Pedro Alexandre Afonso Simão

Bachelor of Computer Science and Engineering

IoT Platforms for Building Automation with Energy Efficiency and Comfort Concerns

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: Vasco Amaral, Assistant
Professor, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa

Co-adviser: Jácome Cunha, Assistant
Professor, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2017

IoT Platforms for Building Automation with Energy Efficiency and Comfort Concerns

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would like to start by thanking my advisor, Assistant Professor Vasco Amaral, and coadviser, Assitant Professor Jácome Cunha for all support, advices and availability provided while writing this document. Also, a big thanks to my colleague João Cambeiro, for all technical support. I also would like to thank to the NOVA University of Lisbon and in particular to the Department of Informatics, for all the journey of the last years. My closest family for giving me the opportunity to study far away from my home town. All my friends and colleagues that always supported me along this journey.

Abstract

It is increasingly common to work and live in buildings controlled by some system, the so-called Building Automation Systems, where to keep the levels of comfort and reduce energy consumption are very important requirements. These systems control from heating, ventilation, air conditioning, to lights intensity, with the goal of reducing energy costs and make the building occupants satisfied.

However, these systems are proprietary and have high costs associated, due to the required equipment to deal with all the devices and the distinct communications. Therefore, in general the main goal is to reduce these costs, being quite difficult due to the vast devices heterogeneity.

In this dissertation, we implement a Building Automation System taking advantage of existing IoT solutions. Thus, this thesis explores how IoT solutions can fit adequately in the scenario of building automation.

To validate our technological choices and evaluate the adequacy of the chosen middleware, we made use of an existing case study of a room with multiple components and an aquarium as a subsystem. We have compared the different IoT approaches and their impact in the energy consumption and occupants comfort.

The results obtained helped us to realise that in fact there are several aspects that can be improved in order to reduce energy consumption and maintain occupant comfort. An initial investment in the implementation of these systems may involve different equipments and development effort to achieve the desired solution. However, in long term it is worth the effort and initial investment on these systems, since they can actually reduce the energy consumption and guarantee good conditions for the room occupants.

Keywords: Cyber-Physical System, Internet of Things, Internet of Things Architectures, Internet of Things Platforms, Building Automation, Intelligent Buildings

Resumo

É cada vez mais comum trabalhar e viver em edifícios controlados por algum sistema, os chamados Building Automation Systems, onde manter os níveis de conforto e reduzir o consumo de energia são requisitos muito importantes. Estes sistemas controlam o aquecimento, ventilação, ar condicionado e intensidade das luzes, com o objetivo de reduzir os custos de energia e satisfazer os ocupantes do edifício.

No entanto, estes sistemas são proprietários e têm um grande custo associado devido ao equipamento necessário para lidar com todos os dispositivos e as diferentes comunicações. Portanto, em geral, o grande objetivo passa por minimizar este custo, sendo complicado devido à grande diversidade de dispositivos.

Nesta dissertação, implementamos um Building Automation System aproveitando as soluções IoT existentes. Assim, esta tese explora como as soluções IoT podem adequar-se a um cenário de building automation.

Para validar as nossas escolhas tecnológicas e avaliar se o middleware escolhido é o mais adequado, usamos um caso de estudo existente de uma sala com vários componentes e um aquário como subsistema. Comparamos as diferentes abordagens IoT e o impacto que estas têm no consumo de energia e no conforto dos ocupantes.

Os resultados obtidos ajudaram-nos a perceber que de facto existem vários aspetos que podem ser melhorados de maneira a reduzir o consumo de energia mantendo o conforto dos ocupantes. Um investimento inicial na implementação destes sistemas podem envolver vários equipamentos e um esforço de desenvolvimento para atingir a solução desejada. No entanto, a longo termo vale a pena o esforço e o investimento inicial nestes sistemas, uma vez que estes conseguem de facto reduzir o consumo de energia e garantir boas condições de conforto para os ocupantes da sala.

Palavras-chave: Sistema Ciber Físico, *Internet of Things*, Arquiteturas *Internet of Things*, Plataformas *Internet of Things*, Automação de Edifícios, Edifícios Inteligentes

Contents

List of Figures

List of Tables

Glossary

Arduino Micro-controller used to build digital devices with capabilities to sense and control objects in the physical world.

Enterprise Service Bus Refers to an environment designed to promote interconnectivity between different services.

feedback loop Outputs of components serve as input as a part of chain of cause and effect.

Gateway Piece of networking hardware that interconnects networks with different network protocol technologies by performing the required protocol conversions.

Message Broker Refers to a program module that translates a message from one protocol to another, validate messages and route the message for one or more destinations.

Acronyms

API Application Program Interface.

ARP Address Resolution Protocol.

BAS Building Automation System.

BLE Bluetooth Low Energy.

CPS Cyber-Physical System.

CSV Comma Separated Values.

FR Functional Requirements.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure as a Service.

IoT Internet of Things.

JSON JavaScript Object Notation.

MQTT Message Queuing Telemetry Transport.

NFC Near Field Communication.

NFR Non-Functional Requirements.

OSGi Open Service Gateway Initiative.

PaaS Platform as a Service.

REST Representational State Transfer.

RFID Radio Frequency Identification.

SDK Software Development Kit.

UI User Interface.

UML Unified Modelling Language.

US User Stories.

Chapter 1

Introduction

This chapter contains a brief introduction about the work developed in this dissertation. Section 1.1 introduces a short description of BAS. Then, the challenges of implementing a Building Automation System (BAS) are presented (section 1.2), followed by the problem statement and final goals (section 1.3). We then lists the expected results of this dissertation (section 1.4). Lastly, we present the global structure of this document (section 1.6).

1.1 Context and Description

The automation of tasks is an old concern of computer science, either safety reasons (to make various tasks safer for humans) or because of economic reasons (to increase productivity and efficiency). If we extend this concerns to other domains such as Building Automation, we see a focus on the topic of reducing energy consumption.

In the last few years, the urgent global environmental issues have favoured a quick change in how energy is used. For a sustainable development and a reduction of the impact that energy consumption has on the environment, it is necessary to resort to strategies that allow us to use resources in a more intelligently [SB15].

The biggest impact of the unintelligent use of energy lies in the large buildings (industrial and service). Thus, the concept of BAS has emerged with the motivation of environmental safety.

BASs began by resolving the major part of energy waste in buildings: heating, ventilation, and air conditioning systems. However, lighting expenses should also be taken into account, especially in commercial or even residential buildings [Ast+16].

Several technologies help in the implementation of these systems. The general idea is to have components installed in the environment with measuring capabilities (sensors), that provide data to the system. The system is in charge of analysing the data provided by sensors and sending signals to other components installed with the capability of changing the desired state (actuators). Actuators are mechanisms responsible for turning on/off or changing intensity values of some device.

This setup has provided the concept of intelligent buildings, which can provide comfort to the occupants, while efficiently maximise the use of energy. In this way, it is possible to observe a reduction of energy costs and the environmental impact [Dom+16].

1.2 Challenges

Designing and implementing a BAS is a complicated process where several challenges can arise.

Initially, it is important to analyse the environment where the system will act. Sometimes it is possible to find some control system already implemented [Ast+16; Lea+14]. This makes it a challenge due to the closed, custom, lack of documentation, and property nature of these systems, which gives the equipment manufacturers full control over installations and upgrades. Therefore, these systems have a high cost associated [Cor14; Ker+16; Tat16].

On the other hand, it is important to know what kind of components are necessary to control and how the system will communicate with them. The system must take into account a vast heterogeneity of components available, and have the capacity to deal with them. The challenge is then to decide the proper architecture and technologies that the system should adopt, not only to handle all kind of components but also to make it easily adaptable when new components must be supported [Dom+16].

There is a need to realise what entities will interact with the system and understand their roles and preferences. To engage the entities to help reducing energy costs, techniques and mechanisms based on games (gamification) can be used. This will create conditions to stimulate the entities to take specific actions by making use of the human psychological predisposition to engage in games. These techniques can encourage humans to complete tasks that would normally be tedious [Det+11]. However, having humans has a part of the system may cause some issues due to the complex psychological and behavioural aspect of humans [Mun+13].

The vast heterogeneity of devices is a problem for the BASs. There is no simple way to deal with this vast variety, so building owners need to choose equipment within the same range, which makes the price of these solutions higher [Dom+16].

The Internet of Things (IoT) concept has been revolutionising the implementation of these systems, with the vision to connect everything to the Internet, and it is possible to acquire low-cost hardware modules that make buildings more efficient and economical to operate [Att+16; Cor14].

Currently, many IoT platforms exist and the functionalities provided can help the implementation of BASs [Cor14; Tat16; Zha+16]. Despite each platform documentation and functionalities provided are well-defined, it is hard to understand which solution is the most appropriate for a specific context.

1.3 Problem Statement and Final Goals

Taking into account the challenges stated in section 1.2, the goal of this dissertation is to provide a comparative analysis of the existing **IoT platforms for building automation with energy efficiency and comfort concerns**.

Thus, the research question this dissertation intends to answer can be formulated as follows:

- **What are the concerns that should be taken into account, and the most appropriate IoT solutions that can be used, to implement a BAS while ensuring low energy consumption and occupant comfort?**

To understand the best solution to adopt based on existing work, we have analysed what were the key concepts an IoT platform should adopt in order to support the IoT paradigm. In chapter 4 we present a discussion on the several possible solutions based on architecture. The most relevant aspects of each one take into account the case study described in chapter 3. However, for this thesis, we had the requirement that the solution should not be limited to the case study, so that it could be more general. Therefore, it should be considered the platform capacity to be easily adapted to new requirements, a different environment, the detection of errors and unexpected behaviours.

To validate this work, we have implemented a solution using some of the selected IoT platforms and compared them taking into account the concerns coming from the conducted analysis, and if the goal of ensuring low energy consumption and occupant comfort is fulfilled.

1.4 Expected Contributions

As we will see along this document, at the end of this dissertation, based on the research carried out, it is expected to contribute to the state-of-art and accomplish the following goals:

- A study about the state of the art of the Internet of Things architectures;
- An analysis about the selected Internet of Things platforms;
- An analysis of what are the key concepts and features an Internet of Things platform should provide;
- A comparison about the selected Internet of Things platforms

- An explanation of how Internet of Things supports the implementation of Building Automation Systems;
- A validation plan to analyse energy efficiency in real case scenario considering the behaviour of different users.

1.5 Research Project

The research conducted by this dissertation is part of the NOVA LINCS Smartlab research project, which aims to create an integrated automation solution for an open office. The Smartlab will serve as our case study (described in chapter 3).

The result of the analysis coming from the first chapters will help us to understand what are the concerns that should be taken into account and an adequate IoT platform that can be used to implement a BAS in the Smartlab.

1.6 Document Structure

In addition to this chapter, this document is structured as follows:

- Chapter 2 - **Background**: This chapter explains important concepts to understand the work proposed in this dissertation. It starts by explaining the concept of CPS (section 2.1) and IoT (section 2.2). Then it introduces the idea of BAS and its components (section 2.3).
- Chapter 3 - **Case study**: This chapter describes our case study. It starts by describing the scenario (section 3.1), all the components and the relationships between them (section 3.2). Then, it describes the available services provided by these components (section 3.3). It concludes by describing the mechanisms implemented for occupancy detection (section 3.4).
- Chapter 4 - **State of the art**: This chapter presents the state of the art of the IoT architectures (section 4.1) and the required elements to accomplish an IoT solution (section 4.2). Then, it introduces some existing platforms for implementing an IoT solution (section 4.3). Then, it presents a list of what are the key concepts and features an IoT solution should provide (section 4.4), and whether these platforms provide or not these concepts. After explaining what an IoT platform is, it introduces the IoT middleware concept (section 4.5) and Fog Computing (section 4.6), and explains how these concepts are related to the IoT architecture and platforms. It ends by presenting how IoT enhances BAS (section 4.7).
- Chapter 5 - **Comparison Review**: This chapter describes in more detail the selected platforms. It starts by presenting the differences in the process of connecting new devices to the platform (section 5.1). Then, it describes the tools provided for Data

Processing and Visualisation (section 5.2). Then, it presents how the platforms handle Application and User Management (section 5.3). In the end it is presented a detailed discussion about the platforms (section 5.4).

- Chapter 6 - **Conceptualisation and Implementation**: This chapter describes our work on the case study, from the requirements to the implementation of a concrete solution. It starts by presenting the conducted requirement analysis, describes the stakeholders, the user's stories, the functional and non-functional requirements (section 6.1). Then it details the user stories using use cases (section 6.2). In the sequence, it presents the architecture styles and views selected (section 6.3). Finally, it ends by showing the result of the implementation phase (section 6.4).

- Chapter 7 - **Evaluation and Results**: This chapter presents the evaluation of our work in several dimensions. It starts by showing the results of the questionnaire conducted to collect the opinion of the occupants of the case study (section 7.1). Then, it presents the analysis of the temperature and brightness data (section 7.2). In the sequence, it presents the data regarding the mechanism used to detect presence and the events associated with the presence of occupants (section 7.3). Lastly, it presents the before and after energy consumptions and associated costs (section 7.4).

- Chapter 8 - **Conclusion**: This chapter closes the dissertation by summarizing the results (section 8.1 and section 8.1), while highlighting its limitations (section 8.3) and pointing to future directions of this research (section 8.4).

Chapter 2

Background

This chapter explains important concepts to understand the work proposed in this dissertation. It starts by explaining the concept of CPS (section 2.1) and IoT (section 2.2). Then it introduces the idea of BAS and its components (section 2.3).

2.1 Cyber-Physical System

Nowadays we can see many applications of Cyber-Physical Systems (CPSs). From minuscule systems such as pacemakers to large scale such as power grids. So it is possible to witness CPSs in different areas such as medicine, aerospace, transportation vehicles, defence and robotic systems, industry and building automation systems. CPSs can interact with human through many modalities providing social and economical advantages [Lee+10].

CPSs are defined as the systems that are composed by physical processes, cyber components and network mechanisms. The system operations are monitored and controlled by embedded computers and networks usually with feedback loop where computations affect a specific physical process and vice versa.

Lee and Seshia defined CPS as a set of the following four layers:

- **Physical Layer**: Corresponds to the part of the system that is not accomplish neither with computers or digital networks, that exist in nature. It may include mechanical parts, biological and chemical processes or human actions (represented as physical plant in Figure 2.1);

- **Control Layer**: Is composed by computers responsible to get sensors information, analyse these information with logical controls and make a decision. As a result, computers send an action through a group of actuators that affect the respective physical process [LS15; Raj+10];

- **Sensors and Actuators Layer**: Composed by devices responsible for collect information present in the physical layer and affect a respective physical process [KM15; Raw+15]. The sensors and actuators are the interface between the physical and cyber world. As illustrated in Figure 2.1 a CPS may contain several systems and devices. In order for these components to exchange information a network mechanism is required;

- **Network Layer**: Represents the mechanisms provided for the components of the CPS to communicate.

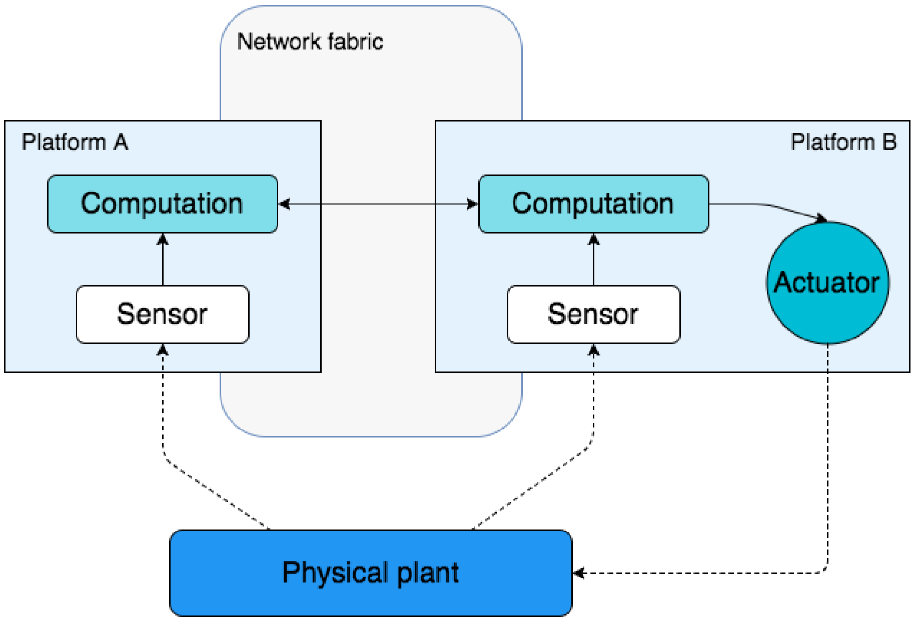


Figure 2.1: Cyber-Physical System model (adapted from [LS15])

Despite the systems and engineering evolution there are some points that should be taken into account. Physical world is not predictable. Therefore systems must be prepared to adapt to unexpected conditions. So, these systems must operate dependably, efficiently, in real-time, and take into account safety and security [Lee08; Lee+10].

The next chapter will introduce the concept of IoT. These concepts are related as, both reflect a vision that aims to connect physical world with cyber components. Lee and Seshia defend that the term CPS is more foundational and durable because it does not compromise with implementation approaches or particular applications [LS15]. Figure 2.2 illustrates the relationship between these two concepts. CPS can be systems connected via Internet and non-Internet technologies, while the IoT are systems that are connected only via Internet technologies.

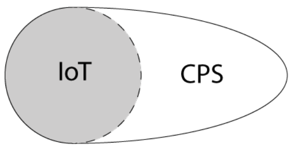


Figure 2.2: Relationship between CPS and IoT (adapted from [Hen16])

2.2 Internet of Things

IoT refers to a technological revolution which aims to create a world where physical devices (things) are connected through the Internet, making it possible for these devices to collect and exchange data in order to accomplish certain goals in a specific context. As mentioned before the IoT differs from CPS because it refers uniquely to devices connected to each other through the Internet. The IoT solutions usually combine physical things with hardware or software. As a result the physical functions of a thing can be enhanced with additional functionalities [Eva12; WF+15].

A particular detail of IoT vision (Figure 2.3) is that these devices are invisibly embedded in the environment around us. Therefore, it is possible to achieve a distributed network of devices of various types communicating with each other with minimum human intervention [Kop11; Per+14; Xia+12]. To make the devices unnoticed to the user IoT requires [Gub+13]:

- Shared knowledge of the state of its users and devices;
- Processing contextual information;
- Analytic tools to help in autonomous and smart behaviour.

IoT are opening new opportunities for a wide number of applications with the promise of increasing our life's quality [Xia+12]. Based on scale, coverage, and user involvement it is possible to divide IoT applications per domain. These domains are classified into:

- Society;
- Industry;
- Environment.

In short, the IoT main goal is to create a better world for human beings. To achieve this goal it is required that the devices have knowledge about what the users want, what the users need and like in a specific time and space, and acting accordingly without direct human instructions [Per+14].

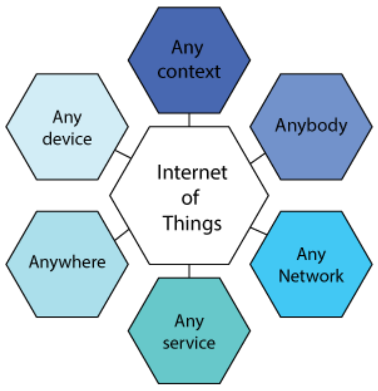


Figure 2.3: Internet of Things vision (adapted from [Per+14])

2.3 Building Automation System

In energy management applications of CPS in large scale, when various systems like heating, ventilation, air conditioning and lights are networked and controlled to achieve some specific goal of energy efficiency and occupants comfort, these type of systems can be called BAS. That is, BAS is a class of a CPS [Ree+15].

In more detail, a BAS is a system that controls and monitors building services responsible for heating, ventilation, air conditioning, lighting and others.

BASs received attention due to its potential to reduce energy costs and make building operation simpler, while improving indoor environment and minimizing environmental impact. To achieve this potential, BASs required a wide range of interconnected components in a distributed manner, which provide information about the environment and enables decision-making regarding how the controlled components will act in order to provide energy reduction costs and occupants satisfaction [Bra+05]. The functions provided by a BAS are distributed in the following areas [Ast+16]:

- heating, ventilation and air conditioning;
- lighting systems;
- shading systems;
- monitoring and data acquisition;
- security and safety management;
- power generation systems;
- energy conservation and storage.

2.3.1 Architecture Levels

The architecture of this distributed system (represented in Figure 2.4) can be organized in three hierarchical levels [Dom+16; Fer+11; Lil+17]:

- **Field level**: In this level belongs all the field devices (sensors and actuators) responsible for metering, setting and switching;
- **Automation level**: This level is responsible to provide control functionality. It is responsible for process measurements, execute control loops and activate events;
- **Management level**: In this level is where all the information about the system is collected and represented. This is where the configurations of the system are introduced. Activities like data visualization, generation of reports and long term data storage belong to this level.

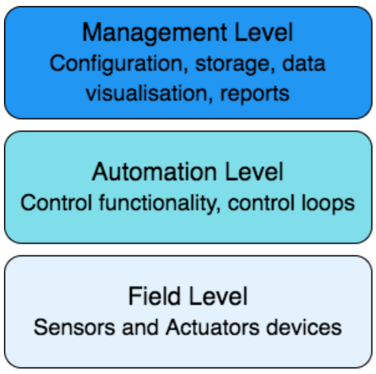

Management Level
Configuration, storage, data visualisation, reports
Automation Level
Control functionality, control loops
Field Level
Sensors and Actuators devices

Figure 2.4: Building Automation System Architecture Level (adapted from [Dom+16])

2.3.2 Sensors, Actuators and Controllers

BASs employ a wide number of components as mentioned above. Without automated monitoring it is hard to operate and remain aware of equipment and system conditions [Bra+05]. In order to fulfil automation BASs requires various types of components. These components are responsible for sensing, metering, setting and controlling and are defined as follows [Ast+16; Dom+16]:

- **Sensors**: Devices with measures capabilities. They measure physical quantities and convert them to a digital or analogue signal;
- **Actuators**: Used to modify the intensity or change the state of physical devices;
- **Controllers**: Application specific hardware with embedded software that controls physical actuators. These hardware modules can have input and output capabilities. Input ports allow a controller to receive data from a monitored inputs or commands from the system. Output ports allow a controller to send signals to controlled devices.

2.3.3 Communication Networks

Communication protocols denote the physical media through which information and commands pass between devices, and are the central question of interoperability [Bra+05]. The backbone of the field level is the **fieldbus**, a digital communication link between field devices such as sensors, actuators and controllers. Devices on the fieldbus network are identified with a unique address and fieldbus support a two way communication that provides reading data from the device and writing into it [LS13]. In this way field devices send and receive information over fieldbus and can communicate with each other or with control devices at the automation level [Mer+09].

The fieldbus results in cable savings and resultant cost reduction comparison to previous analogue communication buses. The field devices can carry out their own computational capabilities. Field devices interoperability doest not become an issue, since fieldbus devices are interoperable. Thus devices from different manufactures can work together without loss of functionality [Sen14].

As mentioned above IoT is about connecting devices, performing computational processes either in embedded computers or in cloud services. In this sense, IoT may actually support the evolution of BASs [Ast+16; Cor14].

2.3.4 Occupancy Detection

One of the goals of BASs is to maintain indoor comfort while reducing energy costs. For this purpose BASs requires occupancy detection techniques supported by sensors or other mechanisms. The most common occupancy detection techniques are based on the following strategies:

- presence or movement detection;
- CO_2 concentration.

In order to occupancy detection to work strategies need to be implemented. These strategies can be supported by two types of systems:

- **Terminal based detection systems**: Occupancy detection is based on mobile phones or Radio Frequency Identification (RFID) tags inserted in objects carried by the occupants;
- **Non-terminal based detection systems**: Occupancy detection is based on sensors to measure CO_2 concentration, infrared levels emitted by surrounding objects and image recording devices.

For reliable occupancy detection different mechanisms should be adopted taking into account the BAS context and the occupants involved [Ast+16].

2.4 Summary

In this chapter it was described the fundamental concepts for a better understanding of the next chapters. We start by explaining what a CPS is, its components and characteristics. Then, we talk about the IoT concept, its components and how it relates with the CPS. Lastly, we introduce the BAS concept, its components and architecture. All of these concepts are important because one of the main goals of this thesis is a BAS system implementation based on IoT technologies. It is important to highlight the CPS concept because this particular case study takes place in an office where its occupants and also the physical aspects are part of the system. Thus, the system has to deal with different equipments and their data (IoT), in order to create automation rules to an energy reduction (BAS) and to deal with the physical aspects and the human presence in the system (CPS).

Chapter 3

Case study

This chapter describes our case study. It starts by describing the scenario (section 3.1), all the components and the relationships between them (section 3.2). Then, it describes the available services provided by these components (section 3.3). It concludes by describing the mechanisms implemented for occupancy detection (section 3.4).

3.1 Overall Description

The Computer Science Department at FCT/UNL has a room, called Smart Lab, with several components installed. The Smart Lab will serve as a test case scenario for the proposed system. The goal is to suggest a simulation platform for energy efficiency studies considering the behaviour of the Smart Lab occupants. To design the system architecture it will take into account all the components already installed in the Smart Lab and their occupants. The challenge is to autonomously control the required components in order to minimise energy waste and operation costs without disturbing and causing side effects to the Smart Lab occupants.

The proposed system must continuously acquire and analyse information from the installed components. Afterwards, it must make the best decision based on the present or absent occupants and the current time.

In Figure 3.1 is illustrated a simplified model of the Smart Lab. It is possible to see that every desk in the Smart Lab has a computer, a light bulb and an outlet. It is also installed a fish tank as a subsystem, a coffee machine, and air conditioning system. The following sections will clarify all the components in the Smart Lab in order to accomplish the desired result.

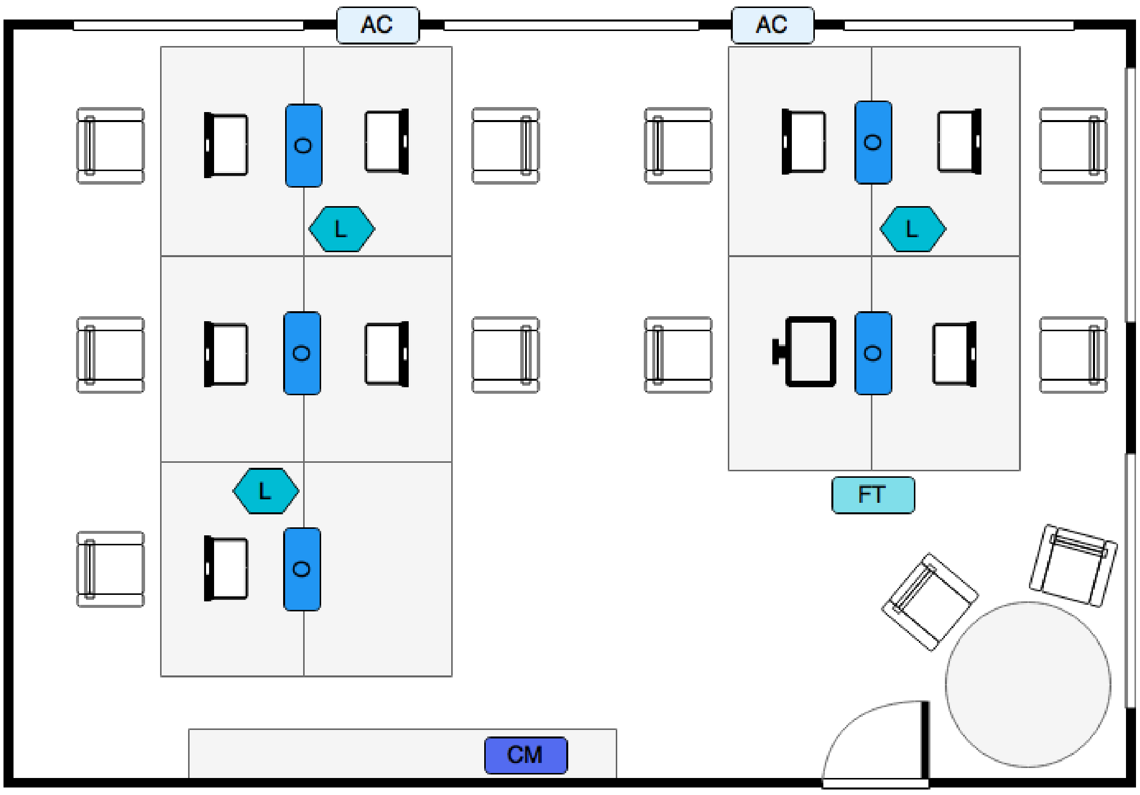


Figure 3.1: Smart Lab simplified 2D model. Legend: AC - Air Conditioning, CM - Coffee Machine, FT - Fish Tank, L - Light Bulb, O - Outlet

3.2 Physical Setup

As mentioned before the Smart Lab is equipped with several components. These components can be distributed by different categories (represented in Table 3.1). In the sensors category we include all the devices responsible for acquiring values about the Smart Lab environment. Currently, we measure luminous intensity, temperature, both inside and outside the room, and energy consumption. It is also implemented a system for occupancy detection (described in section 3.4). The actuators category is composed by the devices responsible to change a specific physical value and control the energy waste in every desk present in the room. The computer will be used as a control component, responsible to evaluate the devices data and apply the required logic to understand what commands need to be sent to the actuators.

In the room it is possible to find a fish tank that contributes to a relaxing atmosphere in the room. The fish tank present in the Smart Lab is its only subsystem. The fish tank subsystem has its own components in order to maintain the fish tank operational (represented in Table 3.2). In addition to help taking care of the fish, the subsystem also needs to take into account the aquaponic capabilities provided by the fish tank. To accomplish this the fish tank is composed by a set of sensors and actuators in order to automate the control and maintenance tasks of the fish tank.

Table 3.1: Smart Lab physical components installed

Component	**Type**	**Description**
LED Lights	Actuator	Used to change the light intensity of the room. It can be turned on or off. It is also possible to set a specific value for light intensity and colour.
Outlets	Sensor/Actuator	Outlets have both measuring and sensing capabilities. They can be used to measure the energy at a specific workstation. In the other hand it is possible to turn on or off the outlet.
Estimote Beacons	Sensor	Estimote Beacons are used for occupancy detection, temperature and light measuring.
Computer	Controller	Computer used as the main controller of the installed devices on the lab.

Table 3.2: Fish tank equipped components

Component	**Type**	**Description**
Open Aquarium	Controller	Responsible for automating the control and maintenance tasks that take place in the fish tank, in order to maintain a good, environment to the fish.
Ph level sensor	Sensor	Component used to measure the Ph level of the tank water.
Temperature sensor	Sensor	Component used to measure the current temperature of the water.
Water level sensor	Sensor	Component used to measure the level of the water inside the tank that may decrease due to evaporation.
Lights	Actuator	The tank has some plants that need a certain time of light exposure. This actuator is responsible to provide luminosity to the plants inside the tank, if the natural light is not sufficient.
Ventilator	Actuator	Based on the water temperature, this actuator is used to cool down the water if high temperature levels are measured.
Feeder	Actuator	This actuator is used to feed the fish according to a pre-defined schedule or, though the computer.

One Arduino is installed to control the fish tank tasks. The sensors and actuators are connected to the I/O ports of the Arduino. The Arduino is configured in order to feed the fish in a specific schedule and turn on/off the lights to provide luminosity to the plants inside the tank. This controller is connected to the server mentioned above making it possible to manually control and monitor the fish tank through the provided computer.

3.3 Available Services

The setup coming from the components mentioned before provide a set of services. The following sections describe in more detail the services supported by the current setup present in the Smart Lab.

