low-cost sensors, so that it is cheap to implement redundancy mechanisms. These mechanisms, accompanied by efficient routing protocols, allow us to improve data accuracy. Moreover, redundancy of devices and communication links can be explored to improve reliability of IoT systems.

After data acquisition, it is necessary to store the raw data for further processing. Here, it is also possible to explore redundancy with the objective of creating a reliable architecture. To this end, in a fog-based architecture, databases can exist both in fog systems (e.g., a FogPhone or set of FogPhones) and cloud servers. FogPhones can provide temporary data storage during network outages and, once communication is reestablished, send all data to a cloud server. Moreover, in addition to database redundancy and to temporary storage, it is possible to optimize communications, as not all data need to be stored centrally.

III. REDUNDANCY-BASED RELIABILITY MODELS FOR IOT

In this section, we propose several models for increasing IoT network reliability, based on combinations of communication links redundancy and devices redundancy.

The basic scenario we considered is composed by an IoT network where each sensor (thing) is connected to the central server using a gateway device and a network link. Fig. 4 shows a first reliability model that simply considers one backup link for each main link connecting each gateway to the server, i.e., a 1:1 redundancy.

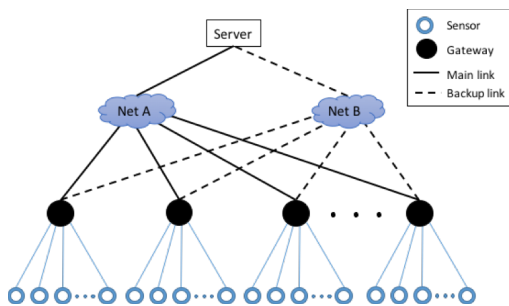


Fig. 4. Reliability model with alternative communication links

Additionally, we also considered including redundancy at the gateway level, because gateways are critical components of IoT networks. In this scenario, each gateway will have a backup gateway, working in a master - slave configuration. This is depicted in Fig. 5.

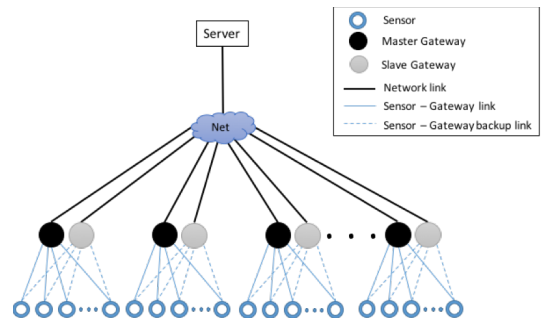


Fig. 5. Reliability model with redundancy in gateway devices

The two previous schemes can be combined into the model presented in Fig. 6, where reliability is improved by using both communication links redundancy and gateway redundancy.

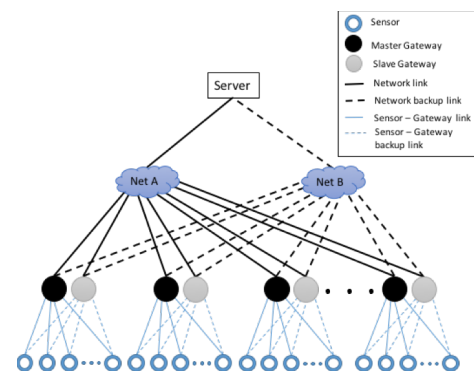


Fig. 6. IoT reliability model with dual redundancy (communications and devices)

The model presented in the Fig. 6 can be simplified in the case where sensors are integrated in gateways, which is quite a common case. The resulting model is shown in Fig. 7.

In the next section, we evaluate the presented models using several case studies. In the experiments, we assumed, without loss of generality, that communication between the sensing systems and the gateway uses a wired network interface, as this is usually the case in industrial environments.