3.3.1 Control and Monitor

The Smart Lab has a computer responsible for control and monitor activities. It provides the capability to get information about the state of the sensors and control actuators through a developed Application Program Interface (API). A User Interface (UI) is also provided in order to visualize relevant information about the components installed in the room.

All the components listed before are based on Internet communication. Therefore, all the sensors, actuators, and controllers have a given IP address and the computer can communicate with them via the components IP address.

3.3.2 Lights System

It is possible to observe in figure 3.1 that the Smart Lab has some light bulb installed. These bulbs can be turned on/off directly in the bulb switch as a normal one, and they provide a mechanism to control them remotely. The computer provides the mechanisms to obtain the status of each bulb in the room, to turn it on or off, and to change a light intensity and colour.

3.4 Occupants Comfort

The Smart Lab is used as a work room and occasionally for meetings, and is attended by various kinds of people. To provide the occupants comfort while reducing energy costs, it is required to implement mechanisms in order to the system acknowledge the presence of occupants in the room.

To accomplish this requirement the Smart Lab is equipped with a set of devices (Beacons) to detect human presence through proximity technologies. Beacons are tiny, low power computers that can be attached to walls or objects in the physical world. This devices provide the knowledge about human presence or the specific entities present in the Smart Lab.

The Smart Lab entrance door has a card reader equipment. Only authorized entities can enter the room. However, using this strategy is not enough. Think of a meeting composed by several entities and just one have the authority to open the door. If that entity goes outside the room for some reason it is required to have mechanisms to understand that there are other entities inside. Thus, it is necessary to adopt multiple mechanisms that support occupancy detection [Ast+16].

Providing this knowledge to the computer it is possible to adopt strategies based on the present or absent occupants. The computer, based on the knowledge provided by occupancy detection mechanisms, can adopt strategies to reduce energy costs, such as turn off the lights or the unused outlets.

3.5 Summary

In this chapter, we introduce the case study where the main goal of this work and the next chapters are based on. It is important to notice that there are different equipments installed in the office and the way they are arranged. In the next chapters, namely on the requirements analysis and the conceptualization phases, it will be taken in consideration these equipments. Lately, it will be important to recall the office plant due to the fact that the results reflect the equipments position relative to the office.

Chapter 4

State of the art

This chapter presents the state of the art of the IoT architectures (section 4.1) and the required elements to accomplish an IoT solution (section 4.2). Then, it introduces some existing platforms for implementing an IoT solution (section 4.3). Then, it presents a list of what are the key concepts and features an IoT solution should provide (section 4.4), and whether these platforms provide or not these concepts. After explaining what an IoT platform is, it introduces the IoT middleware concept (section 4.5) and Fog Computing (section 4.6), and explains how these concepts are related to the IoT architecture and platforms. It ends by presenting how IoT enhances BAS (section 4.7).

There are some solutions that implement a building automation with Internet technologies. These solutions implement a prototype system to monitor and control the filed devices in order to reduce energy consumption costs. Jung et al. and Mohamed et al. offer a solution to integrate existing BAS with cloud computing technologies, through a gateway, in order to enhance the current system capabilities. Attitalla et al. and Gusmanov et al. offer prototype solutions that use available APIs and libraries, in order to connect the field devices and provide monitor and control functionalities. These solutions are not based in a reference architecture, and were implemented for a very concrete case. Whereby, they do not offer scalable, and adaptable solutions, that helps to deal with the devices heterogeneity or services integration. That is why, we pretend to analyse the existent approaches, understand the functionalities provided and realise how these fit in the building automation to create an adaptable and scalable solution.

4.1 Internet of Things Architectures

An IoT architecture should have the capacity to interconnect many heterogeneous devices, collect data from multiple sources, and connect several services in order to provide the expected functionality. Before discussing how an IoT architecture is structured, it is important to mention the requirements these architectures should take into account. Wang et al. and Ray list these considerations when designing and IoT architecture:

- **Interoperability**: Allows the integration of heterogeneous devices, networks, systems and services across domains and systems.
- **Service Oriented Architecture principle**: Allows third-parties to offer and consume services;
- **Service modularisation and loose coupling**: Simplifies provided and consumed services by third-parties through reusable and modularized services.
- **Multipoint communication**: Adopts mechanisms to allow objects to communicate with multiple objects at the same time.
- **Dynamic and runtime reconfiguration**: Enables adding and removing objects dynamically to networks. Due to network topology changes, it should allocate resources dynamically in order to create flows among all objects.
- **Simplified deployment**: Simplify the deployment of IoT in order to reduce development costs.
- **Controlled interaction and decentralisation**: Supports distributed data accessing, processing and storage. It allows the users to decide which data can be shared.

Now we can describe how IoT architectures are usually organized. There are several approaches that try to organize the architecture in several layers but the most common are the 3 and 5 layer architecture [AF+15]. The 5 layer architecture (illustrated in Figure 4.1) is composed by the following layers [AF+15; Ara+16; Li+15; Mar+17; Thi15]:

- **Perception Layer**: Composed by all the devices that interact with the physical world in order to collect data (sensors) or to change a desired state or value (actuators). Every device in this layer has a unique identifier (name or address) that identifies it n in the digital domain.
- **Network Layer**: Represents the mechanisms used to transfer the produced data in the Perception Layer to the Middleware Layer through technologies such as RFID, ZigBee, Wi-Fi, 2G, Bluetooth Low Energy (BLE) and many others.
- **Middleware Layer**: This layer pairs a service with its requester based on unique address or names. It provides an abstraction mechanism that enables developers to

work with models that represent a specific object without worrying about hardware specifications. This layer is also responsible for making decisions based on the received data and store data in databases.

- **Application Layer**: Provides output information based on the services requested by the end users. It forms the application to a specific context such as smart home, smart city and many others.
- **Business Layer**: Responsible for system configuration and monitoring activities. Represents the tools used to build reports and graphs based on the data received from the application layer. This layer also represents the mechanisms used to predict system behaviour based on big data analysis.

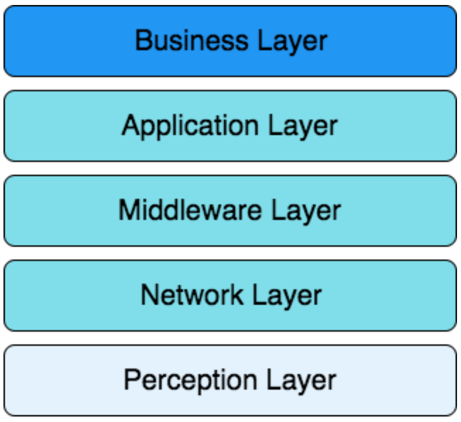


Figure 4.1: Internet of Things 5 Layer Architecture (adapted from [Kha+12])

The 3 layer architecture is an abstraction of the 5 layer architecture. Being structured as Perception Layer, Network Layer and Application Layer. The last layer (Application Layer) represents a merge between the Middleware, Application and Business Layers [AF+15].

4.2 Internet of Things Elements

There are a set of main elements needed to deliver the expected functionality from IoT based on the architecture layers mentioned above. These elements can be divided in the following categories [AF+15; Mar+17; Min13; Pra+16; Wan+16]:

- **Identification**: Device identification is crucial for the IoT to know all the involved objects. Actually, there are multiple methods for object identification through name or address. Identification by name its possible to achieve with electronic product codes (EPC) or ubiquitous code (uCode). Identification by address refers to assign an IP address to the object achieved with today's IPv4, IPv6 and others.

- **Sensing**: Sensing refers to the capacity of collect data from the environment through the related objects, analyse, and take specific actions based on the collected data. The IoT sensing devices can be smart sensors, actuators and wearable sensors.
- **Communication**: Communication is a crucial part of the IoT. It can be restricted to the devices characteristics such as, battery life, data transmission limited range and protocols. The most common protocols used are Wi-Fi, ZigBee, GSM, Bluetooth, Z-Wave, 6LowPAN, Message Queuing Telemetry Transport (MQTT), Thread and many others. Proximity communications such RFID, Near Field Communication (NFC) and BLE (Beacons) are commonly used.
- **Computation**: Computation is composed by hardware platforms such as Arduino, Raspberry Pi and others. Cloud platforms can also be used to provide storing or processing functionalities.
- **Services**: IoT services can be categorized in four types responsible for the following tasks:
  - **Identity-related Service**: Map the identified real world objects to virtual objects;
  - **Information Aggregation Service**: Collect and summarize the data provided by the devices;
  - **Collaborative-Aware Service**: Make decisions based on the data provided by the Information Aggregation services;
  - **Ubiquitous Service**: Provide Collaborative-Aware services to anyone at anytime.
- **Semantics**: The ability to extract knowledge to provide the required services refers to Semantic. In order to extract knowledge it is required to model information, recognize and analyse data to make sense.

4.3 Internet of Things Platforms

An IoT platform provides a set of generic functionalities that can be used to build an IoT application. It is a virtual solution, where data drives business intelligence and each device has something to talk with another device. Meaning that an IoT platform translates devices data, so that it can be used intelligently by another devices. Additionally, an IoT platform provide the required tools that enables a user to implement business use cases and it provides data management, and real-time analysis [NC15].

Currently, there are many solutions that allow us to accomplish an IoT solution. All the platforms present common concepts but there are some differences at architectural level and functionalities provided. The platforms we present were selected based on the criteria that they provide documentation explaining the architecture on which they are based. The following sections briefly describe the selected IoT platforms.

4.3.1 WSO2 IoT

WSO2 is an open-source technology company founded by Dr. Sanjiva Weerawarana and Paul Fremantle in August, 2005. The platform created by this company is based on the Open Service Gateway Initiative (OSGi) technology which allows components to be dynamically installed, started, stopped, updated, and removed. Therefore, it is possible to achieve a completely modular solution [Fre16a; Inc16a; Inccea]. In order to fulfil the IoT paradigm the WSO2 IoT Server was built by reusing the WSO2 components based on a reference architecture.

The architecture, represented in Figure 4.2, is organized in five horizontal layers and two vertical layers. These vertical layers (also called cross-cutting layers) represent the functionality that spans layers. This means, these vertical layers represent a set of functionalities (caching, validation, authentication) that are accessible to all the layers. Each layer is composed by multiple components that provide essential capabilities that help to implement a scalable IoT platform. These capabilities include tools to connect and manage all the devices, data analytics, API management for devices and web-based UIs.

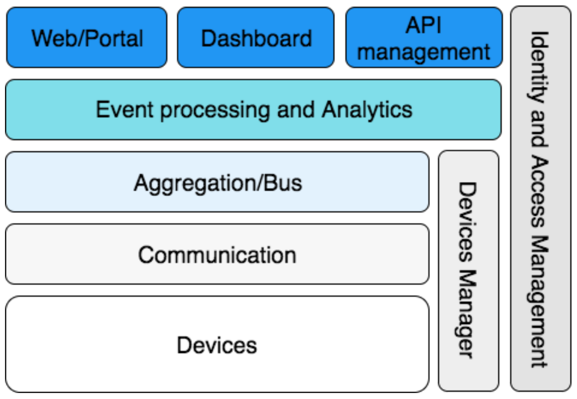


Figure 4.2: WSO2 architecture (adapted from [Fre16a])

Comparing the WSO2 architecture to the five layers architecture mentioned before, the Devices layer represents the Perception layer. The Communication layer corresponds to the Network layer. The Aggregation/Bus and part of the functionalities provided by the Event processing and Analytics layer represents the Middleware layer. The top three layers, Web/Portal, Dashboard and API management, represent the Application and Business layers [Fre16a; Inc16b].

4.3.2 IBM Watson IoT

IBM Watson IoT is a cloud platform created by IBM in 2014. This platform helps to create an IoT solution that aggregate data collected by the connected devices, sensors, and gateways. This platform was built based on an architecture, represented in Figure 4.3, structured in five layers.

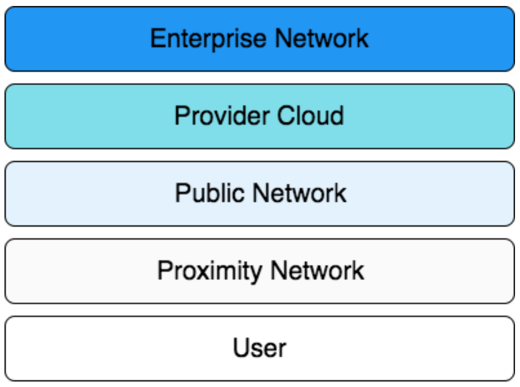


Figure 4.3: IBM Watson IoT architecture (adapted from [Watd])

Comparing the IBM Watson IoT architecture to the five layers architecture mentioned before, the User layers represents the Application layer. The Proximity network corresponds to the Perception layer. The Public network represents the Network layer. The Provider Cloud layer represents the Middleware layer. Finally, the Enterprise Network corresponds to the Business layer.

To fulfil the IoT paradigm IBM Watson IoT uses multiple platform services, each one responsible for a set of specific tasks. These services provide the tools to create an IoT solution. IBM Watson IoT provides recipes to simplify the connection of devices and scenarios that help to implement the architecture [Watb; Watd].

4.3.3 ThingSpeak IoT

ThingSpeak is a cross-platform created in 2010, that enables the creation of sensor applications. This platform was built based on an architecture, represented in Figure 4.4, structured in three layers.

Comparing the ThingSpeak IoT architecture to the three layers architecture mentioned before, the Things layer represents the Perception layer. The Cloud Service corresponds to the Middleware layer and the Services and Application layer represents the Application layer.

We decide to present this solution, because it has some different aspects compared to the other solutions. Namely, the fact that this platform does not provide Device Model mechanisms. The Things layer represents stream data channels, that allow us to send data without the need to associate this data with device properties [Thi14].

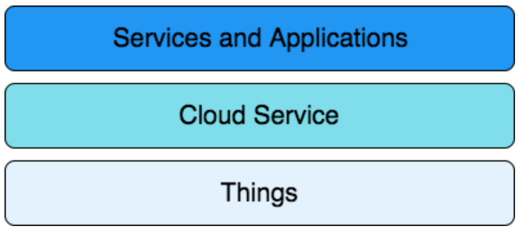


Figure 4.4: ThingSpeak IoT architecture (adapted from [Thi14])

4.3.4 Microsoft Azure IoT

Azure IoT Suite is an enterprise-grade solution created by Microsoft in 2016. This platform helps to build, deploy, and manage IoT solutions using Azure services based on an architecture. The architecture, represented in Figure 4.5, is structured in three layers and provides the required components to enable the communication between devices and cloud-based systems, and the integration of analytics, control and business processes.

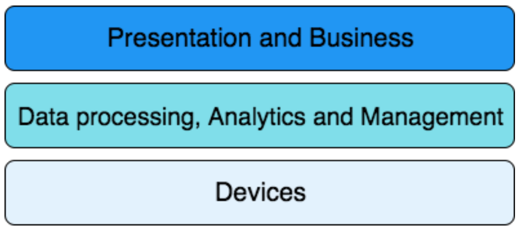

Presentation and Business
Data processing, Analytics and Management
Devices

Figure 4.5: Azure architecture (adapted from [Fre16b])

Comparing the Azure IoT architecture to the three layers architecture mentioned before, the Device connectivity layer represents the Perception layer. The Data processing, Analytics and Management corresponds to the Middleware layer. Lastly, the Presentation and Business represent the Application layer.

Azure IoT Suite provides preconfigured, completed and working solutions to address common IoT scenarios. The Azure IoT Suite is composed by several core platform services and application level components. The components provide a set of required functionalities in order the achieve a modular and flexible IoT solution [DBce; Fre16b].

4.3.5 Amazon Web Service

Amazon WS is an Infrastructure as a Service (IaaS) platform created by Amazon in 2006. This platform allows to easily connect devices, and interact with cloud services. This platform is based on a three layers architecture, represented in Figure 4.6, that provide the capabilities to device management, data analytics and presentation, and external communication to other services.

Comparing the Amazon Web Service architecture to the three layers architecture mentioned before, the Things corresponds to the Perception layer, the Cloud Service

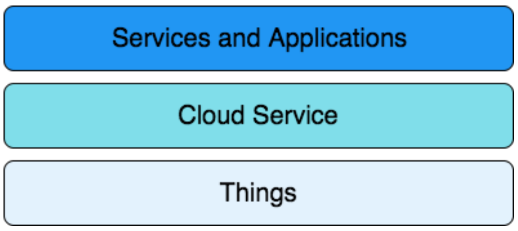


Figure 4.6: Amazon WS architecture (adapted from [Awsb])

represents the Middleware layer and the Services and Applications corresponds to the Application layer.

In Amazon WS the Things layer represents the provided SDKs that help to connect the hardware devices, authenticate and exchange messages using different communication protocols. This platform also provide a concept of Device Shadow of each device that includes the device latest state, making it easier to build applications that interact with the connected devices [Awsa; Awsb].

4.4 Internet of Things Platforms Comparison

Once the platforms have been described, we can start to define the most relevant aspects to be compared. These aspects are structured in four categories. In section 4.4.1 we list the key concepts an IoT platform should provide. In section 4.4.2 we detail the most common mechanisms used for data analytics. In section 4.4.3 we describe the communication models found in these platforms. Lastly, the section 4.4.4 present all the features provided by the selected IoT platforms.

4.4.1 Key concepts

After analysing in more detail the existing solutions, we noticed the concepts that an IoT platform should provide. The following list describes the key concepts from the analysis conducted and Table 4.1 compare these concepts between the mentioned platforms.

- **Device Model**: Enable the representation about device details, such as model and serial number, data emitted, configuration parameters and its operations.
- **Device Management**: Capability to support device management in order to maintain a list of connected devices and their status. It should provide tools to help in device registry, integration, enable/disable device features, control device identifiers and localization, and allow error reporting and handling.
- **Integration**: Capability to provide tools to integrate technologies to empower enterprises to build a connected business. Representational State Transfer (REST) APIs, Enterprise Service Bus and Message Broker are commonly used.

- **Analytics**: Provide tools to collect data from multiple sources and integrate real-time, predictive and interactive analysis.
- **Visualization**: Provide web-based UIs or dashboards for data visualization.
- **External Communication**: Allow to interact with systems outside its network using machine-to-machine communication (APIs).
- **Identity and Access Management**: Ensures security while connecting multiple identities from different applications, APIs and devices regardless of the standards which they adopt.

Table 4.1: Comparison about the key concepts of the studied platforms. Legend: no mention, low or no support ○, medium or partial support ◐, high or full support ●

	WSO2	IBM	Azure	AWS	ThingSpeak
Device model	●	●	●	●	○
Device management	●	●	●	●	○
Integration	●	●	●	●	●
Analytics	●	●	●	●	●
Visualization	●	●	●	●	●
External communication	●	●	●	●	●
Identity and Access Management	●	●	●	●	●

4.4.2 Data analytics

As mentioned in section 4.2 an IoT solution should provide the capability to extract knowledge from the data collected. In order to extract valuable information several mechanisms can be used.

Note that in section 4.4.1, we list Analytics as a key concept. We consider important to specify this aspect because, in our case study multiple mechanisms for data analytics can be used, in order to provide a better comfort for the occupants, and to realise strategies for reducing energy consumption costs.

The following list describes the mechanisms found along the analysis of solutions and Table 4.2 compare these concepts between the mentioned platforms.

- **Machine Learning**: Capability to learn and make predictions based on data through learning algorithms;
- **Natural Language Processing**: Ability to analyse, understand, and derive meaning from human writing or speech;
- **Predictive**: Predict unknown events based on current and historical facts;

- **Real-time**: Ability to analyse the data as soon it enters the system;
- **Batch**: Ability to process transactions in a group without requiring user interaction;
- **Image and Video**: Extract meaningful information from images and videos.

Table 4.2: Comparison about the data analytics tools provided by the studied platforms. Legend: no mention, low or no support ○, medium or partial support ◐, high or full support ●

	WSO2	IBM	Azure	AWS	ThingSpeak
Machine Learning	●	●	●	●	●
Natural Language Processing	○	●	◐	◐	○
Predictive	●	●	●	●	●
Real-time	●	●	●	●	●
Batch	●	●	○	●	○
Image and Video	○	●	●	◐	○

4.4.3 Communication models

IoT is about to connect everything, and therefore it is possible to find many types of communications between the existing components. On that matter a wide range of protocols arise in order to support the different communication models. The following sections describes the communication models found in IoT [Iot; MP]. Although in different ways, all the platforms provide the mechanisms to accomplish these communication models. The biggest difference between them resides in the way they simplify these communications, whether through services or custom gateways.

4.4.3.1 Device-to-Device Communication

Represents the communication between two or more devices that exchange data between them, as the example in Figure 4.7. These devices can communicate through many types of networks, including IP networks, but most often use protocols such as Bluetooth, Z-Wave, ZigBee and others. This communication model transfer small data packages of information between devices at a relatively low data rate [HAM16; MP].

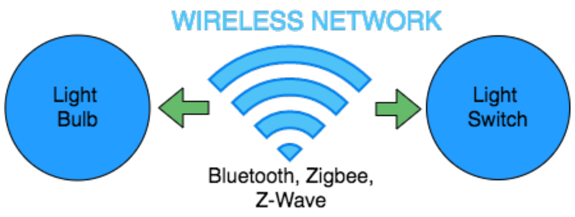


Figure 4.7: Device-to-Device communication model example (adapted from [HAM16])

4.4.3.2 Device-to-Cloud Communication

In the Device-to-cloud communication, exemplified in Figure 4.7, the device is connect directly to an Internet cloud service to exchange data. This approach uses communication mechanisms like traditional wired Ethernet or Wi-Fi connections to establish a connection between the device and the network, but can also use cellular technology. Device-to-cloud communication provides the capability to access devices remotely [HAM16; MP].

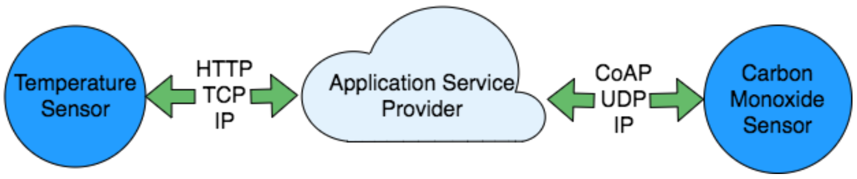


Figure 4.8: Device-to-Cloud communication model example (adapted from [HAM16])

4.4.3.3 Device-to-Gateway Communication

Device-to-gateway communication, exemplified in Figure 4.9, represents the communication between the device through a Gateway in order to reach an Internet cloud service. Gateway devices serve as an intermediary between the device and the cloud service, and provide other functionalities such as security and data protocol translation. Gateway devices can fill the gap between devices with different communications protocols [HAM16; MP].

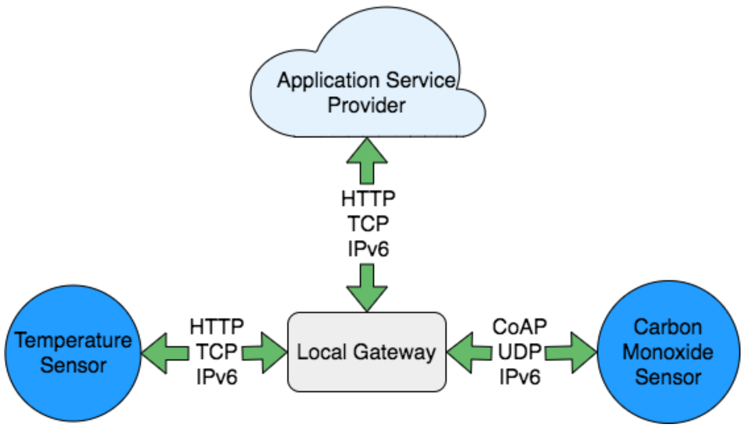


Figure 4.9: Device-to-Gateway communication model example (adapted from [HAM16])

4.4.3.4 Back-end data sharing

Back-end data sharing, exemplified in Figure 4.10 is a communication model that enables the end users to export and analyse data from an Internet cloud service in combination with data from other sources. This model also provides authorized third parties to access devices data [HAM16; MP].

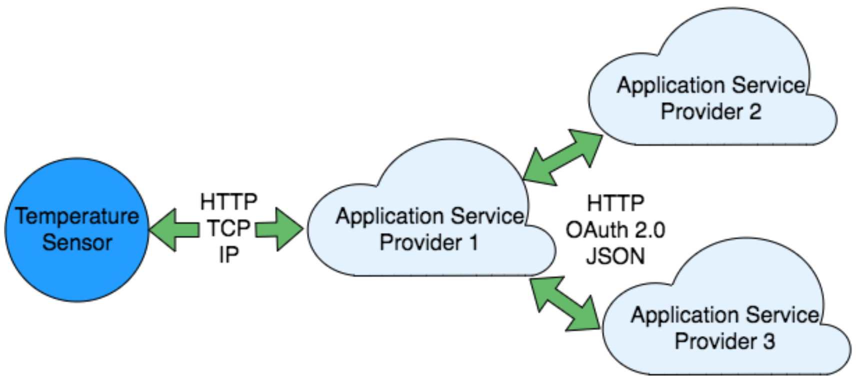

Temperature Sensor
HTTP
TCP
IP
Application Service Provider 1
HTTP
OAuth 2.0
JSON
Application Service Provider 2
Application Service Provider 3

Figure 4.10: Back-end data sharing communication model example (adapted from [HAM16])

4.4.4 Features

The following list describes the features found during the platforms analysis and Table 4.3 compare these concepts between the mentioned platforms.

- **Libraries/SDK**: Provide code to simplify the building and connection of devices and applications that are used by the IoT platform;
- **Virtual/Shadow Device**: Used as a communication layer between the application and the device. The virtual/shadow act as a persistent, virtual representation of the device state in a specific time;
- **Agents**: Software installed on the devices used to perform actions on behalf of another component;
- **Rule Engine**: Ability to create rules in order to trigger a specific action upon an event occurrence;
- **Group/Zone**: Ability to create groups or zones and manage the belonging devices;
- **Tasking**: Ability to forward control commands to the devices;
- **Messaging Services**: Ability to publish or subscribe to topics of the devices;
- **Alerts and Notifications**: Trigger notifications and alerts to notify the users about a certain event.

Table 4.3: Comparison about features provided by the studied platforms. Legend: no mention, low or no support ○, medium or partial support ◐, high or full support ●

	WSO2	IBM	Azure	AWS	ThingSpeak
Libraries/SDK	○	●	●	●	●
Virtual/Shadow Device	●	●	●	●	○
Agents	○	●	●	●	●
Rule Engine	●	●	●	●	●
Group/Zone	●	◐	●	●	○
Tasking	●	●	●	●	●
Messaging Services	●	●	●	●	◐
Alerts and Notifications	●	●	●	●	●

4.4.5 Discussion

Architectures are either more or less suitable for a specific context, therefore there is no such thing as an inherently good or bad architecture. However, there are some guidelines that should be followed when designing architectures. Even if the architecture does not satisfy one of the rules, it does not imply that the architecture will fail its purpose [Bas+12; Som10].

As mentioned in the section 4.1, the IoT architectures are usually structured in 3 or 5 layers, and all the platforms described in section 4.3 are in accordance with these patterns.

It is important to note that there are many solutions that allow us to create an IoT solution. The platforms mentioned in section 4.3 were selected because they are the ones that provide documentation explaining the architecture on which they are based and their architectures and elements provided resemble on the IoT architecture and elements mentioned in sections 4.1 and 4.2. As well as the features they offer are the ones that fit the case study. However, there are other solutions available that should not be discarded.

A question to be taken into account is the fact that some of the solutions mentioned are Platform as a Service (PaaS). This means that these platforms allow the users to develop, run, and manage applications without the concern of building and maintaining the infrastructure required to develop and launch an application [Law08]. This type of platforms have some advantages such as [Law08]:

- **Productivity/Efficiency**: Hosting the development environment increases productivity and lets release products faster and reduce software costs;
- **Scalability**: Eliminates the need to configure hardware modules, storage subsystems, and security, thus making the deployment environment scale.

However, PaaS platforms have some drawbacks too [Law08]:

- **Accessibility**: Connectivity problems or the PaaS system crash and the platform vendor goes out of business making the platform inaccessible;
- **Portability**: When using these platforms the users are dependent of the platform which they are working with. The developed applications are limited to the platform used and in many cases there is no easy way to transfer the product elsewhere;
- **Security**: All content is on the side of the platform vendor. The users are not in control of sensitive information within the platform.

In our evaluation we will take into consideration the characteristics usually associated to PaaS platforms we just introduced. However, we will simply assume their are present/absence if the platform under evaluation is a PaaS or not. That is, we will not further investigate the degree of each characteristic in the corresponding platform.

Another issue to be taken into consideration is the fact that solutions are composed by several components. Most of the components already exist. Therefore, by aggregating these components and providing the mechanisms to make them communicate with each other, it is possible to achieve a solution that fits the IoT requirements and functionalities.

However, we emphasize that, although the platforms can provide the expected characteristics from an IoT solution, the biggest difference that can arise between them is in terms of performance, usability, scalability, and ease of setup.

4.5 Internet of Things Middleware

In the above sections we saw what is an IoT platform, its requirements and presented multiple architecture approaches. However, there still exists solutions that abstract one of the problems of the IoT platforms. As mentioned before, nowadays there are multiple communication protocols and many types of devices. This often makes it harder to handle the communication between all devices. Therefore, other solutions may arise in order to provide smooth communication between all the components of an IoT system.

IoT middleware is an interface that allows the interaction between the IoT components. The main purpose of these solutions is to act like a mediator that hides the heterogeneity of the different devices, components and services in an IoT system. It does not replaces an IoT platform because these two have different approaches. We can consider an IoT platform as a suite that simplifies processes like development, deployment, maintenance, analytics and intelligent decision. An IoT middleware is a suite service mainly aimed to handle the heterogeneity problem of an IoT system, allowing smooth communication between the different components. Additionally an IoT platform specifies internally a middleware, it is specific and in accordance with the platform architecture [NC15].

There are several IoT middleware approaches, and these also have different types of architectures. These architectures can be classified as service-based, cloud-based and actor-based. The service-based IoT follows a three layer architecture composed by Physical Plane (sensors, actuators), Virtual Plane (server infrastructure) and the Application Plane (utility). This approach has some limitations, namely because it provides the functionalities to collect data but does not for data analysis. It is also a heavy solution that requires computational resources and it needs to be deployed in multiple nodes or powerful gateways between the IoT devices. The cloud-based IoT is composed by functional blocks and the services are limited to the existing blocks. The provided functionalities are exposed through a set of APIs. The resources provided can be accessed and controlled only by vendors provided applications or cloud supported APIs. The actor-based IoT can be visualised as a three circular architecture. The outermost represents the sensors and actuators, the middle circle the devices used for access and the inner circle the cloud. A particularity of this approach is that the middleware can be embedded in all the layers, since it is designed to be light-weight. Thus, the middleware computation units are distributed across the network.

The difference between these approaches resides on the provided support to add new devices, the type of services and computer units they support and where the middleware can be deployed. The service-based is deployed on servers or in the cloud. Usually it has a limited set of functionalities and it is restricted on external services integration. The service-based is also not designed to be extendible or customised by the users. The actor-based architecture provides the best latency and scalability because it can be deployed in any layer and device. Thus, it can perform computations where it provides more benefits.

All these approaches support security and privacy. However, the cloud and service based approaches may have weak security in the communication between the middleware and the physical devices, due to the fact that middleware can not be deployed or embedded on devices. Therefore, it can compromise the data sent from devices to the middleware [Ngu+17].

4.6 Fog Computing

In the above sections we described what are IoT platforms and presented examples of some existing solutions. Then we described what are IoT middlewares and how they can fill the gap concerning the devices and protocols diversity. There is another concept called Fog Computing that can help on building IoT systems. Usually IoT system demands significant computation and storage resources. Thus, the question of where these resources should be placed may arise. An obvious solution would be to have these resources available through the cloud. However, using the cloud for IoT systems may be infeasible in many scenarios [Yan+14]. Yannuzzi et al. enumerates the following scenarios that are not favoured by the current cloud:

- **Mobility**: Applications that require compute and storage support on the move. Such support can be required under fast mobility patterns. Traditional cloud needs to adopt strategies to achieve pervasiveness, while providing the reliability expected by these applications.

- **Reliable control and real-time**: Applications placed in locations where the communication with the cloud is to expensive or unreliable. Obviously, we cannot guarantee reliable control under such conditions when these applications require very low latency. In these applications of IoT, sometimes are required compute resources to control and apply logic. Therefore, these scenarios demand external compute and storage resources. However, current communications and cloud are not prepared to handle these applications requirements based on low latency, or real-time.

- **Data aggregation and analytics**: Applications that involve data management and processing are well supported by the traditional cloud centralised model, and it is perfectly suitable for controlling and managing data produced by a large number of components. However, when multiple distributed components produce data and demand analytics and external processing for decision making, the traditional cloud may not have the capability to ensure this. Instead these applications required resources placed close to the data producers that can be used for local processing and data analytics for fast decision making. Relevant data will be pushed to the cloud only when its content is important.

Note that these scenarios are listed as features or requirements of an IoT platform, and yet there are IoT platforms based on cloud computing or in centralised data management mechanisms. So we can conclude that the IoT platforms face complex challenges for supporting the requirements presented above. Thus, the concept of Fog Computing offers a way to deal with these challenges. Yannuzzi et al. shows why combining the Fog Computing with cloud computing is the most plausible way to build an adaptable and scalable platform for IoT.

The concept of Fog Computing consists on carry out computation and storage functionalities to the near edges of the network. Figure 4.11 represents the main vision of the Fog Computing. Note that there is a layer between the physical devices and the upper layer. The Fog level is composed by multiple nodes responsible for some computing and storage functionalities in order to handle the dependency that the bottom layer has with the upper layer [Chi16].

So comparing the Fog architecture with a traditional one we can see that the Fog approach carries out the data nearest the end-user instead of using infrastructure data centers. It carries out the communication nearest the end-user instead of depending on the services provided by the upper layers. Lastly, it carries out the control and management functionalities nearest the end-user instead of depending on specific gateways. In general the core of a traditional view is located at the upper layer and the bottom layers use the services provided. In a Fog view the core is located at the bottom layers [Chi16].

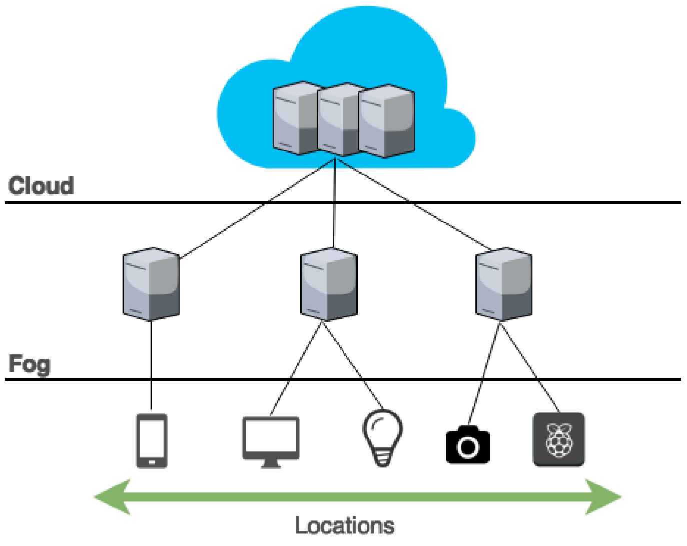


Figure 4.11: Fog Computing

Table 4.4 shows the differences between the Fog nodes and the Cloud in terms of response time and storage availability. In short, we should take advantage of this concept when building an IoT solution where Fog nodes will be helpful to minimise the latency, conserve network bandwidth, operate reliability, and move the data to a better place for processing when needed.

Table 4.4: Fog comparison with cloud

	Fog Nodes near devices	**Fog aggregation nodes**	**Cloud**
Response time	Milliseconds	Seconds to minutes	Minutes, days, weeks
Storage time	Transient	Hours, days, weeks	Months, years
Geographic coverage	Local	Wider	Global

4.7 Building Automation System enhanced by Internet of Things

As the IoT grows, may surge the concern of what role the BAS will play in this new revolution where more and more devices and information exists. Many BAS manufacturers do not believe that BAS will disappear or weaken due to the arise of IoT technologies. Instead, the manufacturers believe that the BAS will see their capabilities and functionalities enhanced [Tat16].

Traditional BAS solutions have high costs associated, making it difficult for small or medium sized buildings to adopt these solutions. These solutions can have high costs due to the closed, custom, and property nature of these systems, which gives the equipment manufacturers full control over installations and upgrades. The IoT solutions are changing this paradigm by enabling solutions that make buildings more efficient and economical to operate. So, bringing the IoT technologies to building automation allows to dramatically reduce BAS costs [Cor14; Ker+16].

Building automation is using data analytics to improve building performance. So, it is possible to see a trend that BAS is moving from building level to enterprise level and eventually to the cloud [Att+16; Moh+16; Zha+16].

As we saw earlier, a BAS architecture is composed by 3 layers. We also saw that usually an IoT solution is composed by 5 layers. It is possible to notice that there are similarities between the layers of each of the architectures. The BAS bottom layer (Field level) is composed by the field devices, such as in the IoT architecture bottom layer (Perception layer). The BAS middle layer (Automation level) is responsible for decision-making, such as the Middleware layer present in the IoT architecture. The top layer (Management level) of the BAS architecture, represents the system configuration, the tools for data visualization and data storage, which is similar to the upper layers of the IoT architecture. Since the IoT platforms can support the BAS requirements and functionalities, and taking into account the case study mentioned in chapter 3, it is possible to implement a BAS using an IoT platform.

4.8 Summary

To choose the best approach, taking into consideration our case study, it was necessary to study the existing technologies. This study involved a rigorous analysis of multiple approaches to IoT architectures and platforms. As a result we present the common and divergent key aspects between these approaches, being possible to observe that there are several aspects that can influence in the choice of the solution. Lastly, we presented the concepts of IoT middlewares and Fog Computing, how they relate to IoT solutions and how they can help in the challenges faced by traditional solutions. The result of this chapter helped us to choose the best approach considering our case study and to realise that there are some differences that imply tradeoffs for someone who wants to build an IoT system. Thus, the analysis from this study creates a comparison guide for future platforms.

Chapter 5

Comparison Review

This chapter describes in more detail the selected platforms. It starts by presenting the differences in the process of connecting new devices to the platform (section 5.1). Then, it describes the tools provided for Data Processing and Visualisation (section 5.2). Then, it presents how the platforms handle Application and User Management (section 5.3). In the end it is presented a detailed discussion about the platforms (section 5.4).

In this paper, we will evaluate the major significant differences between the selected platforms in the previous chapter, namely a platform that we have full control and a PaaS. Thus, taking into account the analysis from Chapter 4, we choose two platforms. The platform we opted to represent the PaaS scenario was the IBM Watson, and the other the WSO2 platform. We choose these two platforms because in addition to have a similar architecture, at layer level, they also provide the features we need for our case study. Additionally, in terms of data analytics, these two are the ones that offer a solid basis for future applications. Another reason that made us choose these platforms was that in comparison with the others, these offer a richer documentation with examples of applications and how to build a complete solution.

Initially, to provide a good comparative analysis of these two platforms, we developed a prototype in both platforms with the minimal functionalities for our case study. The next sections of this chapter explain the process in each phase, the differences found between the platforms and in the end we describe in more detail the advantages and disadvantages of each one. We also clarify that we analysed these platforms having in mind their analytics mechanisms by basing our opinion solely on the available documentation. In Chapter 6 we will explain why we decide to implement the control rules independently from the platform. However, the comparative analysis made about the analytics mechanisms will be useful for future applications, like pattern and behaviour recognition about users and devices.

5.1 Devices Management

In this section, we want to demonstrate the differences in the process of creating and connecting a new device to the platform.

To add a new device, we need to ensure that the platform knows what type of device we want to connect. Firstly, it is necessary to create a Device Type that represents our device. The Device Type is an abstraction of the device and allows us to specify what are the properties and operations of a device.

After the Device Type is created, it is possible to create a device of the respective type, the Device Model, that specifies the values of the properties present in the Device Type. Usually, after a device is created the platform provides the data required to connect the device.

Connecting to the device allows us to send data or commands to the respective device appliance. In our case we use Agents, software used to perform actions on behalf of another component. Agents are the bridge between the physical device and its model, it communicates with the device to collect data, and send it to the platform, or to send commands to the device (Figure 5.1 exemplifies the relationship between this components).

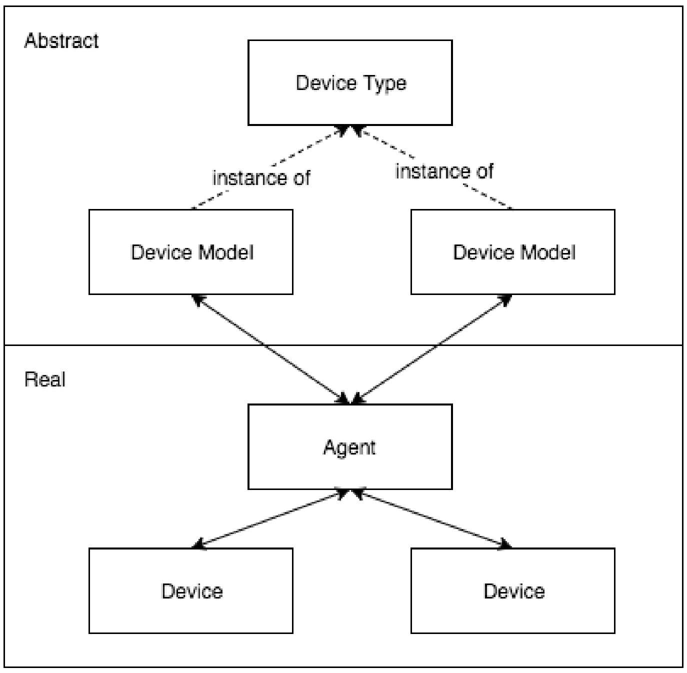
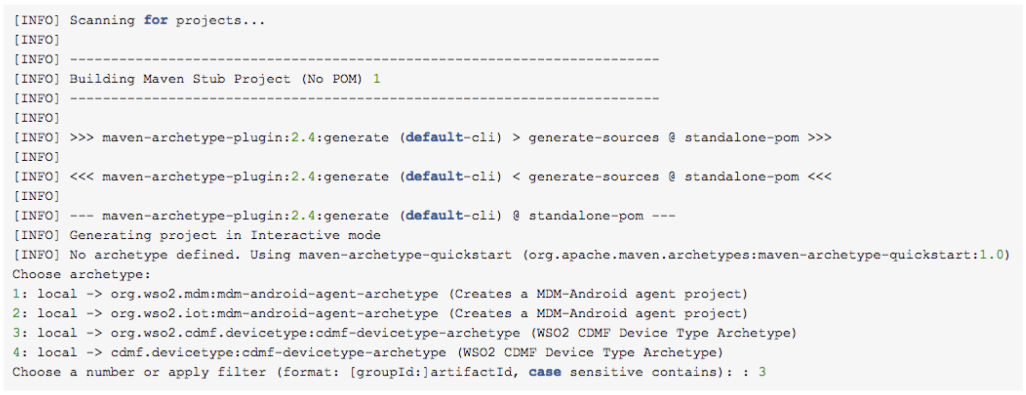


Figure 5.1: Relationship between device type, model and the physical device. Device Type acts like an interface and the Device Model instantiate it. The Device represents the physical device that connects to their model, through the Agent software.

WSO2

To create a new Device Type in the WSO2 platform, we use Maven Archetype toolkit (represented in Figure 5.2). This tool provides a project template and generates a sample project based on the provided properties values [Mav]. The generated code is an OSGi module, called Device Plugin that represents the new Device Type.

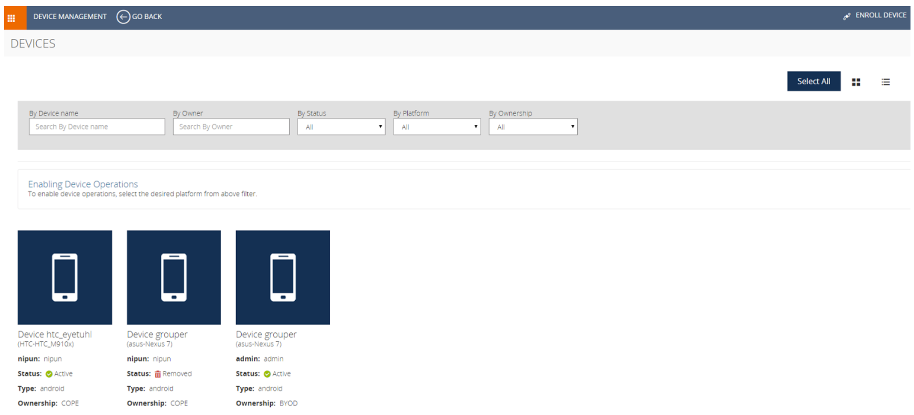


```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
Choose archetype:
1: local -> org.wso2.mdm:mdm-android-agent-archetype (Creates a MDM-Android agent project)
2: local -> org.wso2.iot:mdm-android-agent-archetype (Creates a MDM-Android agent project)
3: local -> org.wso2.cdmf.devicetype:cdmf-devicetype-archetype (WSO2 CDMF Device Type Archetype)
4: local -> cdmf.devicetype:cdmf-devicetype-archetype (WSO2 CDMF Device Type Archetype)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 3
```

Figure 5.2: WSO2 create device type (adapted from [Inc16a])

To connect the module to the platform, it is required to edit a configuration file, that specifies the installed modules, and installs it through Maven. To specify the Device Type properties and operations, it is required to change the source code of the generated code [Inc16c].

After the Device Plugin is installed, it is possible to add a new device in the platform through the WSO2 platform dashboard UI (represented in Figure 5.3). To add a new device, it is required to fill a form with the fields that represent the device properties. This process results in a new Device Model added and a set of files downloaded. The downloaded files contain an example program of how to connect the device and a configuration file that contains the device unique identification and the properties required to connect the physical device with its model [Inc16b].



Enabling Device Operations
To enable device operations, select the desired platform from above filter.

Device htc_eyetuhl
(HTC-HTC_M910x)
nipun: nipun
Status: Active
Type: android
Ownership: COPE

Device grouper
(asus-Nexus 7)
nipun: nipun
Status: Removed
Type: android
Ownership: COPE

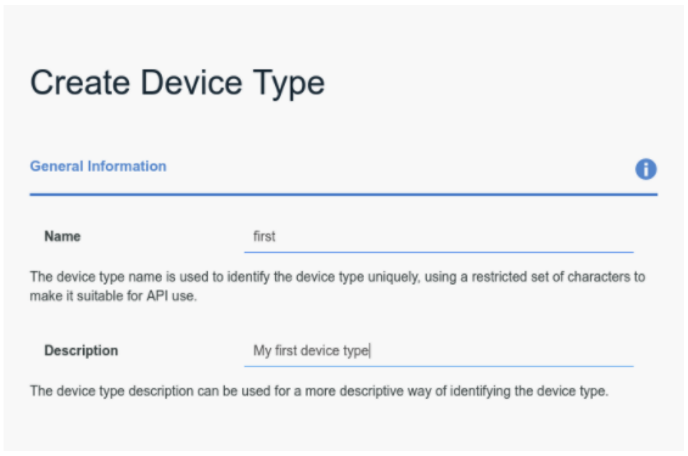
Device grouper
(asus-Nexus 7)
admin: admin
Status: Active
Type: android
Ownership: BYOD

Figure 5.3: WSO2 enrol device (adapted from [Inc16a])

Since WSO2 platform does not provide libraries that helps to connect the devices to the platform, it is up to the the developer in charge to develop the Agent code, based on the example and the configuration file provided.

IBM Watson

To create a new Device Type in IBM Watson platform, we use the Dashboard UI (represented in Figure 5.4) and we need to fill a web form specifying the device properties. A set of predefined properties are presented, and for detailed properties we can provide a metadata object represented in JavaScript Object Notation (JSON) syntax.



Create Device Type

General Information

Name: first

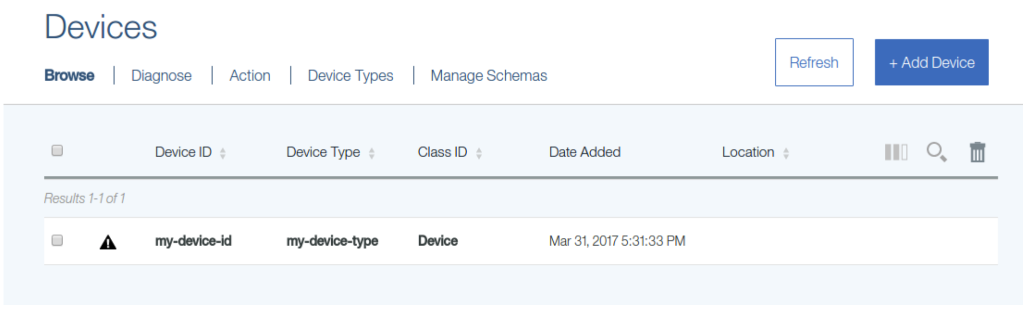
The device type name is used to identify the device type uniquely, using a restricted set of characters to make it suitable for API use.

Description: My first device type

The device type description can be used for a more descriptive way of identifying the device type.

Figure 5.4: IBM Watson create device type (adapted from [Wate])

After the Device Type is created, it is possible to create a new device model, through Watson Dashboard UI (represented in Figure 5.5), that specifies the property values defined by the Device Type. This process results in a new Device Model added, and the required information to connect the device to the platform is presented [Wate].



Results 1-1 of 1

Device ID	Device Type	Class ID	Date Added	Location
my-device-id	my-device-type	Device	Mar 31, 2017 5:31:33 PM	

Figure 5.5: IBM Watson enrol device (adapted from [Wate])

The IBM Watson platform provides a large variety of libraries in different programming languages that helps to connect the devices to the platform. In this case, it is up to the developer to choose the preferred tool and use it to develop the Agent code [Watc].

5.2 Data Processing, Analytics and Events

In this section, we will explain the differences found on the tools provided by the platforms for data analytics. As we mentioned above in Chapter 4, these platforms provide mechanisms for data analytics based on real-time and persisted data.

Usually an IoT platform follows a workflow for data analytics that consists in the phases represented in Figure 5.6.

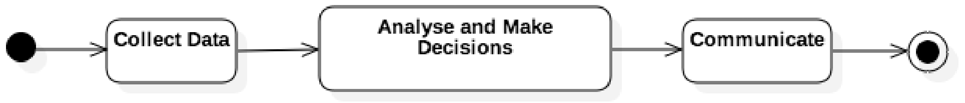


Figure 5.6: Data Analytics Workflow (adapted from [Inccef])

The first phase its about collecting data that comes from different sources, for example, an agent software publishing data to a topic, about device status. The second phase consists in analysing the collected data to produce meaningful information. Lastly, these data should be available through different mechanisms, such as visualisation and event notifications.

WSO2

The WSO2 component responsible for data analytics is the Data Analytics Server. This component provides multiple mechanisms to allow users to extract knowledge about the data analytics. The following list presents the data analytics mechanisms WSO2 provides and how they work.

- **Interactive Analytics**: Uses the Apache Lucene Query Language and all the data must be prepared and organised taking into account the queries to be performed. Therefore, we have to configure our data tables and columns indexed according to the queries to be performed. This configuration is performed through the UI provided by the Data Analytics Server component [Incceb; Inccec; Incced; Inccef].

- **Batch Analytics**: Uses Spark SQL as engine, being possible to create scripts to execute scheduled and interactive queries.The scripts can be created through the UI provided by the Data Analytics Server component [Inccec; Inccef].

- **Real-time analytics**: Uses Siddhi Query Language, designed to process event streams to identify complex event occurrences. "Events" is an object associated with only one event stream, and is composed by a timestamp and set of attribute values. Event streams are logical series of events ordered in time [Inccec; Inccef].

- **Execution Plans**: In order to analyse the event streams it is required to create Execution Plans. The Execution Plan is used to store event processing logic and are composed by a set of queries and related input and output event streams. To create an Executing Plan we must specify its unique name, the input and the output streams identifiers and the query in Siddhi Language. As a result, every time an event occurs on the specific event stream, the execution plan is triggered, the query is executed and the result is sent to the specified output event stream [Inccec; Inccef].

- **Predictive Analytics**: Uses the WSO2 Machine Learner component, that provides a user friendly wizard like interface, which guides users through a set of steps to find and configure machine learning algorithms [Inccec; Inccee].

IBM Watson

IBM platform provides multiple services for data analytics that users can choose according to their needs.

Users ask questions about their data in order to produce meaningful information. Therefore, IBM Watson platform provides services that allows the user to ask questions about their data. The Watson Analytics service request the user to select a data source through an UI. The data source can be a local file in Comma Separated Values (CSV) format, an online service for hosting data, or a external database system. Once the data source is configured, the user can configure the required questions. Lastly, through an UI the user select the values and properties to be evaluated. This process results in a dashboard with multiple interactive views taking into account the properties selected by the user [Wata]. The following list presents the data analytics mechanisms IBM Watson provides and how they work.

- **Streaming Analytics**: Provide tools to ingest, analyse and correlate data in real-time as it arrives from different sources. Allows to process huge volume of data in motion, and helps to find opportunities and risks across the data. Supports data from different sources and formats such as, unstructured text, video, audio, geospatial, sensor, and uses the data to make decisions. IBM also provides a complete solution with a development environment for stream analytics [Comceg].

- **Decision Optimization**: Service that helps to solve real-life problems, by providing tools to make decisions about a goal, or to maximise profit or reduce cost, while satisfying a set of constraints and limitations. This service run on the cloud by adopting mathematical or constraint programming, and is accessible using APIs or the Software Development Kits (SDKs) provided by IBM [Comceb].

- **Geospatial Analytics**: Keep track of IoT devices location on the move opens a number of new applications and opportunities. This service is used to monitor when devices enter or leave specific locations, or to calculate for how long a device was in a specific location. The service uses a message broker allowing the devices to publish information continuously through the MQTT protocol [Comcec].

- **Graph**: Analyse enormous quantities of data and the relationships between these data using traditional relational and tabular methods can be a complex task. Thus, this service simplify this process by representing the data as a set of notes and the relationships between these data as vertices and each node can have attributes associated. It enables to answer questions about large networks of interrelated data [Comced].
- **Machine Learning**: Service that provides a set of REST APIs that can be called from any language. It allows the development of predictive models, or to make smarter decisions and solve tough problems [Comcef].

5.3 Application and User Management

As mentioned in Chapter 4, these platforms provide the mechanisms required for user and application management.

User management is the platform capability to manage entities in order to allow the resources to have different authorization levels by different entities. The entities may have different roles and permissions. Therefore, the platform must provide the required mechanisms to manage the existing entities roles and permissions. Furthermore, it should guarantee that the entities only access the authorized resources [Comcee; Incceh].

Application management is the platform capability to enable applications management during their life cycle. This goes from the initial development process and deployment to monitoring.

Figure 5.7 illustrates the life cycle of an application developed in the owner's perspective. An application life cycle is composed by several activities. Initially the "Develop Activity" is where the application actually developed or modified and made available. The "Publish Activity" its where the application properties are configured, such as, its security requirements, access rates and limitations, and its certificates. "Manage Activity" consists on application keys management, their policies and versions. Lastly, the "Monitor Activity" consists of collecting information about the application behaviour, its usage and user feedback evaluation [Comcea; Incceg].

In the other hand, we have the consumer's perspective. Figure 5.8 illustrates an application life cycle of an application in the consumer perspective. The initial activity, "Find" consists in searching for applications by filtering tags and rating. "Explore activity" consists of evaluating the other user's feedback, ratings or ask questions to application's owner. "Subscribe Activity" its where the consumer registers the application for use, and subscribes the provided API and in turn receives its application and authorization keys. Lastly, "Evaluate" its when the consumer provide feedback and rate the application and shares its experience or provide suggestions [Comcea; Incceg].

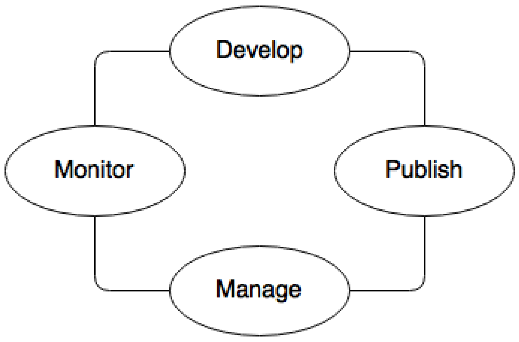


Figure 5.7: Application owner life cycle perspective (adapted from [Incceg])

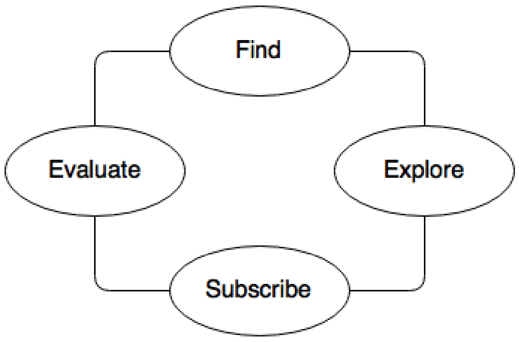


Figure 5.8: Application consumer life cycle perspective (adapted from [Incceg])

Both platforms provide a dashboard for user management, where an authorized user has an overview view of the system. Listing all the existing users, and their activity is possible. The platforms dashboard allows to create new users, with specific roles and permissions previously created, or to modify and remove users permissions. In the dashboard, it is also possible to create or edit system's roles and permissions. In both the platforms a user can represent different entities that interact with the system, such as a device, another program or system. The platforms store the users and their permissions in a database that can be accessed by all the components of the platform through identity and access components, for further authorisation validation [Comcee; Incceh].

The platforms also provide a dashboard for application management, where an authorized user can publish or subscribe applications. For a user with permission to manage applications, the dashboards provide the functionalities to create or configure applications. It is possible to manage access levels, APIs limit usage, application visibility, and the applications keys for both production and sandbox environment. Any authenticated user has permission to search and subscribe to the existing applications [Comcea; Incceg].

The platforms also provide the capability to manage users and applications through a REST API. However, only the IBM Watson platform provides multiple SDKs in different programming languages that simplify the process to build external applications that may use these management functionalities.

5.4 Discussion

In this chapter, we reflect on all the aspects from an early stage of development of the agents required to communicate with the devices and platform, to the final phase of deployment. Both platforms were able to provide the required tools for a prototype implementation of a minimal set of functionalities. This prototype was implemented with the functionalities to store and control devices state. Thus, we create two device types, and added 2 devices of each type for both platforms. It was stored data about the devices state. In what respects to devices control, it was built a sample application to control the devices through a REST API provided by the platforms. The sample application allows the devices manual control or schedule control that turn on or off the devices at morning or at night. We will enumerate some aspects we found relevant to this process using the selected platforms.

The platforms approach the process of creating new device types in a very different way. As mentioned above, WSO2 uses Maven tool to create a module that represents a device type, while IBM Watson provides a form to specify the device type properties. It is easy to see that this process is more complex using the WSO2. However, WSO2 ensures that all devices are according to its type, unlike IBM Watson. A device type is represented as an OSGi module in WSO2, and it is required to stop the system every time we need to add, edit or remove one module. In IBM Watson a device type is represented through JSON, and it can be modified any time without stoping the system. However, editing device types on IBM Watson may cause issues on existing devices. Thus, it requires special attention when changing a device type. In WSO2 as it needs to stop the system to install or edit modules, the module installation process ensures whether the specified properties respect the existing devices, notifying with an error otherwise. Additionally, IBM Watson provides functionalities to add, edit and remove devices types through a REST API, unlike WSO2 that does not support these functionalities.

Having the device types created on both platforms, we can start adding the devices of each type. On both platforms this process can be done by filling a form. In WSO2 the form shows exactly the properties specified by the device type previously created. In IBM Watson the forum shows the properties specified by the device type and allows to add extra fields to a specific device. Note that in IBM Watson we can add extra fields to a device, allowing each device of the same type to have different properties. However, adding extra fields to specific devices may lead to issues when creating an external application for devices control, because the devices fields of the same type are not consistent. This issue does not happen in WSO2 because the platform ensures that devices of the same type have the properties. Additionally, both platforms provide a REST API to create, edit and remove devices of the platform. It is also important to mention the lack of functionality by IBM Watson to create groups of devices. Otherwise, WSO2 allows the creation and management of devices groups either through a dashboard or REST API.

Once the devices are added to the platforms the next step is to connect the physical devices with the platform. In this phase it is required to get the access credentials for each device, and implement the software for the agent that will handle with the commutation between the physical device and its model. To implement the agent software used to communicate with WSO2, the developer in charge must deal with all the required code, since WSO2 does not provide SDKs. Thus, the developer must handle the communication with the physical devices, the authentication process and the publication of data by himself. It is a time-consuming process because usually, an agent is in charge of multiple devices of the same type. Thus, it must adopt mechanisms to handle multiple processes. The software agent code for WSO2 is not part of this dissertation, so it was reused code of another work in development. In the case of the development of software agent code for IBM Watson is simplified through the SDKs provided. Thus, the developer just needs to handle with the physical device communication, since the platform communication is encapsulated in methods provided by the SDKs.

For devices' credentials WSO2 platform provide a configuration file when the device is added to the platform. The file contains the unique device identifier, the platform endpoints to be used by the software agent to publish devices data and subscribe to events. The configuration file also contains the tokens required for device authentication on the platform, in order to allow him to publish data to its specific endpoints. Additionally, the agents must handle the process to refresh device authentication tokens. Again, this process is simplified by IBM Watson. It allows to generate device credentials after adding the device to the platform, that will be used by the agents to authenticate the device. So we can conclude that IBM Watson simplifies the agent development process, handling with the communication between the physical device and the platform. These platforms present many differences taking into account the credentials used to authenticate the devices on the platform. WSO2 forces tokens to be refresh within a specified time. IBM Watson generate tokens with an expiration date, and does not allow to refresh tokens validation. This means that the system administrator has to generate new tokens manually, and the refresh token process can not be automated. Additionally, IBM Watson allows that one and only one device with the same credentials can be authenticated at the same time. Otherwise, WSO2 allows multiple authentication with the same credentials, being possible to publish data from different sources. Thus, the system administrator must ensure that the data published for each device is consistent between all the sources. Lastly, to publish data about the devices state and subscribe to events, both platforms support the MQTT protocol, but IBM Watson also allows these functionalities through a REST API over the Hypertext Transfer Protocol (HTTP) protocol.

One important aspect for those choosing an IoT platform, is the volume of data that flows through the network. So, we used the WireShark tool in order to evaluate the volume of data flowing on the network when communicating with these platforms. Since the devices specification are different in both platforms implies that the volume of data are different. The WireShark tool allowed us to evaluate the data flowing between the agent to the platforms from the moment of device authentication to the first time the agent publish data about the device state. We notice that the authentication process by an agent that communicates with IBM Watson has a volume of data of approximately 4536 bytes. This same process but communication with WSO2 has approximately 3025 bytes. In both cases, the data is the sum of all requests and responses between an agent and the platform from the initial request, to credentials verification and authentication confirmation. The process to publish data about a device state by an agent that communicates with IBM Watson has a volume of data of approximately 423 bytes, while the same process communicating with WSO2 has a volume of data of approximately 1056 bytes. Despite the published data of the devices has equal values, the big difference on the volume of data is due to the fact that WSO2 uses a single endpoint for each device property, while IBM WAtson uses the same endpoint for all the properties.