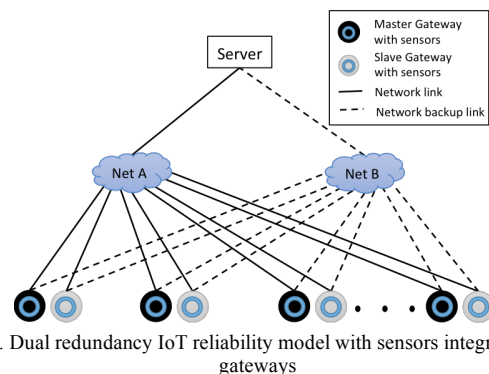


Fig. 7. Dual redundancy IoT reliability model with sensors integrated into gateways

IV. CASE STUDIES

A. Experimental Setup

We worked with an IoT network testbed whose structure is shown in the Fig. 8. Our system uses a personal computer as http server and database. These services are mounted over the FI-WARE³ platform. The Arduino Uno represents any embedded system that performs data acquisition, using

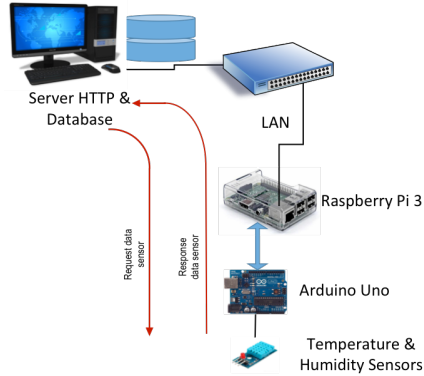


Fig. 8 Structure of the used IoT network testbed

sensors, but does not support wireless communications, thus requiring a gateway for data communication with the server.

In our tests, we worked with digital temperature and humidity sensors (specifically, DHT11 sensors). The sensors were connected to an Arduino Uno through digital inputs, using the DHTLIB library for sending data. The data was collected and transmitted to a Raspberry Pi 3, which worked as a gateway device. For this communication, we used the I²C bus protocol. Finally, the Raspberry Pi sent all the information collected from the sensors to the server through an Ethernet or Wi-Fi link.

For each sensor data reply received, a round trip time was reported. This was measured by the local clock of the computer, from the instant the request left the server to the instant when the reply arrived. This request was implemented with a Python program using TCP connections. We defined one socket for the main connection and other socket for a backup connection.

As the Raspberry Pi uses the Linux operating system, when a sudden power outage happens this can generate problems when restarting the system, thus leading to service unavailability. On the other hand, if there are problems in communication links these can also generate service unavailability. In this context, we implemented mechanisms that improved the reliability of the system, based on the already mentioned concept of redundancy. Specifically, we considered two approaches in our reliability case studies: device redundancy and link redundancy.

B. Device Redundancy

In this approach, we implemented and tested a scenario based on the model presented in Fig. 5. Two Raspberry Pi 3

worked as a cluster (Fig. 9). The cluster was implemented through the Raspberry Pi LAN ports. One Raspberry node worked as a master and the other worked as a slave.

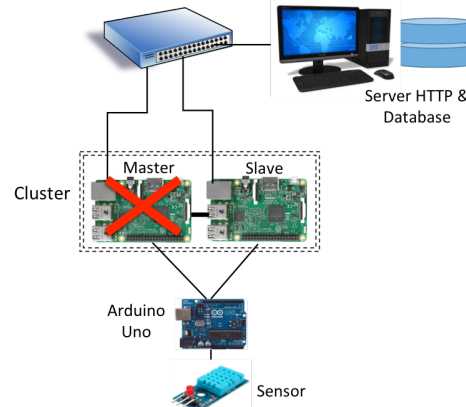


Fig. 9 Scenario (a): Device redundancy with a Raspberry Pi cluster

For implementing the scenario (a), the Arduino board was connected to both Raspberry Pi nodes, as shown in Fig. 10. A voltage level converter (BSS138) from 5V (Arduino) to 3.3 Volts (Raspberry Pi) was necessary for this connection.

In each Raspberry Pi, we installed the Pacemaker and Corosync packages to have cluster functionalities. Next, a Raspberry Pi was configured as master and the other as slave. When these systems are working as a cluster, heartbeat signals are exchanged to know the state of each cluster element (active or passive). Using these signals, a failure on the master during the transmission data process causes the automatic switching to the slave.

On the other hand, we installed a server and client program in Python between the PC and the Raspberry Pi cluster, and configured two sockets: an "active" socket with the master and a "standby" socket with the slave. In the client program, the PC continually sent data requests to the cluster over the "active" socket. When the master failed, this produced a TCP disconnection and its port was closed. The client program

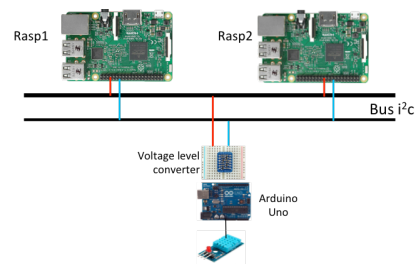


Fig. 10 Connection between the Raspberry Pi cluster and the Arduino & sensors.

detected this problem and reconnected over the "standby" socket. The round-trip time was continuously measured during the data transmission process, both during normal operation and during failures.

C. Link Redundancy

In what concerns link redundancy, we analyzed three scenarios based on the model presented in Fig. 6 and using the

³ FI-WARE is a project defined by an independent open community in the European Union. This project develops a complete platform for Future Internet and next generation services

cluster configuration of scenario (a).

Firstly, in scenario (b) the wired link of the master was disconnected and the system switched to the wired link of the slave (Fig. 11).

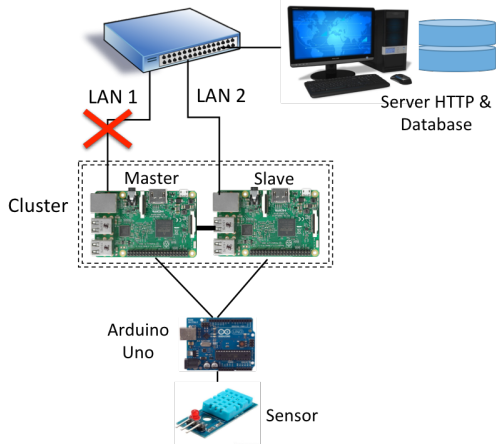


Fig. 11 Scenario (b): link redundancy using two wired links

Secondly, in scenario (c) the wired link of cluster was disconnected and the system used a wireless link (IEEE 802.11) through a FogPhone to send sensor data to the server (Fig. 12).

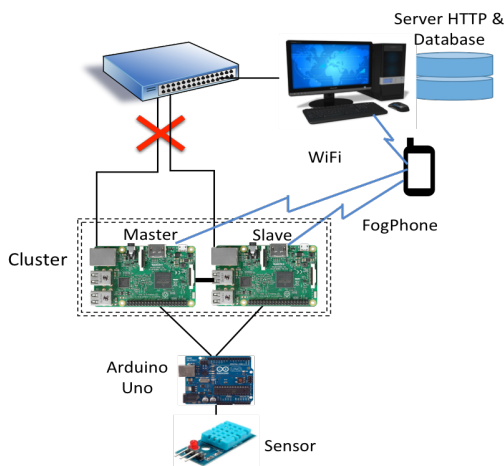


Fig. 12 Scenario (c): link redundancy using a FogPhone as hot spot

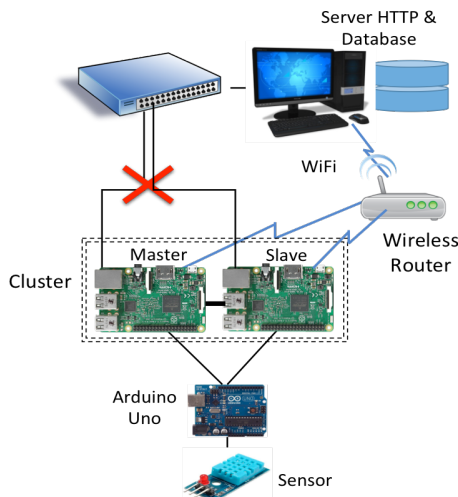


Fig. 13 Scenario (d): link redundancy using a wireless router

Finally, scenario (d) was similar to scenario (c), with the difference that it used a wireless router as a backup link (Fig. 13).

In these scenarios, a cable failure causes a sudden disconnection and the TCP port hangs, losing the connection and locking the socket during the default timeout. It is necessary to force its closure and to not wait for the expiration of the default timeout. For this, in the client program a flag was activated, after which the reconnection phase was started.

V. EXPERIMENTAL RESULTS

We measured the average RTT in normal conditions (i.e., no faults) and in each test scenario. The measurements were performed within intervals of 4 minutes, and repeated 10 times. In each test round, we collected over 110 RTT measurements, the exact number depending on the time it took to re-establish communication via the backup link.

Then, the average RTT for each scenario was calculated using a confidence level of 95%. Table I presents these results. It is apparent that in most cases the average RTT values suffer only a slight increase even though link or device failures exist, thus showing the effectiveness of the proposed reliability architecture.

Additionally, the last column in Table I presents the mean of the maximum RTT during the test period, in normal condition as in the four scenarios. We can see that link redundancy (i.e., scenarios (b), (c), and (d)) leads to considerable lower values of this parameter.