After devices are created and communication is established, it is important to know if platforms are receiving the data correctly and they can send a control command to the devices. Both platforms provide a dashboard where it is possible to check existing devices and their status in real-time. WSO2 presents a chart for each device property, and the chart data is updated whenever the platform receives a new value. IBM Watson by default does not show any chart, because it is up to the user to choose which type of chart to use for a given device property. There is a difference in the platforms' dashboards for device control. WSO2 dashboard provides the functionalities to control a device property manually. Otherwise, due to the fact that there is no uniformity in the properties of the created devices, IBM Watson does not provide this option. However, both platforms provide the required tools to control the devices through a REST API by specifying the device unique identifier, the property identifier and the new value.

To develop an application that can communicate with the platform in order to obtain data and control the devices, it is required to create an application in the platform. This application is created by the system administrator and it can be done through the platforms dashboard. The system administrator specifies the application name and subscribe to a set of provided APIs. Then it is required to generate authorization keys with scopes that define the permissions of the created application. Furthermore, these keys will be used in the modules to be developed. The process of application management is available through the platforms dashboard and also through platforms REST API.

To close this comparative analysis about the platforms, we will note some relevant points that we consider relevant for the development of an IoT system. Not focusing specifically the WSO2 and IBM Watson platforms, but what they represent, that is an open-source platform that gives us control of our system and all the process depends on us and a platform with the characteristics of a PaaS. In an open-source platform, a greater effort is required not only for initial setup but also on deployment, always to have someone designated to maintain the system and its resources. Otherwise, a PaaS the resources management is handled by the service host. Another question is the platforms updates, that was a point that cause some delays in the work of this dissertation. As we implemented the solution, some issues were discovered on the WSO2 platform. Another users already reported these issues and they were being fixed by the WSO2 platform developers. Since some of our components were already dependent from the current platform version, it was not possible to upgrade to a new version, as it invalidates some of the components already developed. In a PaaS usually, the upgrades are transparent to its users. Another aspect to be taken into account when developing an IoT solution is whenever a platform provides SDKs for application development. In our case, all the implemented code to communicate with IBM Watson was always more accessible and required less development effort compared to the code developed to interact with WSO2, since it does not provide SDKs. Later this can create issues on the components that interact with WSO2, because a change in the platforms requests parameters may invalidate some methods of the implemented components. Otherwise, using the platform provided SDKs, usually minimises these issues since the methods are encapsulated. Another consideration when construction an IoT solution is the documentation provided by the chosen platform. We notice that although there are many available solutions, only a few offer rich documentation and practical examples. Both WSO2 and IBM Watson presented reasonable documentation. However, the documentation provided by WSO2 had several flaws which caused some issues. Another point in favours a PaaS is the integration of external services. To integrate external services, IBM Watson simplifies this process by requiring only to select the service and configure authentication methods for the selected service. Regarding the initial setup and the deploy phases, there are a few differences. As expected, in a PaaS this process is simplified and the deploy of the final solution is as simple as just changing the solution to production mode. If the whole development process goes as expected, then at the end we have a solution ready to go. An open-source platform where it is required to prepare all the infrastructure where it will be deployed, there are some aspects to take into account. We have to take into account the specific hardware requirements, configure accesses, databases, and handle possible network configurations. Another important aspect is the lack of communication in some environments. In our case study, the loss of access to the Internet is an event that happens regularly. Therefore, using a PaaS implies the loss of communication with the platform, and t is necessary to adopt mechanisms to deal with these flaws. In the case of a platform located on our local network, the loss of Internet signal does not imply that the components can not communicate with the platform, since they are in the same network as the platform. It is also important to mention that in deployment process requires opening some specific ports, namely for database services, port for communication with the MYSQL protocol for example, and for Message Broker services over the MQTT protocol. In some scenarios, opening local ports may not interfere, but as a PaaS, opening ports for external communication can be a requirement not possible to satisfy. Lastly, the cost of these solutions lies in different aspects. In a PaaS the cost is associated with service usage, the growth of the volume of data and services consumption rates. However, the cost associated with the development and deployment phases is simplified, since it does not require to handle with infrastructure and resources configuration and maintenance. Otherwise, the cost of an open-source solution that is located in our infrastructure, requires configuration and maintenance costs.

As a summary, Table 5.1 presents the points where these platforms diverge.

Table 5.1: Platforms Comparison Overview

	WSO2	**IBM**
Create Devices	Requires Maven Archetype tool. Edit configuration file and source code for additional properties.	Web form based.
Device Properties	All devices of the same type have the same properties.	Devices of the same type may have additional fields.
Devices Group	Allows to create and manage groups of devices	Does not provide device group features
Device Agents	Does not provide libraries. Developer deals with entire code.	Provide multiple libraries in different languages. Developer can choose what tools to use.
Publish and Subscribe	MQTT	HTTP and MQTT
Security	Random generated tokens. Requires refresh tokens mechanisms.	Random generated tokens with expiration date. Administrator handles tokens renewal.
Data Visualisation	Real-time charts for every device property.	User must select the types of charts and devices properties.
Device Control	Allow device control through dashboard.	Does not allow device control through dashboard.
Application Development	Does not provide libraries. Developer deals with entire code.	Provide multiple libraries in different languages. Developer can choose what tools to use.
Setup and Deploy	Configure system infrastructure. Setup databases and network ports.	Setup network ports.
Costs	Infrastructure configuration and maintenance.	System growth and usage rates.

5.5 Summary

The analyses made in the previous chapter and the fact that we have observed certain characteristics that may influence the platform choice, made us create this chapter. To make this document more complete, we chose two platforms with different development and deployment environments and similar architectures and functionalities. We also implemented a prototype with a set of basic functionalities to collect data and control devices state. The differences and the challenges found in this process helped us to create a guide that will allow someone who wants to create an IoT solution to make a better decision.

Chapter 6

Conceptualisation and Implementation

This chapter describes our work on the case study, from the requirements to the implementation of a concrete solution. It starts by presenting the conducted requirement analysis, describes the stakeholders, the user's stories, the functional and non-functional requirements (section 6.1). Then it details the user stories using use cases (section 6.2). In the sequence, it presents the architecture styles and views selected (section 6.3). Finally, it ends by showing the result of the implementation phase (section 6.4).

6.1 Requirements Engineering

Requirements Engineering is the process of finding, analysing and documenting the requirements a system should fulfil, the services and the constraints on its operations [Som10]. Requirements identify the capabilities of a system to resolve a given problem in a given context. They also express the needs of the involved entities and define the system goals and services. Moreover, the requirements can be classified into two categories [Som10]:

- **User requirements**: Statements express in natural language that describe the services the system should provide to the users and its constrains;
- **System requirements**: A more detailed description about the system operations and services to be implemented and its operational constraints.

The following sections describe the process conducted in order to understand what are the entities that interact with the system, what they expect from it and a detailed description of what services the system should provide.

6.1.1 Stakeholders

In Software Engineering the Stakeholder is an individual, team or organization that have interest in the realization of the system [Soc+14]. In order to describe the requirements we start by identifying the entities that interact with the system and what they expect from it. The following list presents the stakeholders found during the analysis:

- **Administrator**: This group comprises those responsible for system configuration, users and devices management;
- **User**: This group comprises those concerned about reducing energy consumption and maintaining a good working environment;
- **Occupant**: This group comprises those who interact indirectly with the system. Acts like an occupant and helps the system in manual tasks that cannot be accomplished with automating tools.

Note that these stakeholders are just an abstraction of roles and permissions of users in a specific context or action. This means that an Administrator can sometimes act like a User or an Occupant and vice versa.

6.1.2 Questionnaires

Questionnaires are a traditional technique used in Software Engineering for requirements elicitation. This technique has some advantages to gather requirements from the stakeholders [Soc+14].

Firstly, we elaborate a questionnaire (can be found at Appendix A) focused on the occupants, based on two studies conducted by [HA97] and [Cae+17]. This questionnaire was attended by 5 occupants of the office, and our goal is to understand what physical aspects are more pleasant for them in a workplace, understand if they were happy with the current environment in the office and how it can be improved.

The questionnaire results were analysed and they helped us to understand some aspects the occupants would like to be changed and what aspects are the most important for them in a work area. Furthermore, these results will be presented and analysed in more detail in Chapter 7.

6.1.3 User Stories

User Stories (US) is an elicitation technique used in Software Engineering to describe in natural language the required functionalities a system must provide expressed in user terms [Soc+14].

Firstly, we discovered what were the expected functionalities and the points of interest for the mentioned stakeholders. These functionalities were analysed and described in the form of US. Then, in order to simplify the understanding of the US found, we give them a unique identifier and divide them in several categories presented in the following sections.

User Management

This group of US represents the management functionalities provided by the system for user management.

- **US-UM1**: As an Administrator, I want to see a list of all registered accounts;
- **US-UM2**: As an Administrator, I want to create new user accounts with specific roles;
- **US-UM3**: As an Administrator, I want to be able to remove existing accounts;
- **US-UM4**: As an Administrator, I want to manage the permissions of a given role;
- **US-UM5**: As an Administrator, I want to create new roles with specific permissions.
- **US-UM6**: As a User, I want to authenticate in the system, so that I can have access to protected resources.

Device Management

This group of US represents the management functionalities provided by the system for device management.

- **US-DM1**: As an Administrator, I want a list of all supported device types;
- **US-DM2**: As an Administrator, I want a list of all registered devices;
- **US-DM3**: As an Administrator, I want a list of all devices within a group;
- **US-DM4**: As an Administrator, I want to add a new device, so that it can be used by the system;
- **US-DM5**: As an Administrator, I want to add devices automatically from a file, in order to simplify the process of adding a new device;
- **US-DM6**: As an Administrator, I want to edit device properties, so that it can represent the real values;
- **US-DM7**: As an Administrator, I want to remove a device, in order to disable a non existing device;
- **US-DM8**: As an Administrator, I want to assign devices to an existing group, or create a new one;

- **US-DM9**: As an Administrator, I want to remove devices of a group, or delete the group;
- **US-DM10**: As an Administrator, I want to visualise the current and historical data of all devices;
- **US-DM11**: As an Administrator, I want to mark a device as critical to ensure that it is not turned off;
- **US-DM12**: As an Administrator, I want to backup all devices data, in order to keep devices data secure in case of system failure;
- **US-DM13**: As a User, I want to manually control a device, in order to override system commands.

API Management

This set of US represents the management functionalities provided by the system for creating and managing applications.

- **US-AM1**: As an Administrator, I want to create a new application, in order to subscribe to existing APIs;
- **US-AM2**: As an Administrator, I want a list of all supported APIs;
- **US-AM2**: As an Administrator, I want a list of all registered applications;
- **US-AM4**: As an Administrator, I want to generate new access tokens with specific permissions, so that they can be used by different users;
- **US-AM5**: As an Administrator, I want to be able to remove an application, so that it can not be used;
- **US-AM6**: As an Administrator, I want to be able to unsubscribe an API, so that the application can not access its resources;
- **US-AM7**: As an Administrator, I want to be able to revoke an access token, so that the users can not use it any more.

Occupant Comfort

This set of US represents the expected system behaviour that helps to provide a better comfort to the occupants.

- **US-OC1**: As an Occupant, I want the lights to adapt their brightness and colour, in order to provide a better visual comfort;

- **US-OC2**: As an Occupant, I want the air conditioning to be adapted, in order to achieve a better thermal comfort;
- **US-OC3**: As an Occupant, I want the space with clean, fresh and circulated air, in order to improve the air quality;
- **US-OC4**: As an Occupant, I want the space with an adequate level of noise, in order not to be distracted.

Presence Detection

This group of US represents the expected behaviour in case of presence detection or inactivity for a certain amount of time.

- **US-PD1**: As a User, I want the lights to turn off when there is no activity, in order to reduce energy consumption;
- **US-PD2**: As a User, I want the air conditioning system to turn off when there is no activity, in order to reduce energy consumption;
- **US-PD3**: As a User, I want the uncritical devices to turn off when there is no activity, in order to reduce energy consumption;
- **US-PD4**: As an Occupant, I want the outlets to turn on when I arrive, so that I can start working;
- **US-PD5**: As an Occupant, I want the lights to turn on when I arrive and if the luminosity is low;
- **US-PD6**: As an Occupant, I want the coffee machine to turn on when I arrive, so that it can be prepared to serve coffee;
- **US-PD7**: As a User, I want the system to detect human presence adopting multiple mechanisms.

Scheduling

This group of US represents the functionalities provided in order to schedule events.

- **US-SC1**: As a User, I want to schedule the time to feed the fishes;
- **US-SC2**: As a User, I want to schedule the time to turn on/off a device, in order to reduce the energy consumption;
- **US-SC3**: As a User, I want to schedule the time to turn on/off a group of devices, in order to reduce the energy consumption;
- **US-SC4**: As a User, I want to edit the time of a previous scheduled time;

- **US-SC5**: As a User, I want to remove a scheduled time, so that it no longer take effect;
- **US-SC6**: As a User, I want the devices to turn on/off at the scheduled time.

Suggestions

This group of US represents the suggestions made to the occupant, in order to help the system in tasks that can not be accomplish by itself.

- **US-SG1**: As an Occupant, I want to be notified to open the window, so that I can take advantage of natural temperature, in order to keep good thermal conditions and help to reduce energy consumption;
- **US-SG2**: As an Occupant, I want to be notified to open the blinds, so that I can take advantage of natural day light, in order to keep a good luminosity level and help to reduce energy consumption;
- **US-SG3**: As an Occupant, I want to be notified to open the window, so that I can keep a good air quality.

Notifications

This group of US represents the functionalities provided in order to notify the user in case of certain events.

- **US-NT1**: As a User, I want to be notified about consumptions out of the ordinary, in order to evaluate if something went wrong;
- **US-NT2**: As an Administrator, I want to be notified about devices that do not change their state over a long period of time, in order to evaluate if something went wrong;
- **US-NT3**: As a User, I want to be notified about important events, so that I can keep track of what is happening;
- **US-NT4**: As a User, I want to be notified about aquarium status, in order to prevent that something bad happens to the fishes;
- **US-NT5**: As an Administrator, I want to be notified about system errors, so that I can fix them as soon as possible.

Aquarium

This set of US describes the required behaviour to keep a good environment in the fish tank.

- **US-AQ1**: As a User, I the want the fishes fed, in order to keep them alive;

- **US-AQ2**: As a User, I want a stable water level in the aquarium, in order to keep a good environment for the fishes;
- **US-AQ3**: As a User, I want the fish aquarium lights to be adapted, in order to keep a good environment for the fishes;
- **US-AQ4**: As a User, I want a stable water temperature for the aquarium, in order to keep a good environment for the fishes.

6.1.4 Functional Requirements

Functional Requirements (FR) are statements that describe the services and resources the system should provide, how the system should behave in particular situations with specific inputs. FR can also define what the system should not do [Som10].

Next we will present what the FR the system should provide taking into account our case study and the users' stories described above. To simplify the understanding of the FR, we give them a unique identifier and divide them in several categories presented in the following sections.

6.1.4.1 User Management

This set of FR describes the functionalities the system should provide in order to manage users, their permissions and roles.

- **FR-UM1**: The system should allow a registered user to authenticate through personal credentials such as username and password;
- **FR-UM2**: The system should allow an authorized user to create new user accounts;
- **FR-UM3**: The system should allow an authorized user to manage the permissions of other accounts;
- **FR-UM4**: The system should allow an authorized user to remove other accounts;
- **FR-UM5**: The system should ensure that exists at least one user with Administrator role. Therefore, it should not be allowed to remove the last user with the Administrator role;
- **FR-UM6**: The system should display a list of the registered users;
- **FR-UM7**: The system should display a list of the supported roles and their permissions;
- **FR-UM8**: The system should allow an authorized user to create new roles and assign permissions.

6.1.4.2 Device Management

This set of FR describes the functionalities the system should provide in order to manage the devices.

- **FR-DM1**: The system should allow an authorized user to add a new device through a form with properties such as, device name, description and position;
- **FR-DM2**: The system should allow an authorized user to edit a device properties through a form;
- **FR-DM3**: The system should display a list of the supported device types;
- **FR-DM4**: The system should display a list of the registered devices;
- **FR-DM5**: The system should allow an authorized user to create groups of devices;
- **FR-DM6**: The system should display a list of the created groups;
- **FR-DM7**: The system should allow an authorized user to remove a group, keeping its devices unassigned of any group;
- **FR-DM8**: The system should display a list of devices of a given group;
- **FR-DM9**: The system should allow an authorized user to remove a device;
- **FR-DM10**: The system should allow an authorized user to mark a device as critical;
- **FR-DM11**: The system should allow an authorized user to add multiple devices at once by importing a CSV file;
- **FR-DM12**: The system should allow an authorized user to change a device state;
- **FR-DM13**: The system should allow an authorized user to consult the current state of a device;
- **FR-DM14**: The system should allow an authorized user to consult historical data of a device state within a specific date and time;
- **FR-DM15**: The system should allow an authorized user to export all data about devices state.

6.1.4.3 API Management

This set of FR describes the functionalities the system should provide in order to manage the provided APIs, the authorized roles and applications.

- **FR-AM1**: The system should allow an authorized user to create a new application through a form with properties such as, name and description;

- **FR-AM2**: The system should allow an authorized user to select an application and subscribe to existing APIs;
- **FR-AM3**: The system should allow an authorized user to define the maximum number of requests per minute an API can support;
- **FR-AM4**: The system should allow an authorized user to generate new access tokens with a specific expiration date and a set of permissions;
- **FR-AM5**: The system should display a list of the permissions available;
- **FR-AM6**: The system should display a list of the supported APIs;
- **FR-AM7**: The system should display a list of the registered applications;
- **FR-AM8**: The system should allow an authorized user to remove an application;
- **FR-AM9**: The system should allow an authorized user to select an application and unsubscribe an API;
- **FR-AM10**: The system should allow an authorized user revoke a previous generated access token.

6.1.4.4 Measurement

This set of FR describes the functionalities the system should perform in order to measure data about its environment.

- **FR-MS1**: The system should collect and store data about office temperature level through indoor temperature sensors;
- **FR-MS2**: The system should collect and store data about outside temperature level through outdoor temperature sensors;
- **FR-MS3**: The system should collect and store data about office luminosity level through indoor light sensors;
- **FR-MS4**: The system should collect and store data about outside luminosity level through outdoor luminosity sensors;
- **FR-MS5**: The system should collect and store data about office air quality through indoor air quality monitors;
- **FR-MS6**: The system should collect and store data about outside air quality through outdoor air quality monitors;
- **FR-MS7**: The system should collect and store data about devices energy consumption;

- **FR-MS8**: The system should only store meaningful data about devices, in order to reduce the size of the stored data.

6.1.4.5 Occupant Comfort

This set of FR describes the functionalities the system should provide in order to keep a good environment for the office occupants.

- **FR-OC1**: The system should turn on/off the lights in order to provide a good visual comfort, taking into account the occupants and the current luminosity level;
- **FR-OC2**: The system should dim the lights in order to provide a good visual comfort, taking into account the occupants and the current luminosity level;
- **FR-OC3**: The system should turn on/off the air conditioning system in order to provide a good thermal comfort, taking into account the occupants;
- **FR-OC4**: The system should regulate the air conditioning system temperature in order to provide a good thermal comfort, taking into account the occupants and the current temperature level;

6.1.4.6 Presence Detection

This set of FR describes how the system should react in human absence or human presence detection.

- **FR-PD1**: The system should be able to detect human presence by evaluating the outlets consumption variation;
- **FR-PD2**: The system should be able to detect human presence by evaluating the entries and exits of the office through a card reader mechanism;
- **FR-PD3**: The system should be able to detect human presence by evaluating the data received by mobile phones in range;
- **FR-PD4**: The system should be able to detect human presence by evaluating the data received by presence detection sensors;
- **FR-PD5**: The system should turn off the lights when there is no activity;
- **FR-PD6**: The system should turn off the air conditioning when there is no activity;
- **FR-PD7**: The system should turn off the outlets at night if there is no activity;
- **FR-PD8**: The system should turn off the coffee machine at night if there is no activity;

- **FR-PD9**: The system should turn on the lights when it detects human presence and if the luminosity is low;
- **FR-PD10**: The system should turn on the outlets when it detects human presence;
- **FR-PD11**: The system should turn on the coffee machine when it detects human presence in the morning.

6.1.4.7 Scheduling

This set of FR describes the functionalities the system should provide in order to enable the schedule of events.

- **FR-SC1**: The system should allow an authorized user to schedule the date and time to turn on/off a device;
- **FR-SC2**: The system should allow an authorized user to schedule the date and time to turn on/off a group of devices;
- **FR-SC3**: The system should allow an authorized user to edit a scheduled rule;
- **FR-SC4**: The system should allow an authorized user to remove a scheduled rule;
- **FR-SC5**: The system should be able to turn on/off the specified devices at the scheduled date and time.

6.1.4.8 Suggestions

- **FR-SG1**: The system should suggest the user to open the window if the indoor air quality is lower than outside;
- **FR-SG2**: The system should suggest the user to open the window blinds if it is possible to take advantage of natural daylight;
- **FR-SG3**: The system should suggest the user to open the window if it is possible to take advantage of outdoor temperature.

6.1.4.9 Notifications

This set of FR describes the functionalities the system should provide in order to alert predefined users about important events.

- **FR-NT1**: The system should notify predefined users when it turns off all the lights;
- **FR-NT2**: The system should notify predefined users when it turns on/off all the outlets;
- **FR-NT3**: The system should notify predefined users when it turns off the coffee machine;

- **FR-NT4**: The system should notify predefined users about devices which are inactive within 1 hour;
- **FR-NT5**: The system should notify predefined users about devices with uncommon energy consumption;
- **FR-NT6**: The system should notify predefined users when it feeds the fishes;
- **FR-NT7**: The system should notify predefined users when the fish tank water level is too low/high;
- **FR-NT8**: The system should notify predefined users when the fish tank water temperature level is too low/high;
- **FR-NT9**: The system should notify predefined users when some unexpected error occurs.

6.1.4.10 Aquarium

This set of FR describes the functionalities the system should provide in order to keep a good environment for the fishes.

- **FR-AQ1**: The system should allow an authorized user to schedule the date and time to feed the fishes;
- **FR-AQ2**: The system should allow an authorized user to edit the scheduled date and time to feed the fishes;
- **FR-AQ3**: The system should be able to feed the fishes at the scheduled date and time;
- **FR-AQ4**: The system should maintain a stable water level on the fish tank by turning on/off the pump water;
- **FR-AQ5**: The system should maintain a stable water temperature level on the fish tank by turning on/off the ventilator;
- **FR-AQ6**: The system should maintain a stable light level on the fish tank by turning on/off the aquarium lights.

6.1.5 Non-Functional Requirements

NFR are not related with specific services provided by the system, but how the system will provide them. These requirements describe properties such as reliability, availability, performance, security, safety, storage and deployment. Therefore they may define constraints on the system implementation, or constraints that affect the overall architecture of the system [Som10]. Since there is no standard for categorizing and listing NFR [Chu+00], we will use the types of NFR proposed by Sommerville [Som10].

The following list presents the NFR the system must ensure, categorized by the types proposed by Sommerville:

Product Requirements: Requirements that specify the behaviour and constraints of the system. May include performance, reliability, usability and security requirements.

- **NFR1**: The system must ensure a moderate response time during its utilization;
- **NFR2**: The system must ensure a moderate response time when reacting to physical events;
- **NFR3**: The system must allow simultaneous access by different users;
- **NFR4**: The system must be secure and resist to attempted attacks by unauthorized users;
- **NFR5**: The system must protect its resources so that they can only be accessed by authorized users;
- **NFR6**: The system must ensure the privacy of users privileged data;
- **NFR7**: The system must ensure that all external communications are encrypted;
- **NFR8**: The system must be fault tolerance in case of misused by users or internal errors;
- **NFR9**: The system must be accurate, reliable and maximise the correct operating time and minimise the recovery time in case of failures;
- **NFR10**: The system must ensure the compatibility and cooperation between its components;
- **NFR11**: The system must be capable of handling the growth of its components;
- **NFR12**: The system must be easily adapted to meet new requirements, correct defects and maximise its lifetime;
- **NFR13**: The use of the system must be intuitive, easy to learn and to memorize by its users;
- **NFR14**: The system must ensure that the users can achieve their goals efficiently and effectively;
- **NFR15**: The system interfaces must be responsive and adapt according to the device used for access;
- **NFR16**: The system must manage the high volume of store data, in order to use the minimum storage space possible;
- **NFR17**: The system must ensure that the scheduled rules are executed, even if due to failure it can not execute the action at the scheduled time.

Organizational Requirements: Requirements driven from policies or procedures in the customer and developer organization. May include development process and environment requirements, and requirements that define how the system should be used.

- **NFR18**: The system must be available during normal working hours, and may be unavailable at night for possible maintenance;
- **NFR19**: The system must calculate the energy consumption taking into account the energy contract agreement established with the institution;
- **NFR20**: The development of the devices' software must ensure that the device state is updated at least hourly;
- **NFR21**: The development of the devices' software must adopt multi-thread mechanisms in order to deal with multiple simultaneous events;
- **NFR22**: The development of the devices' software, whenever possible, must use MQTT protocol for communication instead of HTTP;
- **NFR23**: The development of new devices' software must follow the structure already implemented in other devices, in order to keep the software consistent;
- **NFR24**: The development of the devices' software must ensure that, when the system is down in case of failure or networks issues, the devices data are stored in local memory until system recovers;
- **NFR25**: The developed devices' software must be exported so that it can be portable and executable without requiring to install dependencies.

External Requirements: Requirements driven from external factors to the system. May include legislative requirements, ethical and regulatory requirements.

- **NFR26**: The temperature, luminosity and air quality provided should not compromise the occupants productivity;
- **NFR27**: The temperature, luminosity and air quality provided should not compromise the well-being of occupants;
- **NFR28**: The temperature level must be defined according to the legislation that defines the standards for indoor temperature in work offices;
- **NFR29**: The luminosity level must be defined according to the legislation that defines the standards for indoor luminosity in work offices;
- **NFR30**: The air quality level must be in accordance with the legislation that defines the standards for indoor air quality in work offices;
- **NFR31**: The fish tank's water temperature must be defined according to the standards for water temperature in freshwater fish aquarium.

6.1.6 Requirements Tracing

At this point we described requirements using US and FR, so it is important to relate these two sources. In order to ensure that everything is covered and consistent [AN05].

To handle this, we grouped the US by different tables, and for each user story identifier we specify the matching FR. The result of this process can be found in Appendix B.

6.2 Detailed User Stories

This section detail the most relevant US. Note that we only list the requirements considering our case study, so it is possible to achieve a whole different set of requirements for a different scenario.

Another detail is that there are several requirements, but some of them are already supported by an IoT platform. Therefore, **we will only specify the US focused on automation, the ones that contribute for the reduction of energy consumption, and for improving the comfort for office occupants.**

Firstly we divide the US by sections and for each section we will present a use case diagram, modelled with the Unified Modelling Language (UML) language, followed by a detailed specification of its use cases based on the template suggested in [AN05].

The following identifiers represent the selected US:

- **Device Management**: US-DM5, US-DM11, US-DM12;
- **Occupant Comfort**: US-OC1, US-OC2, US-OC3, US-OC4;
- **Presence Detection**: US-PD1, US-PD2, US-PD3, US-PD4, US-PD5, US-PD6, US-PD7;
- **Scheduling**: US-SC1, US-SC2, US-SC3, US-SC4, US-SC5, US-SC6;
- **Suggestions**: US-SG1, US-SG2, US-SG3;
- **Notifications**: US-NT1, US-NT2, US-NT3, US-NT4, US-NT5;
- **Aquarium**: US-AQ1, US-AQ2, US-AQ3, US-AQ4.

6.2.1 Device Management

This section presents the use case diagram for device management in Figure 6.1 and its use cases specifications. Considering the selected US, the Administrator can mark a device as critical, backup devices data and add devices through a list.

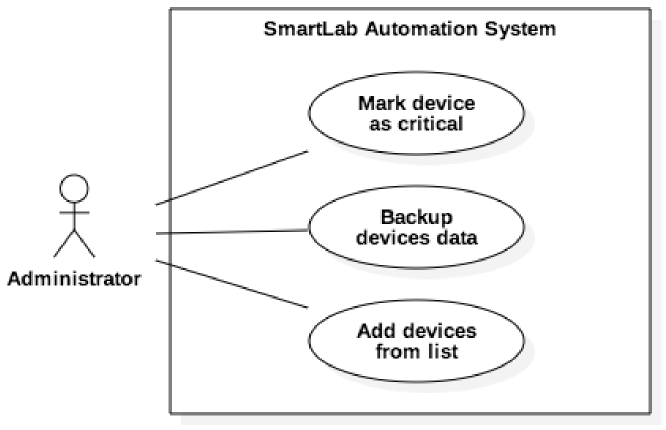


Figure 6.1: Use Case Diagram - Device Management

Use case: Mark device as critical

Brief description: The Administrator marks a device as critical, so that it can not be turned off by the system.

Primary actors: Administrator

Preconditions:

1. The Administrator is authenticated in the system.

Main flow:

1. The use case starts when the Administrator access to the devices page.
2. The system finds and displays all the available devices.
3. The Administrator selects one device of the list and marks it as critical.
4. The Administrator confirms the action.
5. The system marks device as critical.

Postconditions:

1. The device as been marked as critical.

Exceptions:

1. Administrator cancel the use case. The system does not mark the device as critical.

Use case: Backup devices data

Brief description: The Administrator requests a backup of devices data.

Primary actors: Administrator

Preconditions:

1. The Administrator is authenticated in the system.

Main flow:

1. The use case starts when the Administrator access to the devices page.
2. The system finds and displays all the available devices.
3. The Administrator selects multiple devices of the list.
4. The Administrator confirms the action.
5. The system returns a file with the selected devices data.

Postconditions:

1. The devices data is backup.

Exceptions:

1. Administrator cancels the use case. The system does not backup devices data.

Use case: Add devices from list

Brief description: The Administrator selects the file with a list of devices to be added.

Primary actors: Administrator

Preconditions:

1. The Administrator is authenticated in the system.

Main flow:

1. The use case starts when the Administrator access to the devices page.
2. The Administrator selects a devices list file and uploads it.
3. The system validates the uploaded file.
4. The system adds each device on the list to the database.

Postconditions:

1. The devices are added to the system.

Exceptions:

1. Administrator cancels the use case. The system does not add any device.
2. Invalid devices file. The system does not add any device.

6.2.2 Occupant Comfort, Presence Detection and Suggestions

This section presents the use case diagram for occupant comfort, presence detection and suggestions in Figure 6.2 and its use cases specifications. Considering the selected US, the system must ensure the occupants comfort. Therefore it should collect current environment values and act accordingly.

We will only describe the **Adapt luminosity**, **Get current air quality**, and **Change outlet state** use cases. The use case for temperature is similar to the luminosity, and the noise level similar to the air quality.