TABLE I
THE AVERAGE RTT IN NORMAL CONDITIONS AND THE SCENARIOS (A), (B), (C) AND (D).

	Average RTT (μ s)	Confidence Interval (μ s)	Average of max. RTT (μ s)
Normal conditions	394.518	104.8	1593.83
Scenario (a)	450.797	151.84	1669.59
Scenario (b)	401.643	94.02	639.16
Scenario (c)	410.894	91.43	584.80
Scenario (d)	399.610	89.19	800.25

Moreover, as we can see in the Table II, the scenario with the greatest increase in the average RTT when fault conditions are applied is scenario (a) with an increment of 14.27 %, while the scenario with the smallest average RTT was scenario (d), again pointing to the relatively better performance of link redundancy over device redundancy.

TABLE II
DELAY INCREASE RESPECT TO NORMAL CONDITIONS.

	Average RTT increase respect to normal conditions (us)	% of average RTT increase
Scenario (a)	56.28	14.27
Scenario (b)	7.12	1.58
Scenario (c)	16.38	4.08
Scenario (d)	5.09	1.24

The obtained results clearly point to some conclusions. On one hand, IoT device redundancy and/or IoT link redundancy for providing reliability are both feasible and effective. On the other hand, the performance price to pay is quite small. Last but not least, using devices such as FogPhones for providing link reliability does not significantly penalize the performance of the whole system, and can even pay-off if one realizes that these devices can provide local processing and local storage capabilities in addition to link redundancy

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed, implemented and assessed several models for IoT reliability. In addition to proposing these models, the paper presented their practical assessment, whose importance lies in the following issues: i) it targeted actual implementations of device and link redundancy scenarios; ii) model implementations were done with off-the-shelf equipment, thus proving their viability; iii) the fog-of-things strategy was used and validated; iv) several types of device and link faults were analyzed. The obtained results prove the feasibility and effectiveness of the proposed models.

As future work, we plan to evaluate the advantages of FogPhones used as backup for storage and processing of raw data, thus improving the reliability of IoT applications. Moreover, we also plan to explore the use of the Multipath Transport Control Protocol (MPTCP) as another mechanism for IoT reliability. Furthermore, we are also planning to explore reliability mechanisms that use management agents supported in IoT protocols (e.g., CoAP Management Interface (CoMI) or RESTCONF).

The development of Fog Phone systems is still young. Thus, there is a vast amount of unresolved issues and research challenges related to processing and storage capacity. To solve them, we also need to develop distributed algorithms that enable low-power consumption, and provide security mechanisms that avoid malicious attacks or theft of information.

Several challenges also arise from this distributed fog architecture that combines MPS and WSNs. Firstly, it is fundamental to setup the virtual infrastructure, namely to choose the distributed cloud supporting devices and to implement the distributed cloud services over heterogeneous devices, capable to transparently operate as one (building a tiny-virtual cluster). Secondly, it is necessary to distinguish and pre-process sensing data to determine which data will be kept in the local-distributed cloud and which data will be also sent to the centralized cloud. This pre-processing can be useful either to facilitate big data processing, by avoiding transmitting large amounts of raw data to a centralized data center, or to deliver local data faster, with higher availability, security and control, among other benefits.

ACKNOWLEDGMENT

The work presented in this paper was partially carried out in the scope of SOCIALITE Project (PTDC/EEL-SCR/2072/2014), co-financed by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and

Internationalization (POCI), European Union's ERDF (European Regional Development Fund), and the Portuguese Foundation for Science and Technology (FCT), as well by SENESCYT – Secretaría Nacional de Educación Superior, Ciencia Tecnología e Innovación de Ecuador and by the Escuela Politécnica Nacional de Ecuador..

REFERENCES

- [1] 4G Americas, "Cellular Technologies Enabling the Internet of Things", November 2015.
- [2] S. Prasad, Ch. Kumar, "A Green and Reliable Internet of Things", Communications and Network, Scientific Research 2013, 5, pp 44-48.
- [3] J. Kempf, J. Arkko, N. Beheshti, K. Yedavalli, "Thoughts on Reliability in the Internet of Things", www.iab.org, 2011.
- [4] N. Maalel, E. Natalizio, A. Bouabdallah, P. Roux, M. Kellil, "Reliability for emergency application in Internet of Things", IEEE International Conference on Distributed Computing in Sensor Systems, 2013.
- [5] Q. Zhu, "Enhancing Reliability of Community Internet of Things Deployments with Mobility", 34th International Symposium on Reliable Distributed System, September 18 – October 1, 2015, Montreal - Quebec, Canada.
- [6] B. Ranjit, K. Hermant, R. Diptendu, "Reliability Modelling of Service Oriented Internet of Things", Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2015 4th International Conference on Reliability.
- [7] Y. Li, L. Tian, "Comprehensive Evaluation Method of Reliability of Internet of Things", IEEE 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 262 – 266.
- [8] M. Ahamad, "Reliability Models for the Internet of Things: A Paradigm Shift", 2014 IEEE International Symposium on Software Reliability Engineering Workshops.
- [9] E. Macias, A. Suarez, J. Loret, "Mobile Sensing Systems, Open Access", Sensors 2013.
- [10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", IEEE Communication Survey & Tutorials, Vol 17, pp. 2347-2376, IEEE 2015.
- [11] M. Mahmood, W. Seah, I. Welch, "Reliability in wireless sensor networks: A survey and challenges ahead", Elsevier 2015.
- [12] S. Chaturvedi, "Network Reliability: Measures and Evaluation", Scrivener Publishing, Wiley, 2016.



Soraya Sinche received the Engineer degree in from the Escuela Politécnica Nacional of Electronic and Telecommunications, Ecuador, in 1999, the master degree from Politecnico di Torino of Wireless Systems and relative Technologies, Turin, Italy, in 2004 and the master degree from the Escuela Politécnica Nacional of Connectivity in Telecommunications Networks, Quito, Ecuador, in 2016. She is currently pursuing the Ph.D. degree at the Department of Informatics Engineering of the University of Coimbra, Portugal. Her current research interests include Internet of Things, wireless communications, wireless sensor networks and mobile phone sensing. Since 2000, she has been Lecture at the Department of Electronic, Telecommunications and Networks of Escuela Politécnica Nacional, Quito, Ecuador.

Mrs. Sinche is a Member of IEEE since 2004 and Senior Member of IEEE since 2015.



Oswaldo Polo received the degree in Electronic and Telecommunications from Escuela Politécnica Nacional, Quito, Ecuador, in 1999. He worked as a Platform Engineer in Telefonica (Ecuador) and a Support Engineer of Security Systems in Ebtel (Ecuador). He is currently researching at the Department of Informatics and Engineering of University of Coimbra, Portugal. His current research interests include Networking equipment (Cisco, Juniper, Huawei), Wireless sensor network, and Embedded Systems.



Duarte Raposo is an enthusiastic of wireless technologies, specifically low-power wireless sensor networks for industrial environments and the future Internet of Things. Duarte has received his Master's degree (2012) in computer engineering from University of Coimbra, Portugal. At present, he is a PHD student of doctoral program in information science and technology at University of Coimbra, Portugal and a research student of Centre for Informatics and Systems (CISUC). He's research topics are: Industrial Wireless Sensor Networks, network management and diagnostic tools for WSN (using machine learning techniques with logging tools). At the same time, Duarte has also been working in the industrial field, participating in several national and international projects as researcher, network advisor and designer, and software developer.



Marcelo Fernandes, was born in a small village near Viseu, Portugal. He is currently finishing his Master's thesis, to obtain his master's degree in Biomedical Engineering from the University of Coimbra, Portugal. In 2016, he become a temporary member of the CISUC-LCT group.



André Rodrigues has a B.Sc. in Informatics Engineering from the University of Coimbra (Portugal), an M.Sc. in Finance from ISCTE Business School, and a Ph.D. in Informatics Engineering from the University of Coimbra in 2013. He works as a teacher at the Polytechnic Institute of Coimbra, giving classes on networking. His main research interests are Wireless Sensor Networks and Internet of Things. He is the author of several papers in international journals and conferences in those areas. He has been serving as a reviewer in top conferences and participated in several European initiatives and projects. He is a researcher for the Laboratory of Communication and Telematics at the Centre of Informatics and Systems of the University of Coimbra. The

publication list is available at: www.researchgate.net/profile/Andre_Rodrigues9.