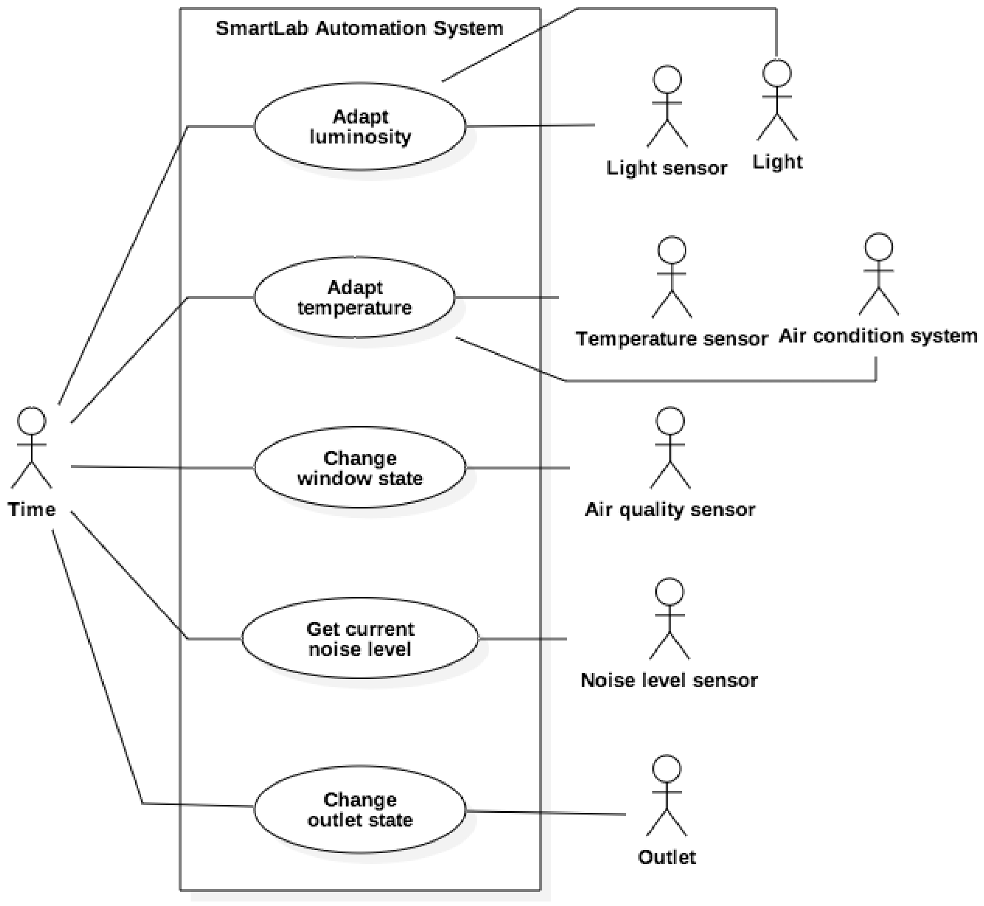


Figure 6.2: Use Case Diagram - Occupant Comfort, Presence Detection and Suggestions. In this diagram, the sensors and actuators are represented as Secondary actors because they are external entities to the system with their own logic.

Use case: Adapt luminosity

Brief description: The system adapts the light status taking into account the current luminosity and human presence.

Primary actors: Time

Secondary actors: Light sensor, Light

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system checks for human presence in the room.
3. If human presence is detected
   a) The system reads current indoor luminosity value from Light sensor.
   b) The system reads current outdoor luminosity value from Light sensor.
      i. If indoor luminosity level is too low
         A. If outdoor luminosity is enough to provide good luminosity
            - The system notifies to open the window.
            - The system turns off the light.
         B. Else
            - The system increases the light brightness level.
      ii. Else
         - The system decreases the light brightness level.
4. Else
   - The system turns off the light.

Postconditions:

1. The system changed the light state.

Use case: Change outlet state

Brief description: The system changes outlet state taking into account the time and human presence.

Primary actors: Time

Secondary actors: Outlet

Preconditions:

1. The outlet is marked as uncritical by an Administrator

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system checks for human presence in the room.

3. If human presence is not detected
   a) Calculate for how long there is no activity.
   b) If it is night and inactivity time is superior to 10 minutes
      – The system turns off the outlet.
   c) Else If inactivity time is superior to 30 minutes
      – The system turns off the outlet.
4. Else
   – The system turns on the outlet.

Postconditions:

1. The system changed the outlet state regardless human presence.

Use case: Change window state

Brief description: The system suggests to open the window to improve the air quality.

Primary actors: Time

Secondary actors: Air quality sensor

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system checks for human presence in the room.
3. If human presence is detected
   a) The system reads current outdoor air quality from Air quality sensor.
   b) The system reads current indoor air quality from Air quality sensor.
   c) If outdoor air quality is greater then indoors
      – The system notifies to open the window.
   d) Else
      – The system notifies to close the window.

Postconditions:

1. The system changed the window state.

6.2.3 Scheduling

This section presents the use case diagram for scheduling in Figure 6.3 and its use cases specifications. Considering the selected US, the User can schedule the time to turn on/off a specific device or a set of devices, edit or remove a previously defined schedule time and schedule the time to feed the fishes. We will only describe the **Schedule time to turn on devices**, **Schedule time to turn on group** and **Edit scheduled time** since these are the most relevant, and the other use cases have a similar flow.

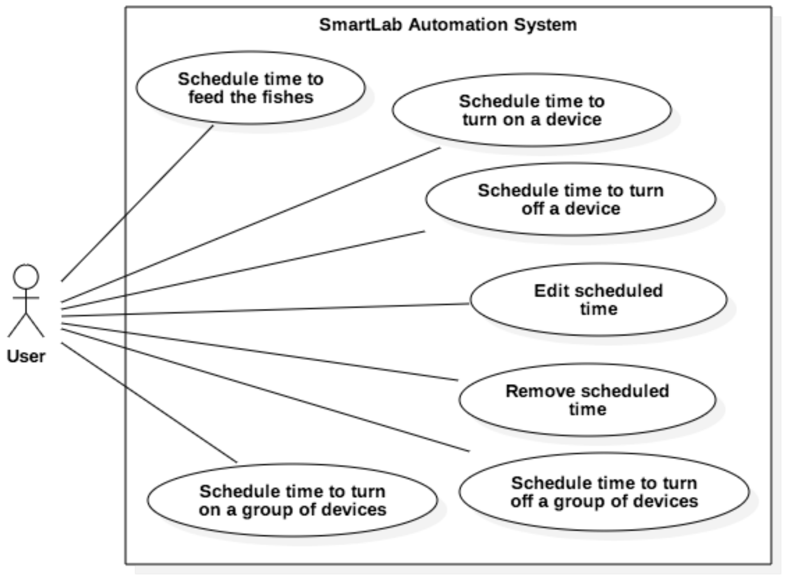

SmartLab Automation System
Schedule time to feed the fishes
Schedule time to turn on a device
Schedule time to turn off a device
Edit scheduled time
Remove scheduled time
Schedule time to turn off a group of devices
Schedule time to turn on a group of devices
User

Figure 6.3: Use Case Diagram - Scheduling

Use case: Schedule time to turn on a device

Brief description: The User creates a rule to turn on a specific device, so that it can be powered on by the system at the scheduled time.

Primary actors: User

Preconditions:

1. The User is authenticated in the system.

Main flow:

1. The system displays a list of all scheduled rules.
2. The User press the button to create a new rule.
3. The system displays all the available devices.
4. The User selects the desirable device, specify the time to turn on the device and confirm it.

5. The system creates a new rule for the selected device.

Postconditions:

1. A new schedule rule for the device is created.

Use case: Schedule time to turn on a group of devices

Brief description: The User creates a rule to turn on a set of devices, so that they can be powered on by the system at the scheduled time.

Primary actors: User

Preconditions:

1. The User is authenticated in the system.

Main flow:

1. The system displays a list of all scheduled rules.
2. The User press the button to create a new rule.
3. The system displays all the available groups.
4. The User selects the desirable group, specify the time to turn on the device belonging to that group and confirm it.
5. The system creates a new rule for the selected group.

Postconditions:

1. A new schedule rule for the group is created.

Use case: Edit scheduled time

Brief description: The User edits a previously created scheduled rule.

Primary actors: User

Preconditions:

1. The User is authenticated in the system.

Main flow:

1. The system displays a list of all scheduled rules.
2. The User selects the rule to be edited.

3. The User specifies the new time to change the device state and confirm it.
4. The system updates the time of the selected rule.

Postconditions:

1. The selected schedule rule is updated.

6.2.4 Notifications

This section presents the use case diagram for notifications in Figure 6.4 and its use cases specifications. Considering the selected US, the system must notify the Administrator about inactive devices, and notify the User about consumptions out of ordinary and aquarium events.

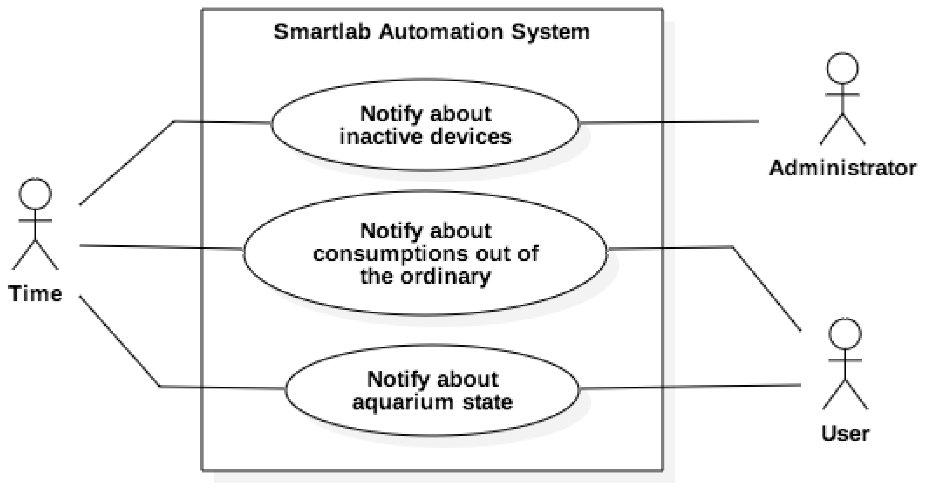

Smartlab Automation System
Notify about inactive devices
Notify about consumptions out of the ordinary
Notify about aquarium state
Time
Administrator
User

Figure 6.4: Use Case Diagram - Notifications

Use case: Notify about inactive devices

Brief description: The system notifies the Administrator when a device does not update its state in the last hour.

Primary actors: Time

Secondary actors: Administrator

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system compares the current time with the time of the last state of a device.
3. If the time is greater than 1 hour
    - The system notifies the Administrator by email about the inactive device.

Use case: Notify about consumptions out of ordinary

Brief description: The system notifies the User when a device has an unusual energy consumption.

Primary actors: Time

Secondary actors: User

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system reads the current energy consumption of a device.
3. The system calculates the average consumption of a device and its standard deviation.
4. If the current energy standard deviation is to high
   - The system notifies the User by email about the device with an unusual energy consumption.

Use case: Notify about aquarium state

Brief description: The system notifies the User about relevant events in the aquarium.

Primary actors: Time

Secondary actors: User

Main flow:

1. The use case is triggered automatically every 20 seconds.
2. The system reads the current state of the aquarium.
3. If the current water level is too high or to low
   - The system notifies the User by email, warning him about a problem with the water level in the aquarium.
4. If the current water temperature is too high or to low
   - The system notifies the User by email, warning him about a problem with the water temperature in the aquarium.

6.2.5 Aquarium

This section presents the use case diagram for the aquarium in Figure 6.5 and its use cases specifications. Considering the selected US, the system must feed the fishes automatically, take care of the water level, temperature and the provided light. We will only describe the **Feed fishes** and **Adapt temperature** use cases since the other use cases have a similar flow. Notice that the aquarium is a subsystem and our system interacts with it through an interface. Thus, we are not representing the aquarium components individually.

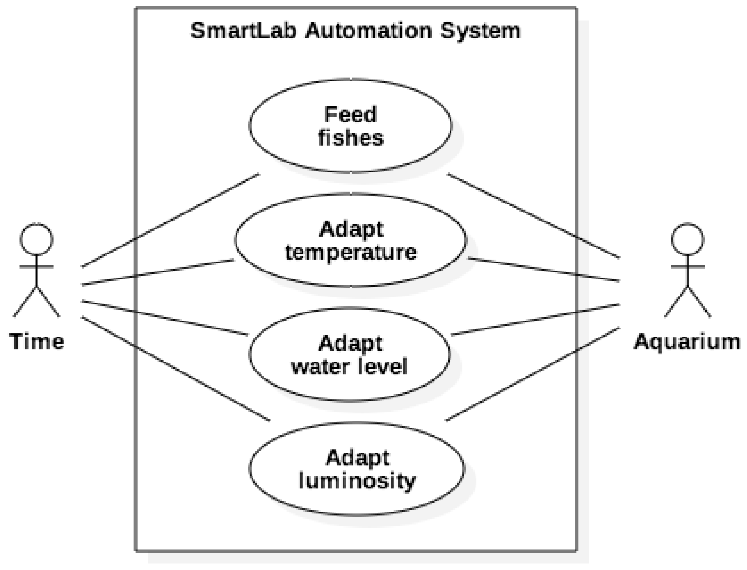


Figure 6.5: Use Case Diagram - Aquarium

Use case: Feed fishes

Brief description: The system automatically feed the fished at the scheduled time.

Primary actors: Time

Secondary actors: Aquarium

Preconditions:

1. The User scheduled the time to feed the fishes.

Main flow:

1. The use case start at the scheduled time to feed the fishes.
2. The system feed the fishes by turning on the feeder.

Postconditions:

1. The fishes were feed.

Use case: Adapt temperature

Brief description:

Primary actors: Time

Secondary actors: Aquarium

Main flow:

1. The use case is triggered every automatically every 20 seconds.
2. The system reads the aquarium current water temperature.
3. If the water temperature is to hot for the fishes
    a) The system turn on the aquarium ventilator.
4. Else
    a) The system turn off the aquarium ventilator.

Postconditions:

1. The aquarium ventilator state changed.

6.3 Architecture

A system architecture is a conceptual model that defines how a system should be organized and structured. It is a critical link between the system design and requirements engineering, since it identifies the organization of the system components, the relationships and constraints between them. The process of creating this model is known as Architectural design, and it needs to satisfy the functional and non-functional requirements of a system. Therefore, the close relationship between the system requirements and the architectural design will influence the architectural style and structure chosen [Som10].

The following sections describe the process conducted in order to achieve a system architecture that conforms to the system requirements defined above.

6.3.1 Non-Functional Requirements Treatment

NFR have a strong influence on design decisions and implementation, allowing us a wide range of possible solutions to choose [Som10]. According to Chung et al., time and cost are usually limited resources, thus we can not work with all the defined NFR. Thus, in order to achieve an appropriate design solution we need to handle the selected NFR, analyse all the trade-offs, resolve possible conflicts between them and assign a degree of importance to each one. The result of this analysis will help us to achieve a well-founded solution based on the system requirements [Chu+00; Ras+03]. To handle the NFR we used a process proposed in [Ras+03] which consists in the activities represented in Figure 6.6. The goal of this process is to identify, select the most relevant NFR, and resolve possible conflicts between them. Furthermore, this process can help to refine the selected NFR and identify conflicts between its operations/use cases [Ras+03]. Our goal focuses only on identify and select NFR and find the conflicts between them.

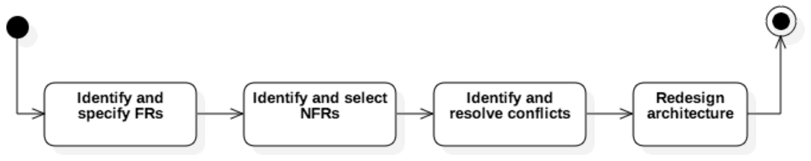


Figure 6.6: Process proposal to handle NFRs (adapted from [Ras+03])

Note that the activity **Identify and specify FR** is already accomplished in Section 6.1.4, and we already identified the NFR in Section 6.1.5. The next activity consists in select the NFR and match them with a real non-functional requirement. Table 6.1 represents the result of this process.

Table 6.1: Identify and Select NFR. The first column represents the non-functional requirement identifier, and the second column the non-functional requirement name.

NFR Identifier	**NFR Name**
NFR1, NFR2, NFR16	Performance
NFR3	Accessibility
NFR4, NFR5, NFR6, NFR7	Security
NFR8, NFR17	Fault tolerance
NFR9, NFR17	Reliability
NFR10	Interoperability
NFR11	Scalability
NFR12	Maintainability
NFR13, NFR14, NFR15	Usability
NFR18	Availability
NFR19, NFR20, NFR21, NFR22, NFR23, NFR24, NFR25	Implementation
NFR26, NFR27, NFR28, NFR29, NFR30, NFR31	Safety

The outcome of the previous activity, provide us a list of the NFR, but as mentioned above treating all the NFR requires an additional time and cost, therefore there is a need to select the most important NFR taking into account our case study and the stakeholders needs [Chu+00]. We choose these NFR because the system should have a reasonable performance for decision-making, it should ensure the safety of the the occupants and it should be easy to maintain and increase so that it can meet its requirements or to handle the growth of its components.

The following list describes the most important NFR and Table 6.2 shows the contributions between them.

- **Performance**: The amount of time required to react to a stimulus or the number of events processed during a certain period of time;
- **Scalability**: Capability of the the system to be enlarged, in size, distribution, or manageability, in order to accommodate the involved components;
- **Maintainability**: Capability of the system to be modified. Modifications may include corrections, improvements or adaptation of the system to changes in environment, or in its requirements;
- **Safety**: Capability to achieve acceptable levels of risk of harm to people, software, equipment or the environment in a specified context of use.

Table 6.2: Identify Contributions between NFR. Each NFR can contribute negatively (-) or positively (+) to others. Empty cells represent null contribution/not significant.

NFRs	Performance	Scalability	Maintainability	Safety
Performance				+
Scalability	+		-	
Maintainability	-			
Safety		-	-	

Analysing in more detail Table 6.2 we conclude that there are some negative and positive contributions. Performance is a requirement that suggests an architecture within a small number of components deployed on a few number of computers instead of distributed across the network. Meaning that it is recommended to use large components rather then small for reducing components communications. On the other hand, Maintainability suggests an architecture to use fine-grain, self contained components, and separate components by concerns or functionalities. Safety is a requirement that suggests an architecture with all safety operations to be located in a single component or in a small number of components. Thus, we can conclude that Performance contribute positively to a Safety regarding a reduced number of components. On the other hand, Maintainability can create potential conflicts for the same reasons it creates with Performance. Scalability suggests an architecture design taking into account the possibility of future changes. Increasing the system components and its resources number, may lead to an increase on its performance. However, a scalable system maybe contribute negatively to Maintainability and Safety, due to the increase of necessary resources or components to maintain, and the need to be careful about existing safety rules [Som10]

Taking into account this analysis we conclude that there is potential conflict between the suggested architectural styles of these NFR. However it can be solved using different architectural styles for different parts of the system [Som10].

The last activity of this process, represented on the Figure 6.6, is **Redesign Architecture** that lead us to the following section, where we will choose the most adequate architecture styles taking into account the list of NFR.

6.3.2 Architecture Styles

As we saw above, a system architecture is a collection of components, their relationships and constraints. Graphically it can be seen as a view that each node represents a system component and the arcs the relationships. Therefore, there are several ways to organize and structure the system components. An architectural style defines a family of such systems in terms of a pattern, specifying how the system components can be combined and its constraints [GS94].

These patterns capture the essence of an architecture that has been used in other systems, describe when they should be used, their strengths and weaknesses. So, an architectural pattern describes a system organization that has been successful in other systems [Som10].

At this point it is possible to analyse a list of architectural styles and choose the ones that best adapt to our system requirements. We will follow the architectural styles proposed by Bass et al. and Garlan and Shaw, in [Bas+12; GS94]. We analyse each architectural style in order to understand the advantages each one could provide to our system architecture. The following list presents the selected architectural styles, and an explanation of how these styles are related to our system.

- **Client-Server**: The client-server pattern is composed by two types of components. The client is a component that requests services of a server component. The server is a component that provides services to the clients. This pattern is useful when there are shared resources required by a large number of clients. The core functionalities are provided by the server component. Thus, this pattern promotes the maintainability and scalability, since the modifications are made in a single location, or a small number of locations.

- **Publish-Subscribe**: In publish-subscribe pattern the components communicate through messages or events announcement. There are components that subscribe a set of specific events, and other components emit these events. A middleware ensures that the triggered events are received by all the subscribers. This pattern promotes maintainability since the components do not know each other, making it easier adding new components or modifying existing ones. On the other hand, this pattern implies a significant cost on performance, since the messages delivery time is unpredictable.

- **Model-View-Controller**: The model-view-controller pattern separates the application functionality into three types of components. A model, that contains the application data. A view, that displays the data represented by the model, and serve as an interaction point to the user. A controller, that behaves as a intermediary between the model and the view and reacts to data state changes, or user actions. This pattern promotes the maintainability and scalability. Since these components are loosely coupled, the modifications in one component will have a minimal impact on the others.

The client-server pattern will help us to structure the system, so its resources and services are shared with all the clients. In this specific case, the server component will provide the required resources and services for user and device management. This pattern is useful since it allows any kind of client device to access the server resources, as long it has an internet connection. Since the resources are server-side the clients does not suffer too much impacted with the server modifications. This, allows us to modify the server services to evolve, or to meet new requirements ensuring a maintainable and scalable system.

The publish-subscribe pattern is used for structure devices communication level, and for handling events that are triggered in system run-time. In this specific case, a middleware will be responsible for event management. The devices will be publishers components, that provide the data about the device state, and subscribers to allow external components to change a device state. This pattern makes it easier to add or remove devices without interfering with the existing ones. Thus, it allows the system the capacity the be maintainable and scalable.

The model-view-controller pattern will help us to structure the system grouping components by concerns. In this specific case, the model components will represent the system stored data, for example the devices properties values. The views will represent the system relevant data about the models, and the controllers are responsible for all the logic to provide the required functionalities. This pattern is useful for system UI, since models, views and controllers are independent we can modify the components to meet new requirements, without affecting too much the existing ones. This ensures the system capability to be modifiable over time.

6.3.3 Architecture Views

Nowadays, systems tend to be complex and our focus is usually on only one specific aspect of the system. It is impossible to represent all the important aspects of an architecture in a single model. Therefore, to simplify the system understanding it is a good practice to adopt multiple views, each one representing different perspectives of the system. These views provide an overall idea of the system decomposed by modules, how these modules interact in run-time, and how the systems components can be distributed. So, each view is important in different stages of the system development process [Som10].

There are several proposals concerning what views should be defined. The most common solution is the 4+1 View Model, proposed by Kruchten, in [Kru95]. This model suggests the following fundamental architectural views:

- **Logical view**: Functionalities provided by the system to end-users. It serves as a communication model between all the involved stakeholders;
- **Process view**: Dynamic aspects of the system, its processes, how they communicate and their behaviour in run-time;
- **Development view**: System programmers perspective, focusing on software management and its subsystems;
- **Physical view**: System engineer perspective, focusing on components topology, how they are mapped from software to hardware and how they communicate.

Figure 6.7 represents a logical view of the system architecture. Note that this model was designed following the model-view-controller pattern. In blue we have the system boundaries, that represent the system interfaces presented to the users for data visualization, interaction and input data, or back-end interfaces for external services. The red components are the controllers, where it is located the main logical functionalities of the system. The model components in orange represent the system state as data models and its relationships. Remember that this diagram only presents the functionalities taking into account the selected requirements, in the previous sections.

Figure 6.8 represents a process view of the system architecture for a very specific scenario, in this case for the process of changing a device state. Initially the automation component requests the list of existing devices to the platform component, creating locally the entities that represent each device. The automation component requests the last state of the devices, and sets the devices properties values accordingly. At the end of this process, the automation component is responsible for evaluating the devices states that need to be adjusted. If a device state needs to be adjusted, the automation model communicates with the platform component with the device to be changed and the new state value, which in turn communicates with the physical device.

Figure 6.9 represents a development view of the system architecture. In this model it is possible to verify that there are 4 main components. The Platform component represents the chosen platform subsystem. In this case we are not representing its internal components because, regarding our system this component it is like a black box, that provides interfaces for user and device management. The Database component represents a

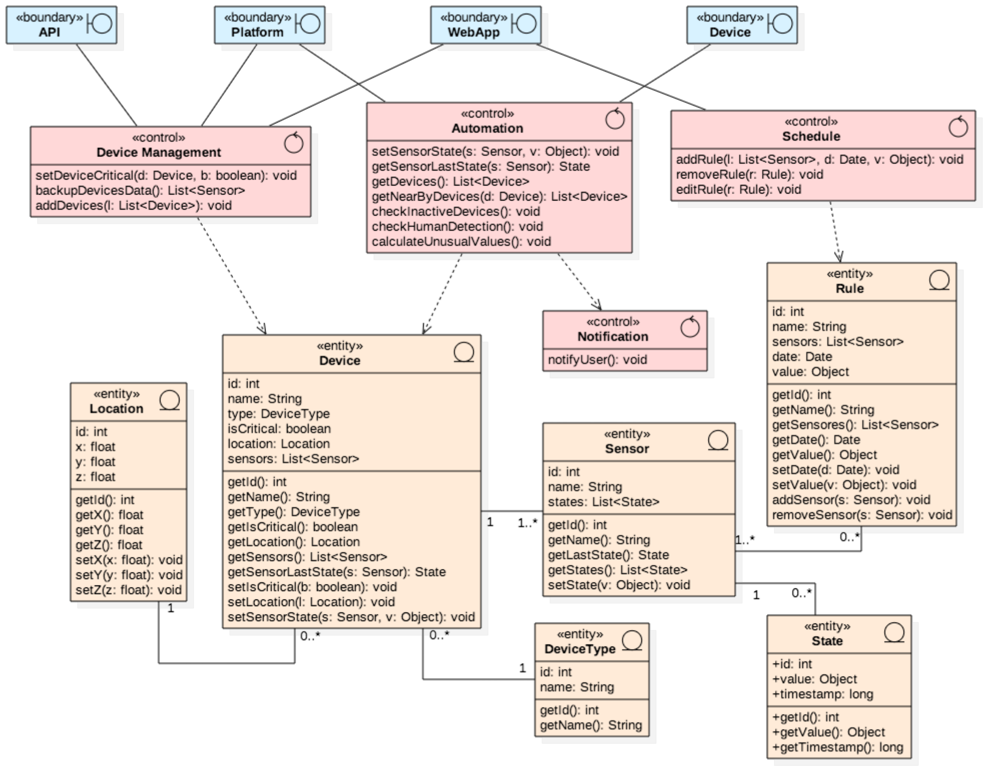


Figure 6.7: Architecture Views - Class Diagram

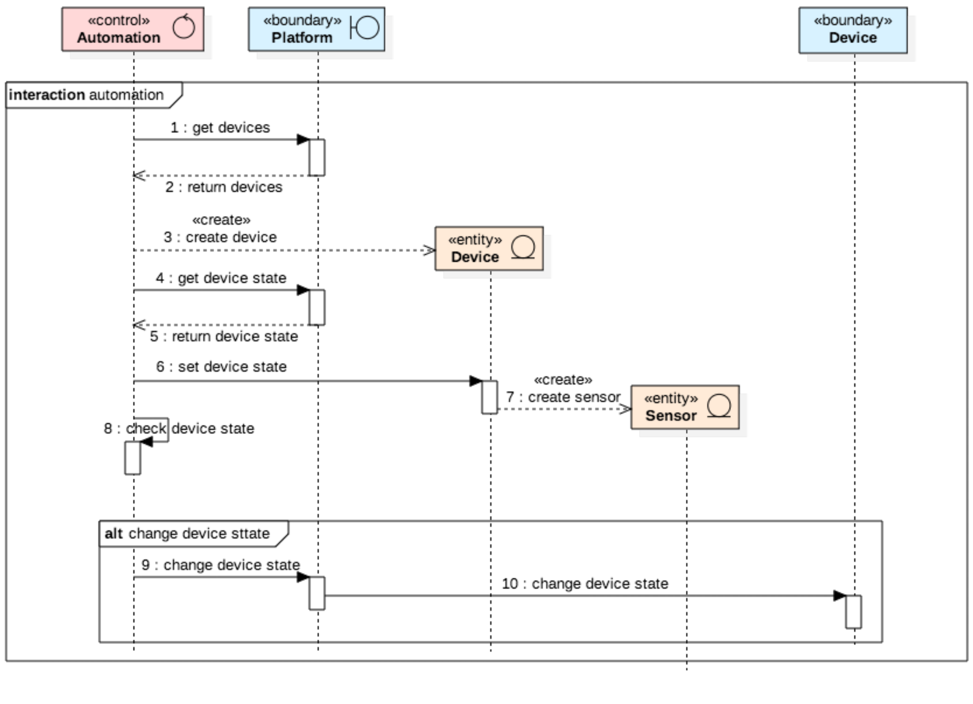


Figure 6.8: Architecture Views - Sequence Diagram

subsystem that stores data that will complement the data on the platform. The component responsible for the automation is represented as the Automation Module subsystem. This component depends on the interfaces provided by the Platform, for device control, and Database to require more detailed data about device and scheduled rules. It also uses an external email service component for notifications. Lastly, the Front-end component is a subsystem responsible for displaying information about the devices in addition to the one provided by the Platform. This component provides the functionalities to define new rules and devices additional values.

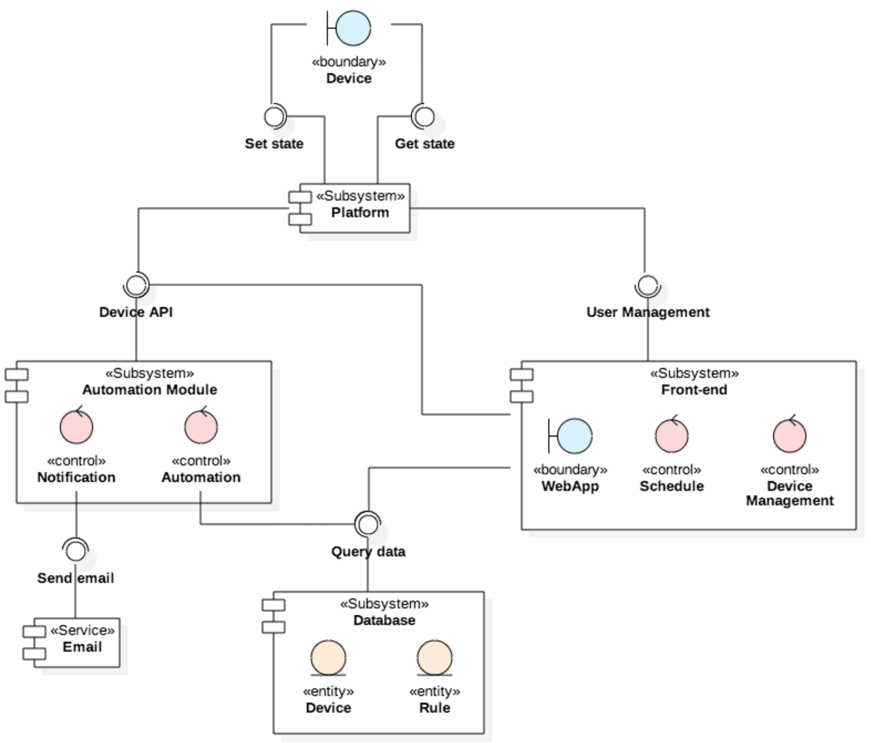


Figure 6.9: Architecture Views - Components Diagram

Figure 6.10 represents a physical view of the system architecture that shows how the software components are deployed to physical ones. The whole system is composed by 4 types of components. The simplest component is the Device, that presents all the physical devices to be used by the system, from sensors to actuators. Device Agents are used as intermediaries for communication between the Platform and the Devices. These can be Raspberries or Arduinos, where are installed the software responsible to collect devices state data and send it to the Platform. In our case study the Device Agents communicate with the Devices through HTTP or Bluetooth. The Platform component is an application server, where the platform chosen will be running. It provides all the functionalities for devices, user API management and data storage. This server communicates with the Device Agents using publish-subscribe mechanism through MQTT protocol. The Device Agents are responsible to publish devices state, and subscribe to topics in order to handle events triggered to change devices state. Lastly, the Computer component responsible for providing the functionalities of the Front-end module. It enables rule management and edit detailed data about the devices. This computer also contains the component responsible for the automation control. Locally, the Automation Control process communicates with the data base system through the MySQL protocol. Furthermore, these components also communicate through a websocket over TCP/IP. This communication path allows the settings modified by the users on the Front-end system, to take effect on the Automation Control process without having to restart it, or wait for it to update its values the next time it communicates with the database. In order to request data about the devices and control them, the Computer component also communicates with the Platform over HTTP.