Fernando Boavida (www.uc.pt/go/boavida) received his PhD in Informatics Engineering in 1990, and he currently is Full Professor at the Department of Informatics Engineering (DEI) of the Faculty of Sciences and Technology of the University of Coimbra. His main research interests are people-centric Internet of Things, wireless sensor networks, mobility, and quality of service. He is author/co-author of more than 170 international publications (books, book chapters, refereed journals and conference proceedings) and 50 national publications. He was the chairman of the Program Committee of QofIS'2001, IDMS-PROMS'2002, NETWORKING 2006, WWIC 2007, FMN 2008, EWSN 2010, FMN 2012, IWQoS 2012, ACM SIGCOMM FhMN 2013, Mobiquitous 2015, and WoWMoM 2016 international conferences/ workshops. He is a senior member of the IEEE and a licensed Professional Engineer. He is a member of the Editorial Advisory Board of the Computer Communications journal.



Jorge Sá Silva received his PhD in Informatics Engineering in 2001 from the University of Coimbra, where is a Tenured Assistant Professor with Habilitation at the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra and a Senior Researcher of Laboratory of Communication and Telematics of Centre of Informatics Engineering of University of Coimbra, Portugal. His main research interests are Internet of Things, Network Protocols, Machine to Machine, and Wireless Sensor Networks. He has been serving as a reviewer and publishing in top conferences and journals in his expertise areas. His publications include 2 books, 5 book chapters and over 170 papers in refereed national and international conferences and magazines. He participated in European initiatives and projects such as FP5 E-NET, FP6 NoE E-NEXT, FP6 IP EuQoS, FP6 IP WEIRD and FP7 Ginseng (as Portuguese Leader). He actively participated in the organization of several international conferences and workshops, (e.g. he was the Workshop Chair of IFIP Networking2006, Publicity Chair of EWSN2009, General Co-Chair of EWSN2010, General Co-Chair of Mobiquitous2015, General Vice-Chair of WoWMoM2016) and he was also involved in program committees of national and international conferences. He is a senior member of IEEE, and he is a licensed Professional Engineer. His homepage is at <http://www.dei.uc.pt/~sasilva>.

B

List of permissions

On this appendix we present a list of all the permissions[70] used by the application as well as the explanation to why we use each one of them. We are aware that some of this permissions raise privacy issues and as such we believe that the transparency of why and how we used them is very important.

- **ACCESS_WIFI_STATE:** To get the list of the configured network the Wifi needs to be turned on. As such, when the user navigates to the Settings screen, we need to check if the Wifi is turned on or off.
- **CHANGE_WIFI_STATE:** For the same reason as above, we need to turn on the Wifi (if it is turn off) when the user navigates to Settings screen.
- **INTERNET:** This permission is necessary in order to access the Internet from the application. We need this permission to communicate with the FIWARE, through HTTP/REST requests.
- **ACCESS_NETWORK_STATE:** This permission allow us to check if the Smartphone is connected to any network. It also allows us to see the information about that connection, namely if is a connection through mobile data or through Wifi. This is necessary to see if we can make the requests to the FIWARE.
- **ACCESS_FINE_LOCATION:** This permission allows us to receive the location of the Smartphone, define by GPS or by the network.
- **READ_PHONE_STATE:** This permission allows us to set a listener of the phone state, that can be: ringing, off-hook or no activity. We need this permission in order to perform the call detection for incoming calls.
- **PROCESS_OUTGOING_CALLS:** This permission is necessary in order to create a listener for outgoing calls. We use this listener in order to perform

call detection for outgoing calls.

- **READ_SMS:** This permission gives access to the SMS log from the Smartphone. We only use this to detect the number of SMS and number of destination/origin. We do not check the content of the SMSs and the number is encrypted before we store it.
- **READ_CALENDAR:** This permission allows us to access the calendar log. We use this permission to check the students academic schedule and events.
- **ACTIVITY_RECOGNITION:** This permission is necessary in order to use the Google Activity Recognition API.
- **SET_ALARM:** This permission is used to set alarms on the Smartphone.
- **RECEIVE_BOOT_COMPLETED:** This permission allows us to implement a listener that lets us know when the Smartphone boot is completed. We use this to auto-start the application when the Smartphone boots.
- **RECORD_AUDIO:** This permission is necessary in order to use the microphone. We use this to measure the sound amplitude, to perform sleep detection. We do not store any audio or process it in any other way.
- **STORAGE:** This permission is necessary in order to store data on the Smartphone. This allows us to create the DB on the Smartphone.