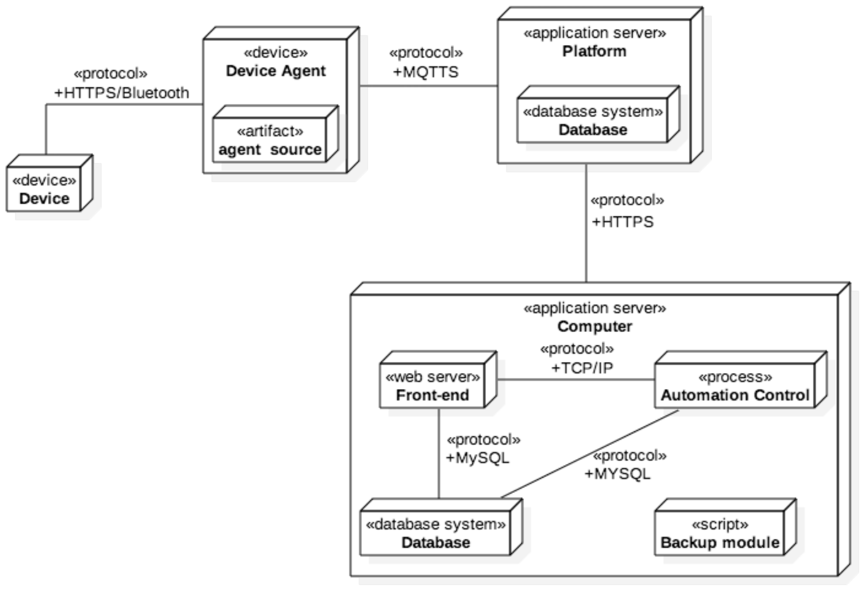


Figure 6.10: Architecture Views - Deployment Diagram

6.4 Implementation

This section goal is to show the work developed to achieve a functional product, taking into account the requirements analysis and the architecture design presented in previous sections. The following sections will approach the technologies used for the components implementation, explain how these are related to the components presented in the architectural design, and the functionalities provided.

In Chapter 5 we made a detailed analysis between WSO2 and IBM Watson platforms. For our final proposal we choose to use the WSO2 platform. It provides the expected functionalities of an IoT platform and for our case study. Being open-source, it gives us the freedom and ownership over the platform and the existing data, unlike the IBM Watson where the data would not be owned by us. Another advantage, is the costs associated with these two solutions. Since WSO2 is open-source its deployment can be done using our infrastructure, not having to support utilization and network bandwidth costs. It also makes us independent of external services in case of some unexpected failure. Another reason do choose WSO2 instead of IBM Watson is due to the fact that our case study is subject to network failures and latency problems. These have a significant impact in automation and control systems. Therefore, having WSO2 platform located on our local network reduces possible network and communication failures.

The chosen platform is able to support some of the functionalities provided by the automation module. However, we decided to separate the control and automation logic from the platform in an independent module. The reason is due to the fact is to make this module independent of the platform chosen. Thus, keeping logic and control rules independent of the platform allows this module to be independent of the platform and enables to capability to adopt another platform in the future. In this case, it will only be necessary to handle the communication between the module and the platform, keeping all the logic and control rules intact. Usually each platform provides different tools, mechanisms and their own languages for control and rule based events. Therefore, it requires an additional effort to learn and develop solutions based on the chosen platform. Thus, making this module independent is another advantage. Another reason is because in future we may decide to follow Fog Computing principle. In this way, since the automation module is independent from the platform, we could change it easily by handling the communication directly with device agents instead of communication with the platform.

The following components communicate with the WSO2 Platform through the HTTPS protocol. The platform administrator has previously created an Application through the APIs management page. It was granted the required permissions to this Application so that it can provide the expected functionalities. As a result, these components have to include on each request the application key and the token provided by the platform administrator. In this particular case, the provided token has an expiration time of 1 year. Therefore, our components do not have to deal with the logic to refresh the token. However, in a real case scenario that logic must be taking into account.

Most of the following presented components were developed using NodeJS. We chose to use NodeJS because it uses an event-driven, non-blocking model. The NodeJS uses internal mechanisms that help to deal with multiple requests, making it easier to develop these modules, reducing development effort by the programmers. Furthermore, it is an open-source technology that can be executed on any platform.

6.4.1 Automation

The developed system has as one of the objectives the automation of multiple devices, in order to reduce the energy consumption while ensuring the occupants comfort. Therefore, before the implementation of the automation module it was necessary to evaluate the following points:

- Understand the standards the system should adopt to improve occupants comfort and ensure good working conditions;
- Set of devices that can be controlled in order to reduce energy consumption without interfering with occupants working conditions.

The following sections describe in more detail the analysis conducted to understand the standards the system should adopt and the required preconditions to ensure the proper system operation.

6.4.1.1 Comfort Standards

In order to understand what are the recommended standards for workspace comfort, we start by analysing previous studies that are realised and test in case scenarios similar to our case study. The result of this analysis helped us to realise what are the standards that should be taken into account to ensure the occupants comfort.

In the case of thermal comfort the environmental conditions for indoor air temperature of a building should be 18ºC in the heating season and 25ºC for the cooling season. Note that these values are the minimum required to ensure minimal thermal conditions in terms of the building structure.

Regarding the indoor air quality, the reference values for air refresh rate should be between $0.4R_p$ and $0.6R_p$, where R_p is the ventilation rate per person per hour [Des].

For visual comfort it is necessary to take into account the natural and artificial sources of light. The outdoor luminosity level is approximately 10000 lux on a clear day. In a workspace near the window this value can drop to approximately 1000 lux, and to values between 25 and 50 lux in a workspace in the middle of a room away from the window. Therefore, it is necessary to adopt strategies to handle the luminosity levels. Depending on the activity the recommended light level is in the range of 500 to 1000 lux, but for activities like normal office work, study, pc work it is recommended levels of 500 lux [Hua+12].

In a study conducted by Huang et al. he tried to understand what are the luminosity levels that provide better visual comfort for office occupants. In this study 120 people from both genders, with ages between 19 and 28 years old took part. The participants were exposed to different levels of temperature, noise, and luminosity during a fixed period of time. At the end of each period of time, each participant had to fill a form that evaluates de degree of satisfaction during that time. After a short break the participant repeated another test with different values. As a result of this study Huang et al. noted that, the range of values that provide higher satisfaction to the participants were between 20.9ºC and 30.4ºC. For luminosity satisfaction the values were above 300 lux, and for noise levels bellow 49.6dB. Huang et al. also concluded that the temperature and noise levels are the ones that have greater impact on the participants satisfaction levels in comparison with luminosity levels. It is important to take these values into account since these can have a negative impact on occupants well-being, mood, and also in their productivity [hor+16; Hua+12; Maxa].

Remember that in our case study we have an aquarium and it needs special attention, because the room temperature and luminosity levels can affect the aquarium water temperature. The aquarium in our case study contains fishes of cold water, and the recommended water temperature should be between 15ºC and 20ºC [Bol09]. Additionally, one of the thermodynamics principles says that the heat generated by objects can be transferred to other objects if they have distinct temperatures [LL17]. Following the heat transfer principle, if the room temperature is too high it implies an increase on the aquarium water temperature, and eventually it can lead to water evaporation. Otherwise, a lower room temperature implies a decrease on the aquarium water temperature. The luminosity level coming from natural or artificial light sources can also cause this phenomenon. Therefore, our system must be able to apply rules to reduce the energy consumption, ensure the occupants comfort with the standard values described above, and also make sure that these values have a minimal impact on the aquarium environment.

6.4.1.2 Devices Categorization

In an initial phase it was required to organize the room devices to allows us to map the devices located in a specific workstation, that lead to a group of devices allocated within the workstation. Thus, the devices were placed in the workstations and we took note of their relative position to the room. Furthermore, the devices position were inserted in the platform and used to calculate the nearest devices from a specific source. This mapping process also helped us to produce a list of devices that can be controlled and in what socket they were plugged in. Table 6.3 presents the result of the mapping process and we can see a description of what device is connected to a specific socket and if it is considered a critical device.

The sockets marked as critical represent the ones that can not be turned off in any case by our system. Note that there are 3 sockets that represent the occupants portable computers. Furthermore, in Section 6.4.1.4 we will explain the reason of this choice. After this mapping, we ask the room occupants to plug their portable computers to the designated sockets, in the ambit of this dissertation and to simplify the process of data measurement, and also to make it easier the analysis of the energy consumption. Additionally, note that the definition of critical devices is customizable through the Front-end module. So it is possible to change in at any moment what devices are critical or not.

6.4.1.3 Control Rules

Before we describe the implementation process, we need to specify the control rules this module will support in this phase. There are several types of rules, the ones triggered in a specific time of a day, or changes of the room environment and also through presence

Table 6.3: Devices connected to sockets map

Outlet	Socket	Description	Critical
1	1,2,3	Aquarium	Yes
3	1	Peripherals	No
3	2	Portable Computer	Yes
3	3	Monitor	No
3	5	Light	No
3	6	Heat/Cooler Fan	No
4	1	Raspberry Pi 3	Yes
4	3	Portable Computer	Yes
4	4	Raspberry Pi 3	Yes
5	1	Monitor	No
5	2	Portable Computer	Yes
5	3	Monitor	No
5	4	Coffee Machine	No
5	5	Light	No

detection events. To specify the existent rules we adopt a syntax proposed by Open Hab [Ope]. Each rule consists of 3 blocks that define its name, what event triggers the rule and its action.

Schedule Rules

Definition of rules that are triggered through time events. These rules were previously scheduled through the UI provided by the Front-end module.

```
rule "turn on uncritical sockets"
when "time is 7.30 am"
then
   1. Iterate sockets collection
      1.1. If socket S is not critical
         1.1.1 Turn on socket S
end

rule "feed the fishes"
when "time is 10.00 am"
then
   1. Feed fishes
end

rule "turn off coffee machine"
```

```
when "time is 7.30 pm"
then
   1. Search for socket with id SOCKET_5_4
      1.1 Turn off socket
end

rule "turn off lights and uncritical sockets"
when "time is 8.00 pm"
then
   1. If the room is empty
      1.1 Iterate lights collection
         1.1.1 Turn off light
      1.2 Iterate sockets collection
         1.2.1 If socket S is not critical
            1.2.1.1 Turn off socket
end
```

Presence Detection Rules

Definition of rules that are triggered through presence detection events, whether it is when an occupant arrives or leaves the workplace, or when the room is empty.

```
rule "turn off lights and air condition"
when "room is empty"
then
   1.1 Iterate lights collection
         1.1.1 Turn off light
   1.2 Turn of air condition system

end

rule "adapt workspace luminosity and every 20 seconds"
when "occupant arrives at workspace"
then
   1. Get workspace current light intensity
   2. Search for the light on its workspace
   3. Calculate the new light state
   4. Change the light state with the calculated value
end

rule "turn off workspace light"
when "occupant leaves the workspace"
then
   1. If occupant leaves the workspace
      1.1 Search for the light on its workspace
```

```
        1.1.1 Turn off the light
end
```

Environment Rules

Definition of rules that are triggered by changes in room environment variables, such as luminosity and temperature levels.

```
rule "adapt lights intensity"
when "room light is not within the standards"
then
   1. If the room is not empty
      1.1 If the current light is lower than standards
         1.1.1 Increase lights brightness
      1.2 If the current light is higher than standards
         1.2.1 Decrease lights brightness
end

rule "adapt room temperature"
when "room temperature is not within the standards"
then
   1. If the room is not empty
      1.1 If the current temperature is lower than standards
         1.1.1 Increase air condition temperature
      1.2 If the current light is higher than standards
         1.2.1 Decrease air condition temperature
end
```

Notification Rules

Definition of rules that are triggered by relevant events about overall system functionalities.

```
rule "notify about room lights"
when "all lights turned on or off"
then
   1. Notify by email if all lights are turned on or off
end

rule "notify about outlets"
when "all outlets turned on or off"
then
   1. Notify by email if all outlets are turned on or off
end
```

```
rule "notify about fishes feed"
when "system feed the fishes"
then
  1. Notify by email when the system feed the fishes
end

rule "inactive devices"
when "device does not update its state within 1 hour"
then
  1. Notify by email about devices that do not update their states
end

rule "system report"
when "a system error occurred"
then
  1. Notify by email the cause of the system failure
end
```

6.4.1.4 Implementation and Components

In the deployment diagram presented in Section 6.3.3 Figure 6.10 we saw that the communication between the devices and the platform is achieved through an intermediary, a device agent. Before we start the automation module implementation it was require to ensure that the agent devices were capable to communicate will all the physical devices, if all the control commands were being sent correctly and if the physical devices data were correctly published to the platform.

The agents installation and setup was a work developed previously and it is not the focus of this dissertation, but it was required to ensure that they were in accordance with our system requirements. Therefore, the agents also must ensure that the published data have a relevant change from the previous device state, in order to reduce the volume of data generated. Additionally, each agent is responsible to publish the device state hourly, to evaluate the possibility of inactive devices. Once these conditions were guaranteed, we started the implementation process of our solution.

In an initial phase of the implementation we started by using a flow-based programming tool for the IoT, that allow us to describe the application behaviour through a network of nodes. This allowed us to create an automation module through a visual language, allowing future changes to be cheaper and faster.

The tool used for the initial phase was Node-red, based on NodeJS and released by IBM in 2014 [Maxb].

Figure 6.11 illustrates an example of a flow created with this tool. In this flow it is possible to observe an input node that provides a HTTP route, that parses the received data and changes the state of the specified light. Despite the advantages provided by this tool, we choose to discard it due to the complexity of our system. However, it was a tool that provided us the implementation of a general solution and helped us in the implementation of the final solution.

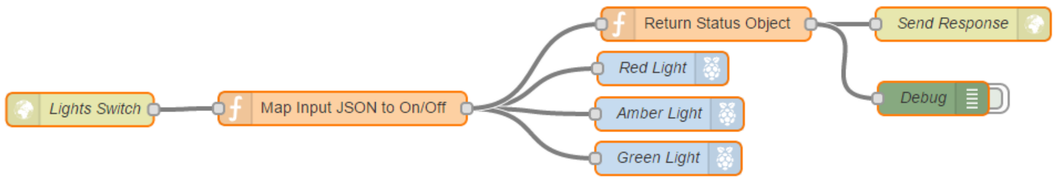


Figure 6.11: Node-Red Flow Example (adapted from [Maxb])

As mentioned before, the automation module should manage multiple environment variables, namely temperature and light, in order to provide comfort to occupants. This management does not happen by turning on or off the devices, it needs to adapt the environment taking into account the occupants inside.

To control the air condition and adapt its temperature we used an adaptation of the algorithm proposed by Purdon et al. [Pur+13] (represented in Listing 6.1). This algorithm has two methods to handle air condition control. The first method uses the room occupants votes, where they can vote to decrease (-1), preserve (0), or increase (+1) the temperature. In this case the temperature is adapted with variations of ±1°C, because lower values would take more time to reflect in the room temperature, and higher values would reflect in drastic changes in the room temperature. The second method adapts the temperature in order to reduce the energy consumption. Purdon et al. they conclude that adapting the temperature with variations of ±0.5°C against the outside temperature may imply a reduction of energy consumption, calling this strategy **Drift**. The system should only operate in Drift mode if the current temperature is not the most suitable and it should stop as soon as it founds the proper temperature level.

Listing 6.1: Air Condition Control algorithm (adapted from [Pur+13])

```
function controlAirCondition()
  if(isRoomEmpty) then
    return turnAirConditionOff()
  end if

  if(userVote < 0){
    return changeTemperature(setPoint - 1)
  }else if(userVote > 0){
    return changeTemperature(setPoint + 1)
  }

  if(drift) then
```

```
    if(setPoint > outdoorTemp) then
      return changeTemperature(setPoint - 0.5)
    else
      return changeTemperature(setPoint + 0.5)
    end if
  end if
end function
```

To control the lights and adapt the room luminosity we used an adaptation of the algorithm proposed by Mohamaddoust et al. [Moh+11]. This algorithm calculates the luminosity level according to predefined activity values for each workstation. As mentioned before, the recommended luminosity levels for our case study workstations its about 500 lux. This value can have a tolerance value of ±10 lux. The algorithm idea is to calculate the difference between the current luminosity and the recommended luminosity value. The difference between these values represent the degree of dimming. However, we noted that the values received by the sensors installed in our case study are not accurate. Although we take into account the idea proposed by Mohamaddoust et al. algorithm to adapt luminosity, it was necessary to change it so that we can achieve stable luminosity levels applicable to our case study. Therefore, instead of calculate the exact value to affect each light bulb, we decided to dim the luminosity over time. This allowed us to achieved the desired luminosity level and change the light bulb intensity with smooth changes without causing drastic luminosity changes.

One of the particularities of this system is the adaptation of the luminosity in a specific workstation. Therefore, the light sensors were positioned so that they can measure the workstation luminosity. Since there is no direct map between light sensors and the light bulbs it was necessary to adopt a mechanism to calculate the nearest light sources from a light sensor. To calculate what are the light bulbs nearest to a specific light sensor we used the euclidean distance. This measures the distance in straight line between two points. Equation 6.1 is the general expression that calculates the distance between the points **p** e **q** defined in an *Euclidean n-space* [Jai+99].

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \tag{6.1}$$

In our case study, the devices coordinates are represented in a three-dimensional space, therefore we can calculate the distance between two points through the simplified expression presented in equation 6.2.

$$d(p,q) = \sqrt{\sum_{i=1}^{3} (p_i - q_i)^2} \tag{6.2}$$

Using this method to calculate the distance between two points, allowed us to know what light bulbs need to be adapted when the luminosity level of a sensor is not within the recommended values.

At the beginning of the automation module development process, there was no mechanism to handle presence detection. This dissertation does not focus in what equipments should be chosen and their installation, it focus on how to handle the existing equipment in order to achieve the desired solution. Therefore, we must adapt a mechanism to handle presence detection taking advantage of the existing ones.

The first attempt results on using network mechanism to detect connected devices. This was achieved using the Address Resolution Protocol (ARP) to get the cached list that maps IP addresses to MAC addresses. Having the list of IP addresses, we filtered it by MAC addresses in order to get the ones that represent occupants mobile devices or portable computers. Lastly, to ensure each device was connected to the room local network we emit a ping. This solution had some problems due to the existence of multiple networks reached by the devices. Therefore, we could not guarantee that occupants devices were connected to our room local network.

The second attempt results on analysing the energy consumption designated for occupants portable computers. Note that in Table 6.3 the sockets 3-2, 4-3 and 5-2 are designated to occupants portable computers. It was also mentioned that we ask the occupants to connect their computers to the designated socket. This option is due to the fact that we realise that it was possible to deduce presence detection by analysing the energy consumption of a portable computer. Figure 6.12 represents an energy consumption chart of a portable computer during a workday. Analysing this chart we can see that before 9 am there was no energy consumption, and it starts registering energy consumption when someone plug in its portable computer. It is also possible to see that during this workday, the occupant leaves the room two times, represented by the time intervals that there is no energy consumption. Even if the occupant leaves the room and closes its computer and leaves it plug in, the energy consumption drops drastically but not to zero. So we consider that the occupant is not present if the power consumption in the sockets is bellow 5 *watts*.

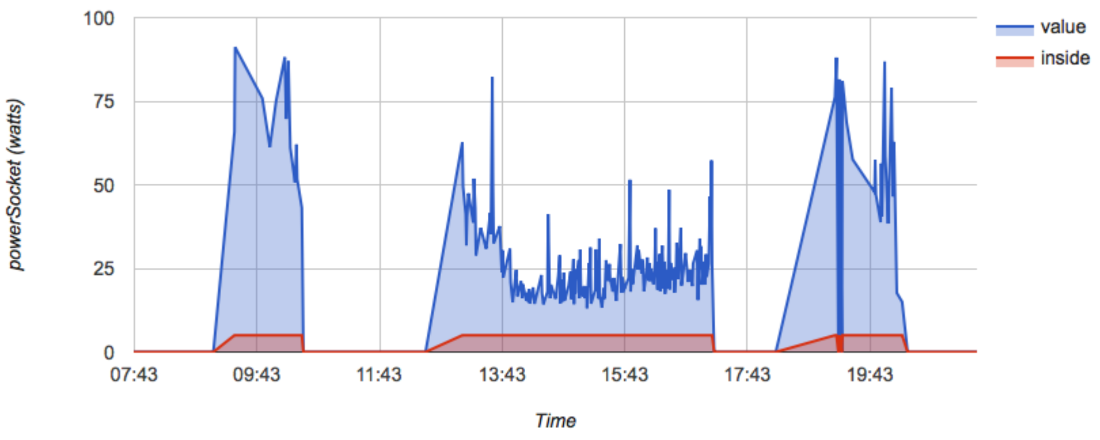


Figure 6.12: Power socket workday energy consumption example

We realised that this method is not accurate due to its limitations, for example the occupant uses his portable computer in battery mode. Therefore, it was developed an external mechanism to read occupants identification card when they enter or leave the room. These data records were compared to the method described above to ensure that they match. In Chapter 7 we will explain how the identification card reader helped us to minimise the limitations coming from the sockets energy reading method.

The automation module final solution was developed using NodeJS, and we start by structure the required functionalities by different services. The adopted structure helps in code organization and it simplifies future modifications, since it only requires to add or modify service functionalities. The following list presents the components in our automation module, describing each one and the functionalities provided.

- **Core**: Main component of the automation module. It deals with the interaction between the other components. It contains the fundamental control logic, the methods to analyse the data coming from the other components and decision make. This module is also capable to adapt to new devices added to the platform, without requiring to restart the module in case of modifications on the platform.

- **Event Handler**: Handles emitted events by the other components and it calls the specified functions registered as listeners of the emitted events.

- **Database Service**: Responsible for querying local database, in order to get scheduled rules and the list of critical devices. These queries are executed when the system starts and every time the Socket Service component receives a message by the external Front-end module.

- **Task Service**: Responsible for rule schedule management. It creates, destroys tasks with the rule specified properties, and it also emits the event to trigger the rule in the specified time.

- **Detection Service**: Handles room presence detection. It evaluates the data to check for occupant presence and what workstations have occupants working. It also evaluates the data to check if the room is empty or not.

- **Devices Service**: Responsible for devices management in execution time. It stores each device state and properties in memory. Provides the functionalities to get devices by a specific device type, and it calculates the nearest devices giving an origin point.

- **Platform Service**: Deals with all the communications between the automation module and the platform, providing the functionalities to get the platform devices list an to get and set devices state. Additionally, to reduce the number of requests made to the platform, it also contains the logic to determine if the change of state is different from the current one.

- **Socket Service**: Provides a websocket service, creating a server in a specific port that waits for new clients to connect. It is responsible for the communication between the automation module and the Backup module, sending a message every time it is necessary to backup the devices data. It also handles the communication with the Front-end module, receiving a message every time a modification to scheduled rules or device properties are made through the Front-end module.

- **Notification Service**: Responsible to send notifications to administrators or occupants. At this moment it provides the functionalities to send alert emails to administrators, when a specific event occurs in the room environment or in case of system failure.

- **Maintenance Service**: Evaluates the devices state consulting the new states coming from the platform and comparing them with the ones in memory. If the evaluating method notes an unusual state or state not updated within 1 hour it notifies the administrators through the Notification Service component.

6.4.2 Front-end

The Front-end module is composed by two sub models, the Web Application and Data Visualization. The Web Application module is used to complement the UI, enhancing the functionalities provided by the Platform. The Data Visualization module provides visualization tools that are not accomplished by the Platform. Since the Platform falls back on the functionalities provided for data visualization and device customization, it arises the need to complement it with these two modules. The following subsections describe what technologies were used in the implementation of these modules, and what functionalities they provide.

6.4.2.1 Web Application

The Web Application module is composed by a Web Server and an UI. To allow modifications and ensure this module maintainability, we choose to divide these components into two. The Web Server interacts with the Platform and the local database, and the UI handles with the user interaction. Therefore, we choose to use technologies that support the separation of the client and server components. Making these two components as independent as possible, we can guarantee that future modifications have the least impact between them.

The Web Server is developed using NodeJS, with the ExpressJS Framework. This framework allows creating web applications following the model-view-controller pattern, which goes according to the architectural design presented before, namely the components diagram. Therefore the Web Server contains the controllers and models required to provide the expected functionalities. However, the view component, as mentioned before is reached using a separated component. The UI component is developed using React, a component-based technology developed by Facebook, that allows the creation of interactive web applications [Fac17b]. Since React does not follow the model-view-controller pattern by default, we choose to structure the application following the Flux Architecture, also provided by Facebook. This pattern allow us to structure the application on the client side following a model-view-controller style, dividing the components by different concepts. Figure 6.13 illustrates the data flow provided by the Flux Architecture. The Action represents the interaction that triggers a specific event. The Dispatcher acts like an message hub, redirecting the actions to the specific stores, that previously subscribe to the actions. The Store represent the application state, instead of a single model it contains all the applications models. It is also responsible to emit events about state change, that will be caught by the subscribed. The view is the component used to display the application state. Note that the flux pattern also follows the publish-subscribe mechanism, where the stores triggered the event message when the state as changed, and the view needs to update [Fac17a].

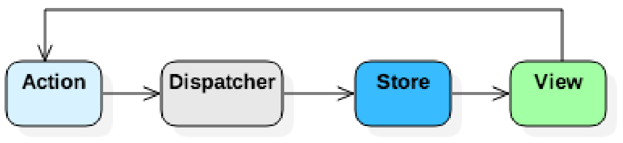


Figure 6.13: WebApp User Interface Flux Architecture [Fac17a])

Lastly, the UI is developed taking into account responsive guidelines, allowing the Web Application to be accessed through any device. Appendix C presents some images of the developed UI. As an extra we added an interface for support, that presents system failure reports, and allows users to suggest modifications or corrections.

6.4.2.2 Data Visualization

The Data Visualization module is developed using the PHP language and provides visualization tools such as charts, average and total values. It was developed thinking on future analyses, to provide more information that the Platform could not. This module does not interact with Platform directly since it uses offline data provided from the Backup module, described in Section 6.4.3.

Figure 6.14 presents the UI developed for this module. As we can see it provides a panel containing multiple options to select periods of time the device and the respective sensor. The chart, created using Google Chart API, is interactive and lets the user zoom or click to check a specific value. An advantage of this UI is that it lets the user navigate through the days, or select the period of Automation on or off. Lastly, this module also provides the functionality to download the chart as an image.

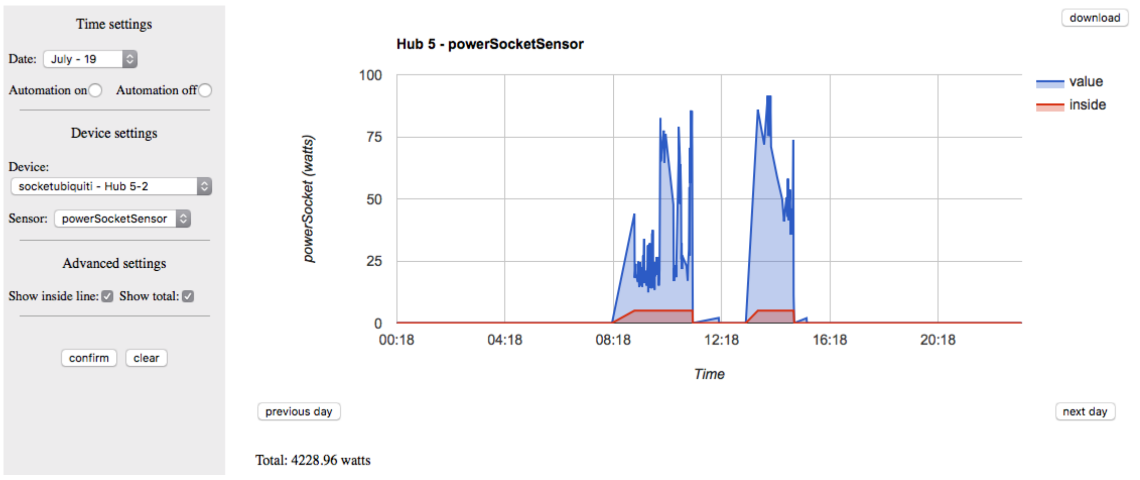


Figure 6.14: Data Visualization User Interface Example

6.4.3 Backup

The Backup component is responsible to backup the devices state stored on the platform between a given time range. It is useful for backing up the specific devices data in a specific time. Since the Platform does not provide the functionalities to analyse the data and create charts within 1 month, backup data also enables more useful visualization tools used by the data visualization component.

The Backup component is a console application developed in NodeJS and it can be executed with an argument that specify the time range, or with no arguments and it runs as a service. Running this component with an argument that specifies the time range, will backup all the devices data between that range for a time period not greater than 24 hours. After the backup is completed the program terminates. Otherwise, running this program without arguments, it will run as a service. When running as a service it creates a communication path with the Automation module, through a websocket. This ensures that the Automation module can trigger the backup script at a specific time everyday without the necessity to worry with this process. Figure 6.15 illustrates the interaction diagram of the Backup module process while executing as a service.

As we can see the Automation module triggers the backup script, that starts by getting the devices list from the Platform, and storing in a local file afterwards. Then the Backup module iterates through the devices and requests each device state to the Platform, and storing all the devices states at the end of the iteration. Note that requiring all the device state even if for a period of 24 hours, leads us with a large amount of requests and data to be handled. Therefore the Backup module ensures that there is a time interval between each request and it makes sure that for large periods it divides in smaller time intervals resulting in a reduced amount of that. For example, backing up the devices state in a period of 24 hours, the Backup module divides in time frames of 1 hour. At the end of the backup process, it notifies the Automation module so that it can notify the users about the completed backup. The output of this process consists of two CSV files, one

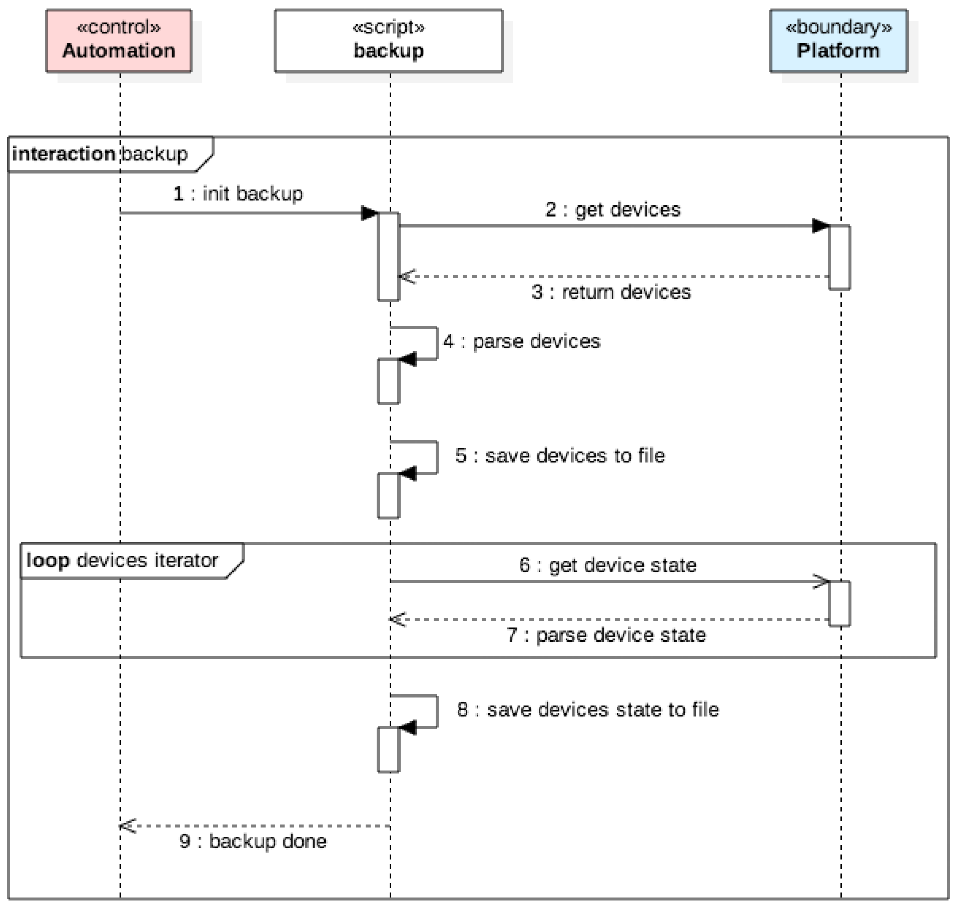


Figure 6.15: Backup Module Overview Diagram

that contains the devices with their properties, exemplified in Table 6.4, and the other the device states between the date range, exemplified in Table 6.5.

Table 6.4: Backup Module - Devices List example file. Legend: Id - device unique identifier. Type - identifier of the device type. Name - description of the device name. (X,Y,Z) - coordinates in meters relative to the room.

Id	Type	Name	X	Y	Z
1	estimotebeacon	Workstation 8	5.500	2.060	0.750
2	smartlights	Workstation 8	5.83	2.53	1.20

6.4.4 Devices Tool

The Devices Tool component is responsible to automate the process of adding devices to the Platform. To add a new device to the Platform it is required to do it through a form in the Platform. Once the device is added, the Platform provides a set of files for download, that contains the device credentials used to authenticate the device with the

Table 6.5: Backup Module - Devices State example file. Legend: Id - device unique identifier. Type - identifier of the device type. Name - description of the device name. Sensor - description of the sensor name. Value - current value of the sensor. Timestamp - date and time of the day of then the state changed occur in milliseconds.

Id	Type	Name	Sensor	Value	Timestamp
1	estimotebeacon	Workstation 8	lightSensor	51.84	1501543563114
2	smartlights	Workstation 8	powerSensor	1	1501545003697

Platform. Having a large number of devices cause this task to be time consuming. Therefore, this module automates this process by adding the devices given a pre-configured file in CSV, and downloads the specific device credential files.

The Devices Tool is a console application developed in NodeJS and it is executed with only one argument that specifies the path for the file to be used. The file must be filled following a template. Figure 6.16 illustrates the Device Tool process interaction with the Platform module.

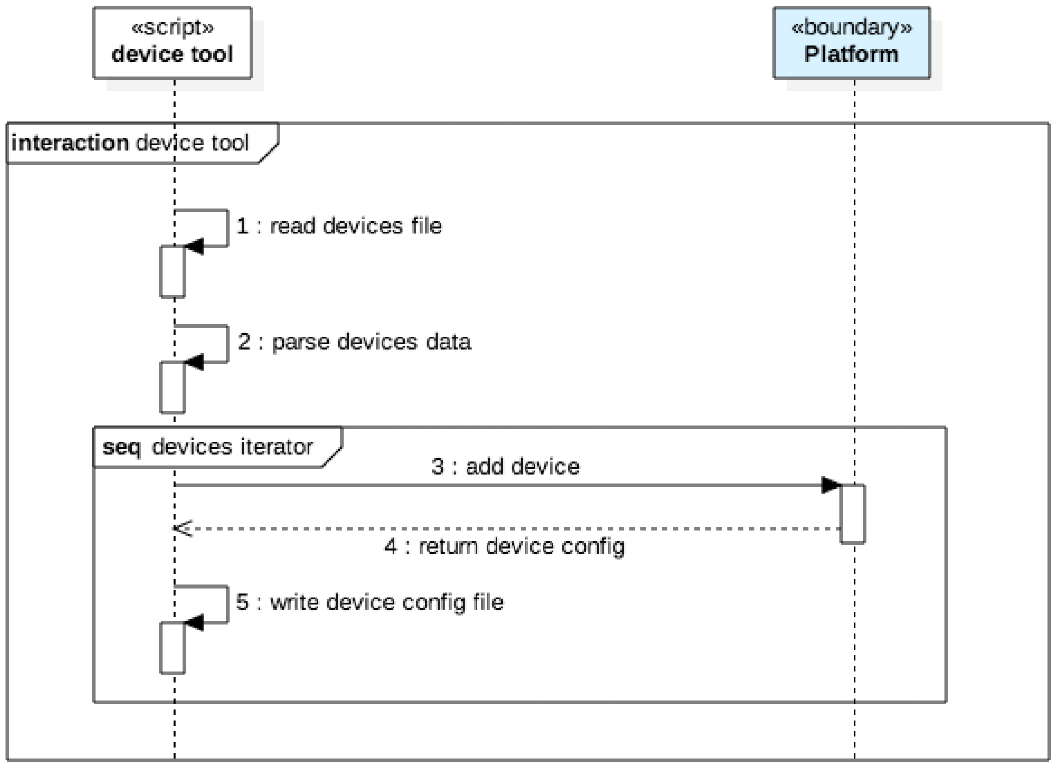


Figure 6.16: Device Tool Overview Diagram

As we can see the Device Tool component starts by reading the devices file, exemplified in Table 6.6 from the specified path and parses the data. Then it iterates each device and requests the Platform module to add a new device with the properties read from the file. The Platform response contains a buffer data that must the read until the last frame. Afterwards, the Device Tool writes the data in the buffer to a local file. As a result of this process we can see the devices added in the Platform and a set of configuration files downloaded, exemplified on Table 6.7, for each device added.

Table 6.6: Devices Tool - Input example file. Legend: Name - description of the device name. Type - identifier of the device type. (X,Y,Z) - coordinates in meters relative to the room. LId - unique identifier for smartlights. BId - unique identifier for estimotebeacons. The last two values are unique for each type of device. Furthermore they will be used by the Platform to communicate with the physical devices.

Name	**Type**	**X**	**Y**	**Z**	**LId**	**BId**
Workstation 8	estimotebe	5.50	2.06	0.75	-	QW
Workstation 8	smartlights	5.83	2.53	1.20	AF	-

Table 6.7: Devices Tool - Output example file

Name	**Description**
server-name	Address of the server endpoint
deviceId	Unique identifier of the device
device-name	Description of the device name
mqtt-sub-topic	Name of the MQTT topic for the device to subscribe
mqtt-pub-topic	Name of the MQTT topic for the device to publish
mqtt-ep	Address of the MQTT endpoint
auth-token	Token used for device authentication over MQTTS
refresh-token	Token used for refresh the new token when the current expires

6.5 Summary

In this chapter the entire process from the conceptualisation to the implementation phases of the final system was presented. We wanted to present all the phases and justifications of our choices. Although this process was based on our concrete case study, we believe that the result of this process can be used in future applications, since all the design and architecture of the system was based on scalability and maintainability requirements. As a result of this process we have:

- A rigorous requirements analysis of the functional and non-functional requirements the system has to guarantee;
- A system architecture design justified through the analysis of requirements and different views;

- A study of recommended comfort values based on similar case studies, medical studies or standards, for a place with the characteristics of our case study;
- A description of the control rules and mechanisms used;
- A description of the components and their interaction resulted from the implementation phase.

CHAPTER 7

Evaluation and Results

This chapter presents the evaluation of our work in several dimensions. It starts by showing the results of the questionnaire conducted to collect the opinion of the occupants of the case study (section 7.1). Then, it presents the analysis of the temperature and brightness data (section 7.2). In the sequence, it presents the data regarding the mechanism used to detect presence and the events associated with the presence of occupants (section 7.3). Lastly, it presents the before and after energy consumptions and associated costs (section 7.4).

In this chapter, we present the results of the questionnaire carried out in the initial phase of the requirements analysis. The questionnaire helped us to collect some features that the system should provide, and that the occupants were generally satisfied with the office. In sequence, we present the results regarding the data collected about the devices states, over two periods of time without automation (from July 10 to 23 July) and with automation (from July 24 to 6 August). With these results we observed that there are several hours the devices are consuming energy unnecessarily, and the impact that the system had in reducing these periods. Finally, we present a cost simulation of the implementation of this system.

7.1 Questionnaire

At the beginning of the requirements analysis phase we made a questionnaire to the office occupants. The questionnaire goals were to understand if the occupants were satisfied with some aspects of the workplace and to understand if they would like to get suggestions to improve their satisfaction level from system to be developed.

One of the questions presented in the questionnaire asked the occupants what is the physical aspect they consider the most important in a workplace. The presented options were **Comfortable temperature**, **Freedom from noise**, **Good luminosity** and **Good ventilation**. Figure 7.1 presents the results of this question. We can observe that a comfortable temperature and freedom from noise are considered the most important aspects in workplace by the occupants.

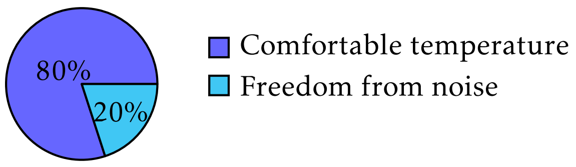


Figure 7.1: Important physical aspects

To complement the above question, the questionnaire presented another set of questions to understand the level of satisfaction for each physical aspect. Therefore, the occupants had to classify the temperature, luminosity, noise and ventilation within the ranks **Very dissatisfied**, **Dissatisfied**, **Satisfied** and **Very satisfied**. Figures 7.2 to 7.5 present the satisfaction levels for these variables. We can observe that the office occupants are satisfied with the current levels. However, one of the goals of this system was to control the office devices in order to improve the occupants comfort. With these results we realise that, since the occupants are satisfied with the current levels provided by the office environment, our system should be able to maintain these levels in a more effective way.

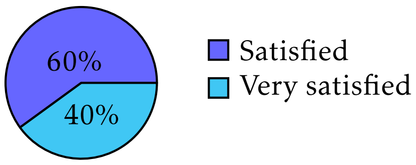


Figure 7.2: Temperature satisfaction level

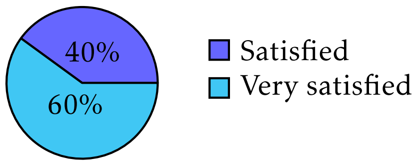


Figure 7.3: Noise satisfaction level

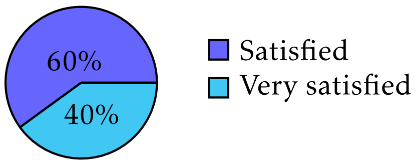


Figure 7.4: Luminosity satisfaction level

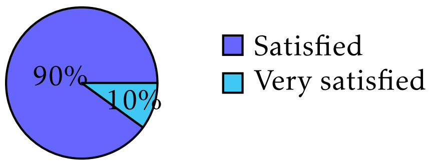


Figure 7.5: Ventilation satisfaction level

Although the occupants were satisfied with the office environment levels, there are some key aspects that need an improvement. Note in Figure 7.6 that 60% of the occupants answer that prefer to work with natural light and 40% prefer artificial light or a mix of the two light sources. In this situation we have to evaluate individual preferences of each occupant, in order to adjust the system according to their needs. However, the system by default must adopt the least energy cost method.

After evaluating the satisfaction levels provided by the office environment, we asked the occupants what were the state of some devices when they arrived at work in the morning. In Figure 7.7 we can observe that 80% of the occupants find the lights turned

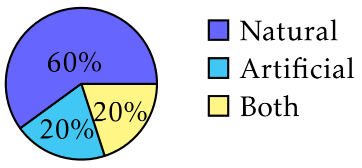


Figure 7.6: Preferable light source

on when they arrive at work in the morning. An important aspect the system should take care of, because at morning if no one is present then it does not make sense to have the lights turn on. We also asked the occupants how they find the air condition system when they arrive at work in the morning. In Figure 7.8 we can observe that all the occupants answer that the air condition is turned off when they arrive. In a future case study, it will be interesting to predict occupants arrival in order to turn on the air condition system, so that it can provide a comfortable temperature even before they arrive at work.

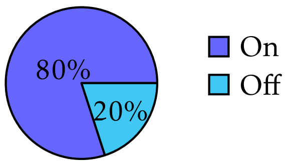


Figure 7.7: Lights state at morning

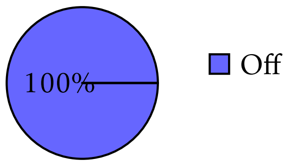


Figure 7.8: Air condition system state at morning

In the questionnaire we asked the occupants if they would like to get suggestions for some actions, to help the system improve their comfort or to reduce energy consumption costs. We adopt to use notifications to suggest the occupants of some actions, since our case study does not have the mechanism to automatically control windows. So we asked the occupants if they would like to get notifications to open or close the windows in order to improve the office environment comfort levels. As we can observe in Figure 7.9, all the occupants answered yes to temperature, luminosity, ventilation variables. Therefore, we had the permission to enable the system to make suggestions to the occupants in order to take advantage of natural resources and turn off office devices.

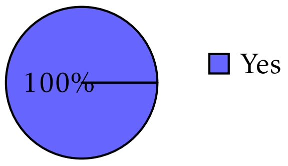


Figure 7.9: Notifications to open/close windows

Lastly, the office has a frequently used coffee machine. Thus, we had the interest to understand if the occupants would like the coffee machine to be turned on when they arrive at the workplace. In Figure 7.10 we can observe that 90% of the occupants would like the coffee machine to be turned on and the other 10% do not. It was noticed what the occupants who answered that they do not want the coffee machine to be turned on at morning, is due to the fact they do not drink coffee at all. So it is an irrelevant improvement for these occupants.

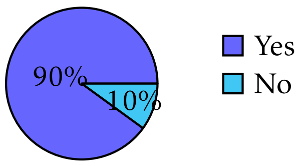


Figure 7.10: Coffee machine turned on at morning

The questionnaire results were helpful in the earlier phase of the system requirements analysis. The system design, architecture and implementation were based on the results obtained from this questionnaire. However, not all aspects were met due to the inability to accommodate all the required equipment. In any case, the system architecture is prepared so that later we can integrate all the missing equipment to help us reach all these aspects. In short, we observe that in a general way the occupants are satisfied with the current comfort conditions provided by the office environment, although there are some aspects relative to temperature and luminosity that can be improved by adopting methods to predict occupants behaviour or by making intelligent suggestions.

7.2 Environment luminosity and temperature

After analysing the occupants opinions, we proceed to the analysis of sensors registered data about the office environment. Once again this analysis helped us to understand that some key aspects need some improvement. Figure 7.11 presents a luminosity chart from a workstation placed near the window, for a period of 14 days. Note that almost everyday the sensor presents luminosity levels too high for a workstation, being possible to observe values between 2000 to 7000 lux.

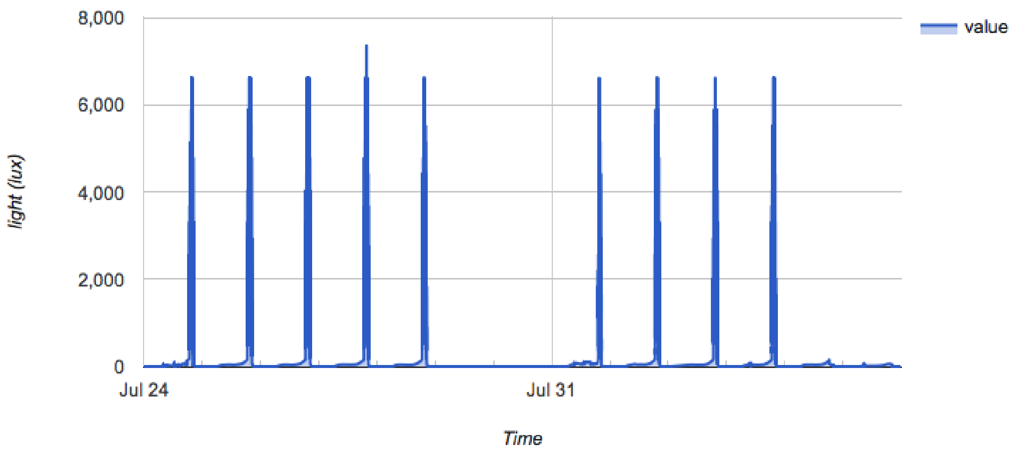


Figure 7.11: Luminosity level of a work station near the window

Decreasing the chart scale to one day, represented in Figure 7.12, we observe that these high values happen between 18:00h and 20:00h and present sudden changes in luminosity levels. This behaviour is caused by the outdoor luminosity coming from the sun light. In this specific case it is important to suggest the occupant to close the window when this event is verified. For comparison, Figure 7.13 presents the luminosity level of one day in a workstation away from the window. In this case we can observe that the luminosity level remains stable for long periods of time, and the variations are gradual over time.

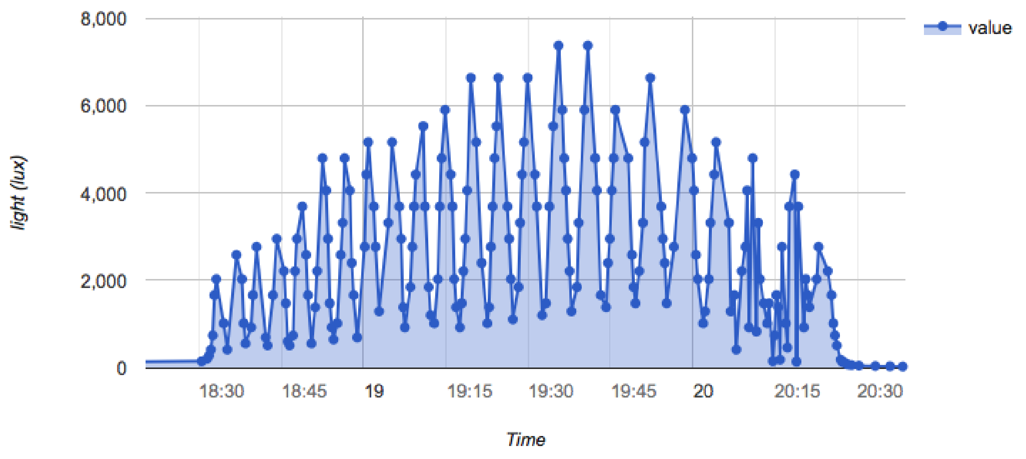


Figure 7.12: Luminosity level of a work station near the window over a day

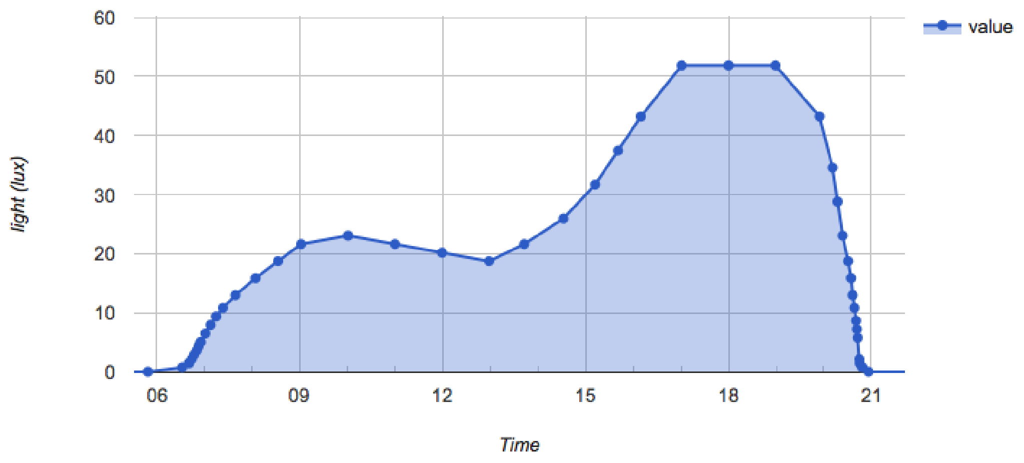


Figure 7.13: Luminosity level of a work station away from the window over a day

We can observe a repetition patten in these events as they are caused by natural elements, in this case the sun. Since the sun is a source of light and heat then it also causes different effects on workstations temperature located at different distances from the windows. In this case we can observe that the workstation near the window reaches higher and lower temperatures comparing to the workstation away from the window. Figure 7.14 presents a chart of a workstation temperature level near the window over a day and Figure 7.15 of a workstation away from the window. We can observe that in both workstations the temperature levels along day has approximately the same gradient. However, the temperature levels reached are very different. In the workstation near the window the maximum temperature was 28.06ºC, the minimum 22.81ºC and the average 25.43ºC. In the workstation away from the window the maximum temperature was 26.13ºC, the minimum 23.5ºC and the average 24.81ºC. Therefore, there are a difference of +1.93ºC in relation to the maximum temperature, -0.69ºC to the minimum and +0.62 to the average.

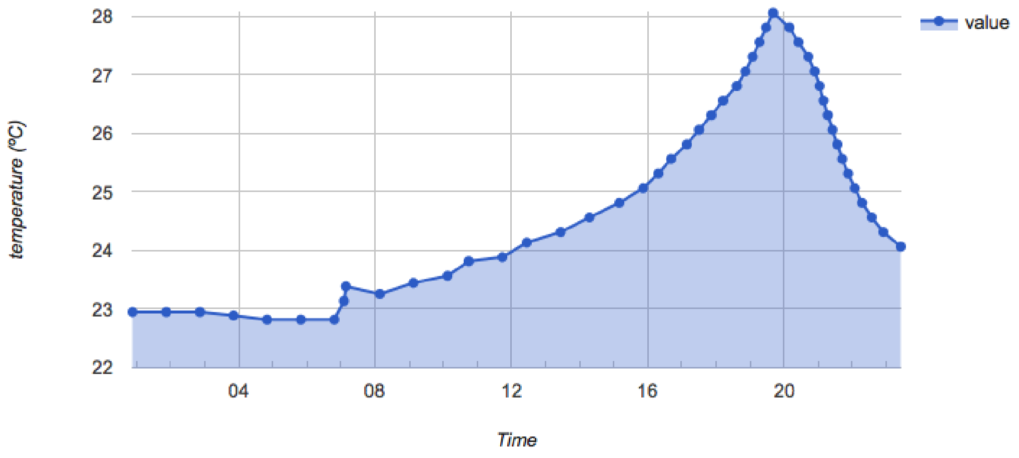


Figure 7.14: Temperature level of a work station near the window over a day

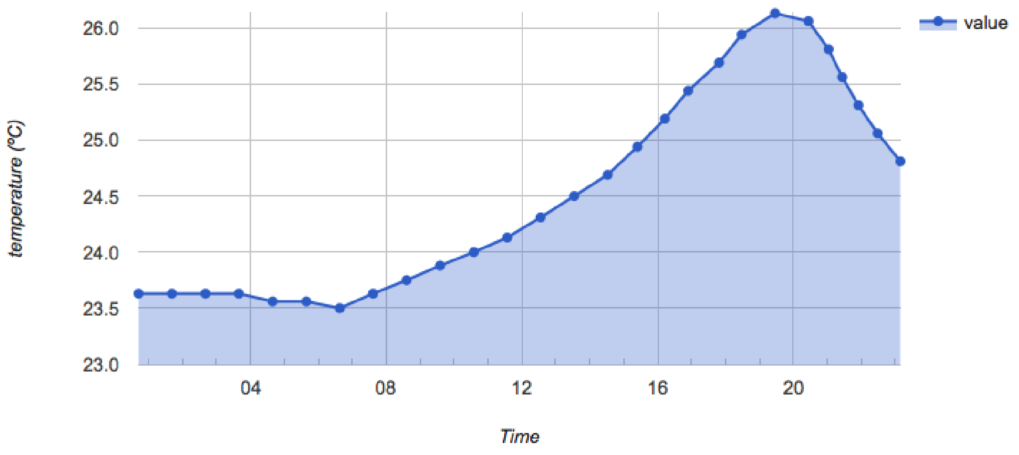


Figure 7.15: Temperature level of a work station away from the window over a day

Analysing the temperature levels records of these stations, we can observe that these differences in temperature levels are maintained throughout the days.

7.3 Presence detection

We intend to show that the mechanism used for presence detection described in Chapter 6 can determine presence detection despite its limitations. To show that the method of reading the energy consumption from the sockets used by the personal computers of each occupant can be used as a presence detection mechanism, the occupants were asked to pass their access card on a card reader every time they enter or leave the office. These data were recorded for comparison with the readings of the sockets. However, we realise that none of these methods is perfect, as both have their limitations. The socket energy consumption may not represent the absence of the occupant if he leaves the office with the computer turned on. The card reader mechanism is also not perfect as it is easy for an occupant to forget to pass the card in the reader. However, combining these two methods we realise that they complement each other. Additionally, it is important to mention that it would be worthwhile to adopt other mechanisms for presence detection. Table 7.1 shows the records about an occupant entering or leaving the office.

Table 7.1: Occupant card reader records

Identifier	**Date**
oc100	2017-09-13 11:08
oc100	2017-09-13 13:11
oc100	2017-09-13 16:16

To further analyse the behaviour of the occupant, we present the energy consumption of the socket designated to the occupant's portable computer. Figure 7.16 presents a chart of the occupant's portable computer energy consumption. In this first example, we can see that the first record of the occupant was on September 13 at 11:08h. In the consumption chart the first record was at 11:08:46h, that is after 46 seconds the card has been record. We can conclude that this occupant arrived at the workplace at 11:08h on September 13. In the analysis of the table and the chart we faced an interesting behaviour. There is a card record at 13:11h, but from the chart we can see that energy consumption continues until 14:22h. To understand the reason off this, we decided to analyse the energy consumption of the monitor used by this occupant.

Figure 7.17 presents a chart of the occupant's monitor energy consumption. We can observe that the first record was at 11:15h. Compared to the occupant's portable computer chart, we can observe that both continue to record energy after the time recorded by the card reader. Additionally, in both energy consumption charts we can see that they begin to record energy consumption around 14:30h. This leads us to think the occupant left the office at the time recorded 13:13h, leaving its portable computer and the monitor turned on. The portable computer and monitor used by this occupant have entered sleep mode due to their inactivity. Both returned to record energy at 14.30h, but there was no record of occupant's card reader. In this case the occupant may have not passed the card on the card reader when he/she returned to the office. Scenarios like these need a more rigorous analysis, to understand the real behaviour of the occupants. Finally, we can see in the energy consumption charts that the portable computer and monitor stop recording

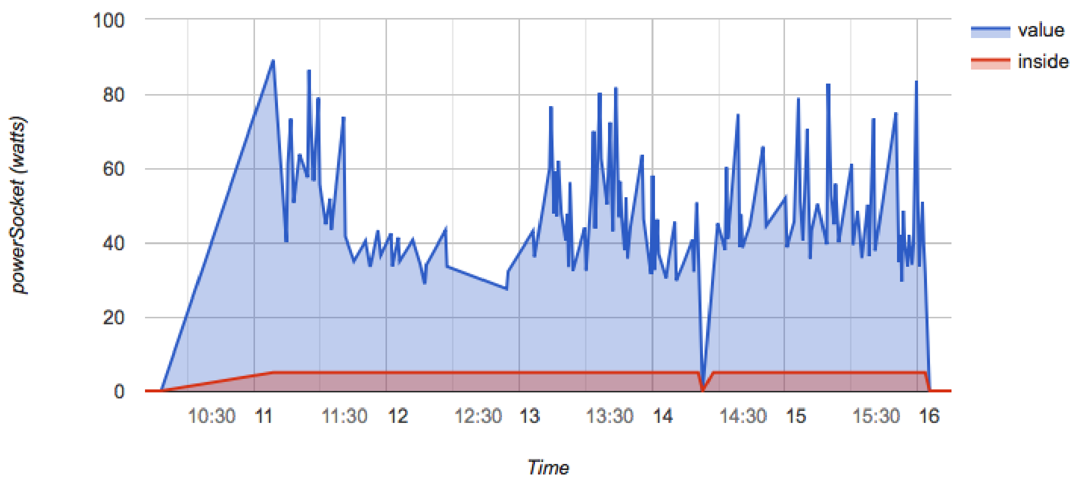


Figure 7.16: Occupant portable computer energy consumption

energy consumption approximately at 16:06h. Later a reading of the occupant's card was registered at 16:16h. As there are no more card and energy consumption records until the end of the day, we can conclude that in this case the occupant left the workplace 10 minutes after turning off his equipment.

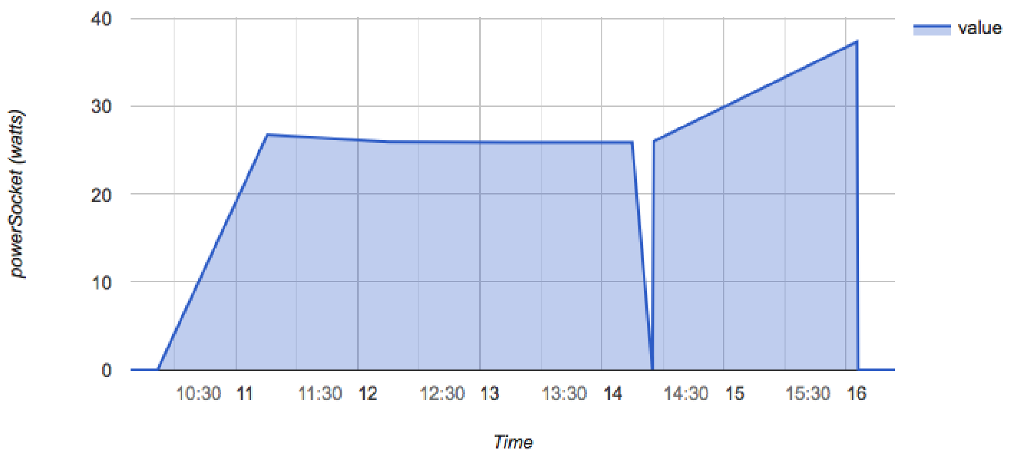


Figure 7.17: Occupant monitor energy consumption

Analysing individually the energy consumption records of the portable computer and monitors' sockets of the other occupants and the card records, we notice that these events happen regularly. However, it is simple to evaluate the first entry of an occupant in the workplace, as well the last exit. It is important to mention the importance of adopting various mechanisms for presence detection. These two approaches used helped us to realise that there are several possible scenarios where the system may not correctly detect presence.

After analysing the data regarding occupants preferences, environment of the case study and the presence detection events, we can now describe how the system reacts to these events. In this specific case we present the behaviour of the light state during a work day, affected by the detection of an occupant in the workplace. Figure 7.18 presents the energy consumption about an occupant's portable computer during its work day. Observing the chart behaviour we see that around 9:30h happens the first energy consumption record. Between 12:51h and 13:24h there is a drop in energy consumption, not being a zero consumption, we can infer that there is a computer connected in sleep mode. Lastly, around 14:30h until the end of the day there is no more energy consumption recorded.

Now we will see the behaviour of the light state during these energy consumption changes. We can observe in Figure 7.19 that the light was turned off until around 9.50h. However, due to presence detected at 9.30h the light turned on. On the other hand we can observe that in the absence of energy consumption between 12:51h and 13:24h, the light turned off. Lastly, the light remained off from 14.38h until the end of the day.

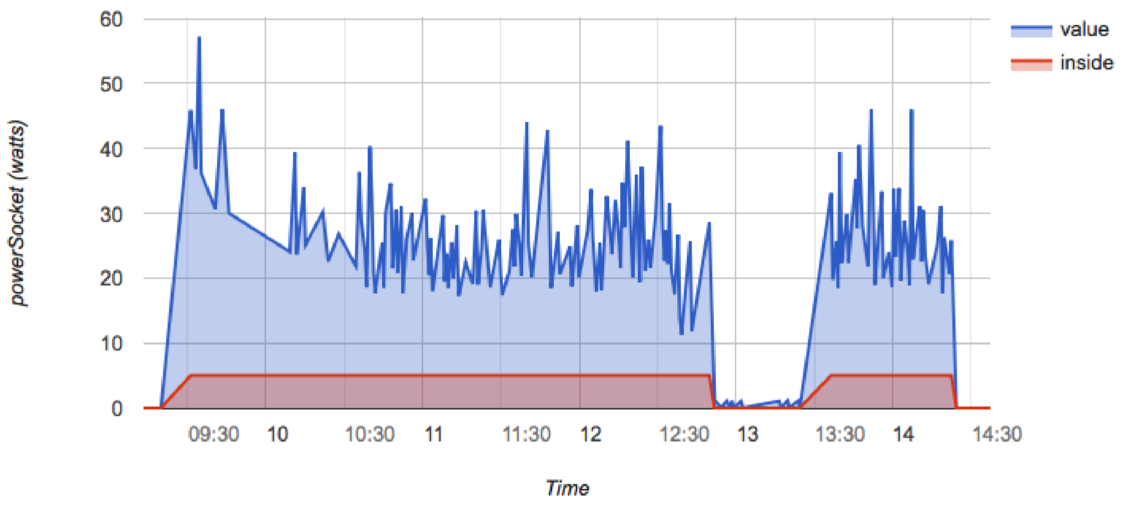


Figure 7.18: Occupant presence detection through energy consumption

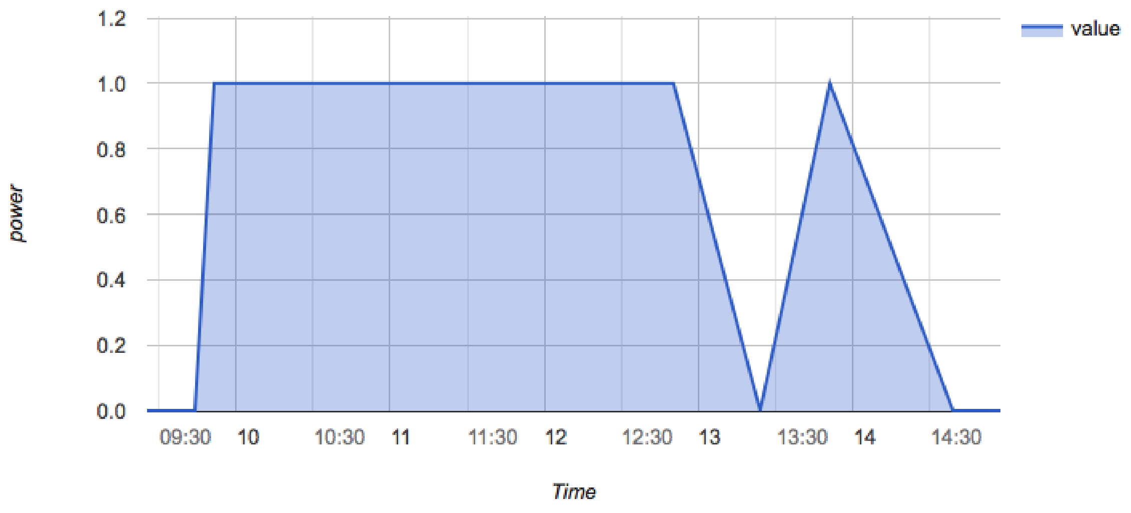


Figure 7.19: Light power state behaviour

Finally we analyse the behaviour of light brightness state against the luminosity levels present in a workstation. It is possible to observe in Figure 7.20 that the luminosity level was below 45 until 10.15h. That is, when the occupant was detected in the office, the current luminosity level was below 45 lux and when the light was turned on at 9.50h the light brightness was set to 0.3 as we can observe in Figure 7.21. However, when the luminosity level reached levels greater than 50 lux at 10.16h a decrease to 0.2 in light brightness can be observed at 10.18h. Later at 10:52h the luminosity levels were bellow 45 lux, therefore it is possible to observe an increase of the light brightness to 0.3. Note that the luminosity level remained less than 45 lux, so the light brightness was not adapted.

Lastly, around 14:00h, the luminosity level reached values higher than 50 lux, in this case there was no light brightness adjustment, because as we saw in the previous chart the occupant left the office at around 14.30h and the light was already off.

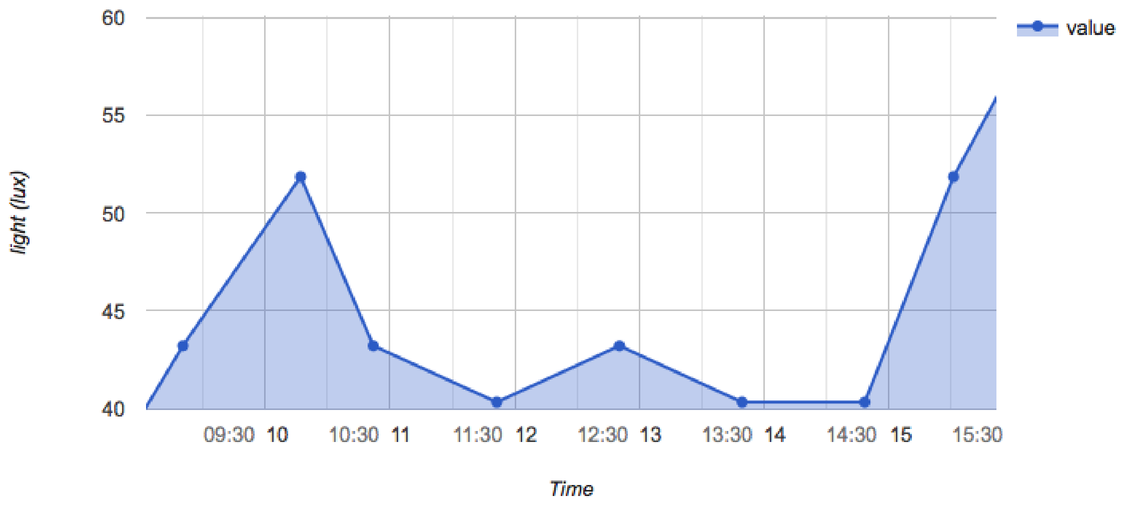


Figure 7.20: Luminosity level of a work station

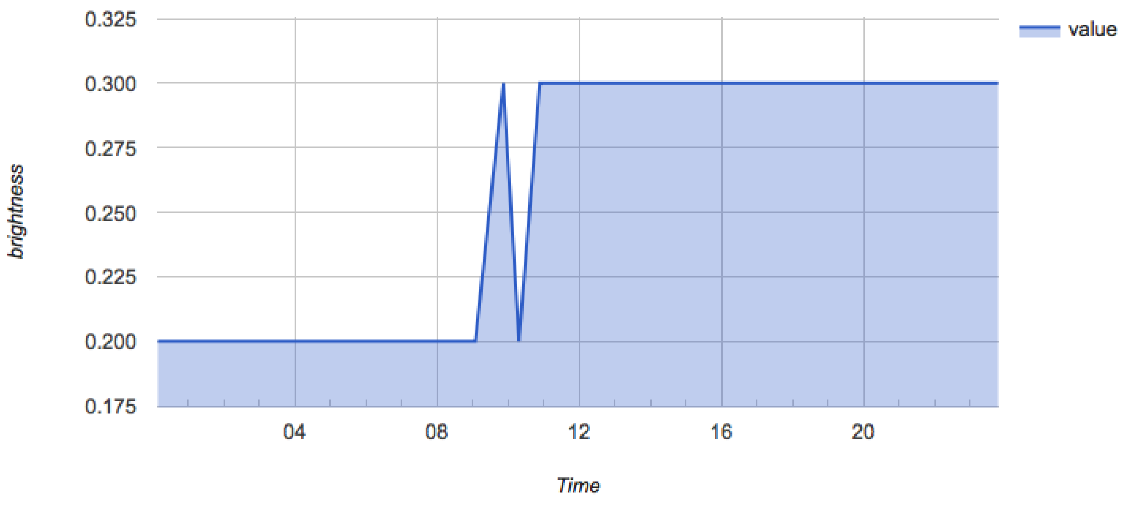


Figure 7.21: Light brightness state behaviour

We recall that as mentioned in Chapter 6 the recommend luminosity levels for a place with the characteristics of our case study is about 500 lux, and we also mentioned that the luminosity sensors used are not the most accurate ones, because they are too sensitive to light changes. For this reason we had to adapt the values to our case study. However, it is important to analyse the system behaviour according to the environment variables and presence detection changes. Thus, we can observe that the system is acting according to the rules for which it was defined to.

7.4 Energy Consumption

7.4.1 Lights

In the previous section we describe how the system adapts the state of the lights according to the detection of the occupants and also to the current luminosity in the office. The system rule of adapting the brightness state or turning on and off a light implies a reduction of energy, due to the fact that the light is turned off whenever the occupant is not in the workstation and when the outside luminosity is sufficient to ensure an adequate level. It is interesting to show the difference that exists in the behaviour of a light before and after the of the system. Figure 7.22 shows the status of two lights during a 14 day period without automation.

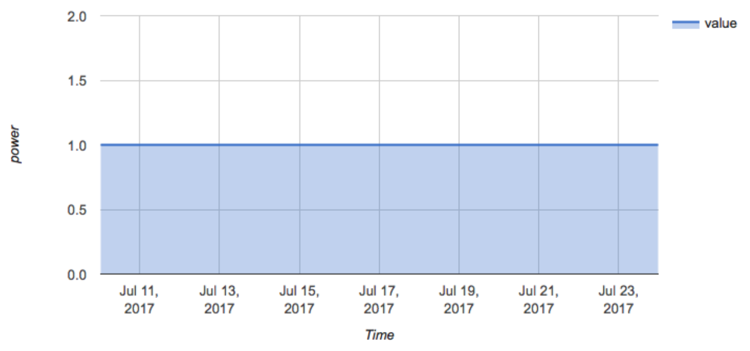


Figure 7.22: Light 1 and Light 2 power state without automation

Note that the lights were always on, but in Figure 7.23 we can observe that Light 1 brightness was 0.15 and in Figure 7.24 the Light 2 brightness was between 0.1 and 0.2. The lights need to keep a constant Wi-Fi connection for communication with the system. The specifications of the light on the manufacturer website, describe that the light has a consumption of $0.7 watts$ on standby mode. Additionally, a light turned off state also has a consumption of $0.7 watts$ and it has a maximum of $11 watts$ [Lif].

Figure 7.25, 7.26, 7.27 and 7.28 show the above lights state in a 14 day period but with automation. It is possible to observe that both lights were off most of the time in comparison with the previous charts. It is also interesting to note that there are differences in the power and brightness values between these two lights. We notice that Light 1 was

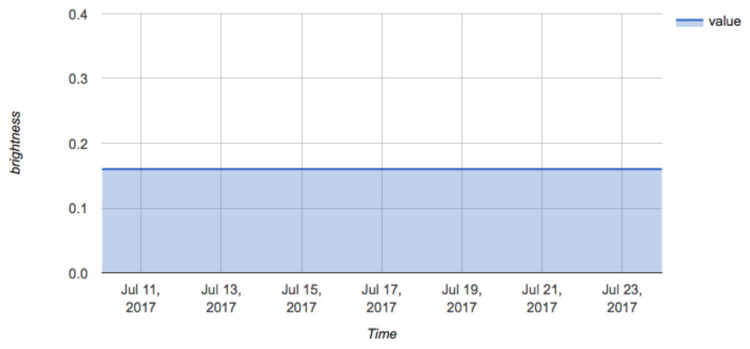


Figure 7.23: Light 1 brightness state without automation

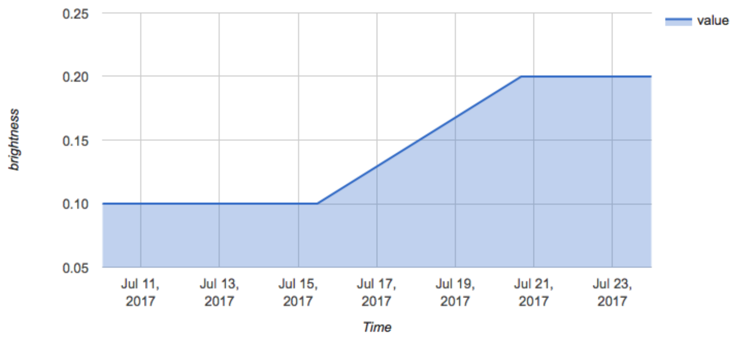


Figure 7.24: Light 2 brightness state without automation

turned on more often than Light 2. This happen due to the fact Light 1 is placed in a workstation occupied more frequently occupants during this period. Additionally, it is also possible to see that Light 1 has lower brightness values than Light 2. This is due to the fact that the Light 1 is located on a workstation closer to the window that receives higher luminosity levels from outside.

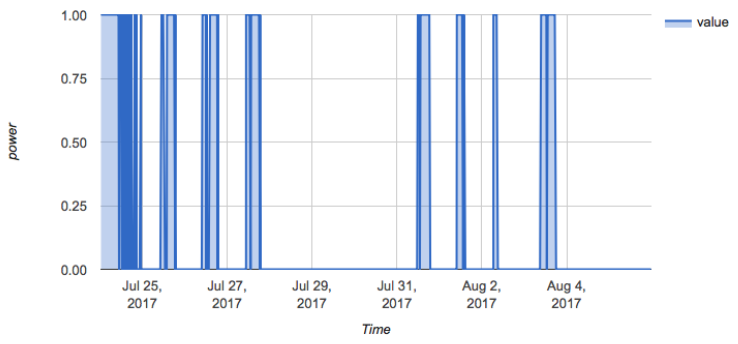


Figure 7.25: Light 1 power state with automation

It is possible to notice that there was a big change in the number of hours these lights were using energy. The used lights have a minimum energy consumption, but it is important to determine the reduction of hours in a scenario with automation. As we saw in Figure 7.22 in the 14 days period without automation both lights were always on and a minimal brightness level for approximately 336 hours. In Figure 7.25 we can

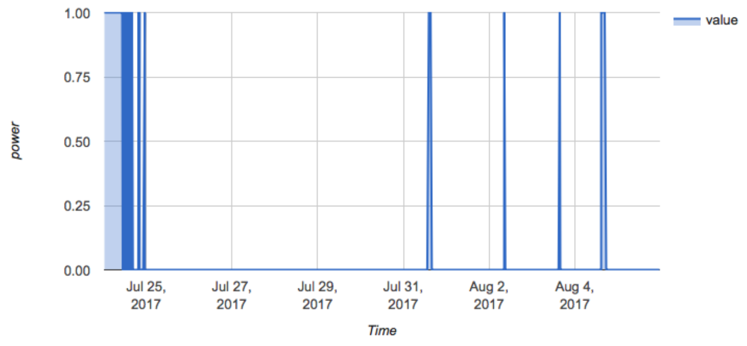


Figure 7.26: Light 2 power state with automation

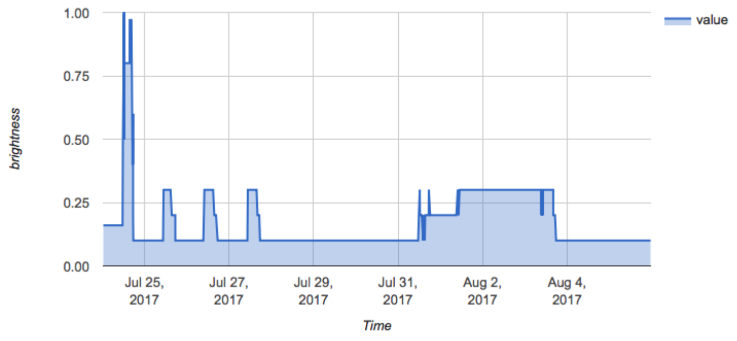


Figure 7.27: Light 1 brightness state with automation

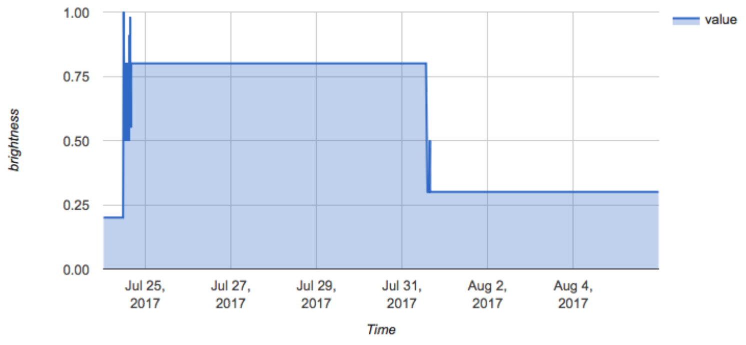


Figure 7.28: Light 2 brightness state with automation

observe Light 1 usage hours reduced to 61 hours. In Figure 7.26 we can observe Light 2 usage hours reduced to 18 hours in a period of 14 days. This means that the Light 1 had a reduction of 81.84% and Light 2 of 94.64% in usage hours.

7.4.2 Monitor

Now we will show the difference of a monitor state present in the office with and without automation. In Figure 7.29 we can observe the monitor state during a 14 day period without automation. The monitor has an energy consumption while active and standby. Thus, it is possible to reduce the monitor energy consumption by turning off its socket. In Figure 7.30 we can observe a reduction on monitor usage hours when the automation is on.

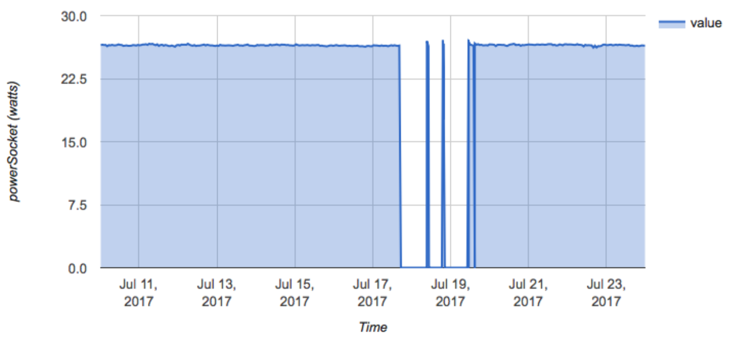


Figure 7.29: Monitor energy usage without automation

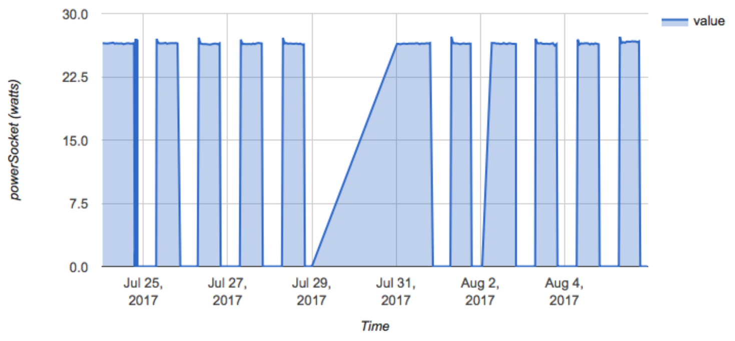


Figure 7.30: Monitor energy usage with automation

It was possible to observe a reduction in the number of hours the monitor was using energy of approximately 309 hours to 132 hours, achieving a 57.28% reduction in monitor usage hours.

7.4.3 Coffee Machine

In the office there is a coffee machine and we notice that it is the equipment that consumes most energy. We also note that the coffee machine is left turned on many times during the day, as we can see in Figure 7.31. In Figure 7.32 we can observe a reduction on the energy usage by coffee machine resulted from turning off the coffee machine socket everyday at evening when the office is empty.

To determine the usage periods of the coffee machine we had to interleave the data between its socket power state and energy consumption. This was necessary in order to evaluate the number of hours the coffee machine was working in the scenario without automation. In the scenario with automation the coffee machine was always on but the system change its state by turning on or off the socket. It was possible to observe a reduction in the number of hours the coffee machine is using energy of approximately 195 hours to 75 hours, achieving a 61.54% reduction in coffee machine usage hours.

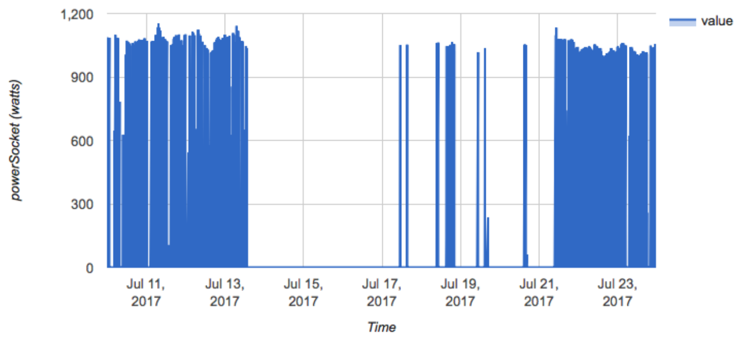


Figure 7.31: Coffee machine energy usage without automation

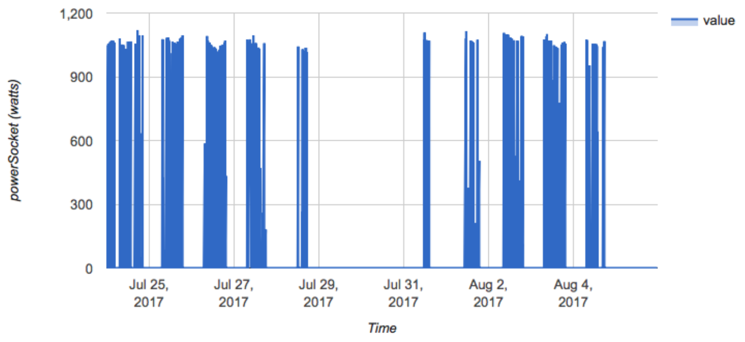


Figure 7.32: Coffee machine energy usage with automation

7.4.4 Other equipment

There are other equipments in the office that can not be turned off by the system in order to contribute to a reduction of energy consumption. In this case the aquarium and the agents used to communicate with the physical devices. Therefore, for both automation scenarios we can observe that every socket used by these devices was turned on (represented in Figure 7.33).

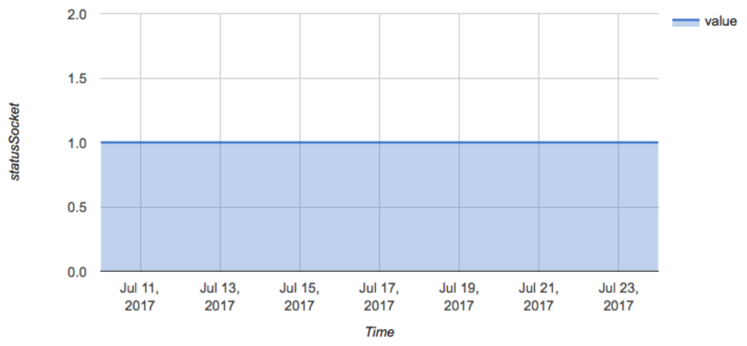


Figure 7.33: Aquarium and Agents power state

7.4.5 Energy cost

One key aspect for someone who wants to implement an IoT solution is to know the associated costs and if it is worth it. We observe in the above sections that in fact the system with simple rules can actually reduce the time of use of the office equipments. In this section we present the impact on the energy consumption cost by reducing the equipments usage time.

Initially we had to determine what plans and consumption rates were applied in our case study. Since this information could not be obtained, we had to simulate using approximate values for kWh cost. We start analysing the existing plans and taxes and we realise that there are several options and the costs can be different at different times of the day [Edp]. To simplify the analysis we decided to present results considering a plan where the kWh cost is constant. In this case the 24 hours are divided into a single fraction of time, **Simple** with a cost of $0.1652(euro/kWh)$.

The next step is to determine the time interval that the equipment has been turned on. Thus, taking into account the usage hours of each equipment, presented in the previous section, it is possible to calculate their energy consumption. The energy in kilowatt-hours per day $E(kWh/day)$, represented in equation 7.1, is equal to the power P in watts (W) times the number of usage hours per day t divided by 1000 watts per kilowatt.

$$E(kWh/day) = P(W) * t(h/day)/1000(W/kW) \tag{7.1}$$

After we determine the energy consumption per day of each device, it is possible to calculate their energy cost. The energy cost per day in euros $Cost(euro/day)$, represented in equation 7.2, is equal to the energy consumption per day $E(kWh/day)$ times the energy cost of $1kWh$ in cent/kWh divided by 100 cent per euro.

$$Cost(euro/day) = E(kWh/day) * Cost(cent/kWh)/100(cent/euro) \tag{7.2}$$

For example, the coffee machine was turned on 195 hours in a period of 14 days in the scenario without automation. This results in an average usage of 13.93 hours per day. Additionally, from Figure 7.31 we can observe that the coffee machine has an average power of $1100watts$. Therefore, its energy consumption is $15.323(kWh/day)$ resulting in an energy cost of $2.53(euro/day)$, using the **Simple** plan with rate of $0.1652(euro/kWh)$. In the scenario with automation the coffee machine was turned on 75 hours. This results in an average usage of 5.36 hours per day. Therefore, its energy consumption is $5.896(kWh/day)$ resulting in an energy cost of $0.974(euro/day)$. It is possible to observe that we achieve a 61.50% reduction in coffee machine energy cost per day.

Table 7.2 presents the calculated cost per day for the rest of the equipments. To get a notion of the costs, we presented the cost per month. Note that the Monitor 2 has two types of usage hours, h_a and h_b. This is because the Monitor 2 has an average of $27watts$ while operating and an average of $1watts$ in standby. In the scenario with automation this standby mode does not exist because the monitor's socket is turned off. Figure 7.34, Figure 7.35 and 7.36 illustrates the differences between the devices usage hours and energy cost per month.

Table 7.2: Equipments energy consumption and cost per day. Legend: CM - Coffee Machine, M1 - Monitor 1, M2 - Monitor 2, L1 - Light 1, L2 - Light 2, AT. Off - Average hours per day without automation, AT. On - Average hours per day with automation, * Values explained later.

	CM	**M1**	**M2**	**L1**	**L2**
AT. Off	13.93h	22.07h	$6h_o + 18h_s$	24h	24h
Cost per day	2.53€*	0.098€	0.029€	0.04€	0.04€
Cost per month	75.94€*	2.95€	0.86€	1.30€	1.30€
AT. On	5.36h	9.43h	6h	4.4h	4.4h
Cost per day	0.974€*	0.042€	0.026€	0.007€	0.007€
Cost per month	29.22€*	1.26€	0.77€	0.24€	0.24€

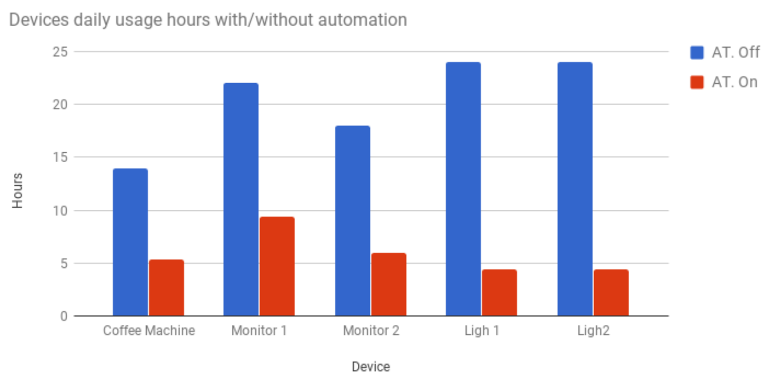


Figure 7.34: Devices daily usage hours with/without automation

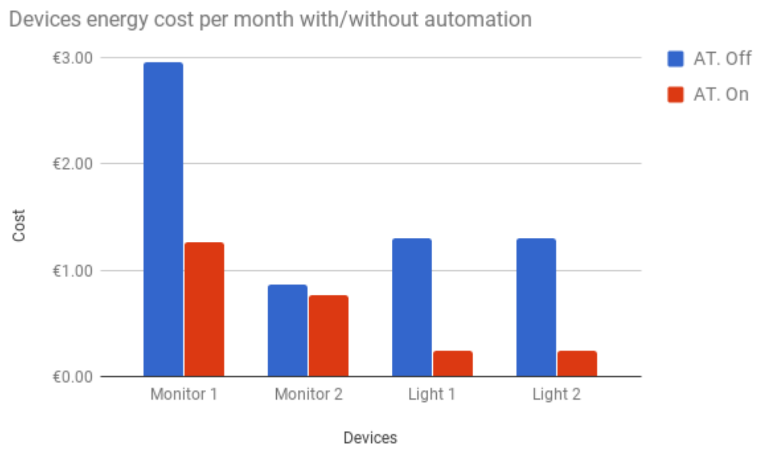


Figure 7.35: Devices energy cost per month with/without automation

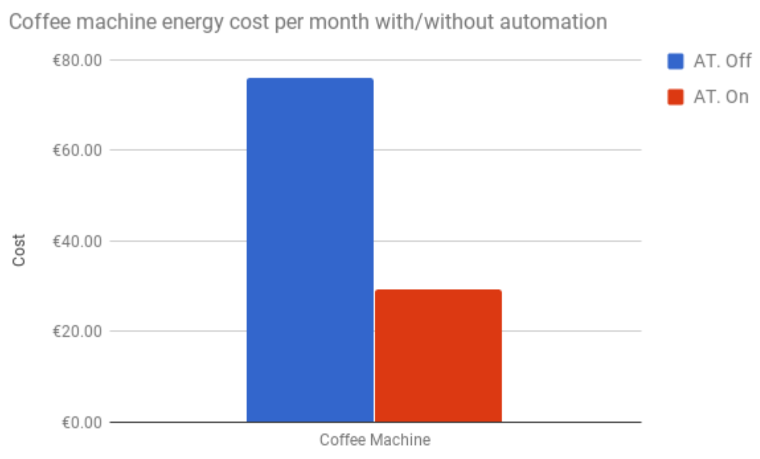


Figure 7.36: Coffee machine energy cost per month with/without automation

Note that the Table 7.2 does not present the energy costs for the agents used, due to the fact they are part of the solution itself. The occupants portable computers and the aquarium are also not presented in the table, due to the fact it just presents the cost of equipments that the system can control. It is also possible to observe that the coffee machine has a high energy cost per day. This happens because the coffee machine has an electrical resistance that heats over time. Thus, the coffee machine does not have a constant power consumption and it is possible to observe periods of consumption peaks and other periods with lower values. However, the equation 7.1 to calculate the energy consumption of a device uses the device power in watts and the period of time the device is consuming energy. In a more realist scenario to calculate the exact value it is required to measure and calculate its cost over the different energy consumption periods.

The total energy cost for a month with the equipments presented in the table for the scenario without automation, results in an energy consumption cost of 82.35€. After this analysis we can observe that is possible to reduce the energy costs to 31.73€, by reducing the equipments number of usage hours.

To understand if the solution is worth it we have to take into account the purchase cost and calculate the energetic cost of the installed equipments. Table 7.3 presents the required equipments to be installed in the office. Additionally, we have to calculate the energy consumption cost for the Raspberries taking into account that they will be running 24 hours a day.

Thus, three Raspberries operating 24 hours a day, results in an energy consumption cost per month of 0.7134€. Note that the energy cost per month of the scenario without automation is 82.35€ and the scenario with automation reduce this cost per month to 31.73€. To this cost we need to add the energy cost of the 3 Raspberries that results in an energy consumption cost per month of 32.44€, having a difference of 49.91€ comparing with the cost of scenario without automation. Thus, the initial cost to ensure the required equipments for this scenario was 676.4€ meaning that at the end of 13.5 months we have

Table 7.3: Solution equipments list

Name	**Quantity**	**Price**	**Power**
Raspberry Pi	3	43.98€	$2watts$
LifX Light	2	58.93€	$11watts$
Outlets	4	85.60€	-
Estimote Beacon	Pack	84.20€	-
Total	-	676.4€	-

this amount (represented in Figure 7.37).

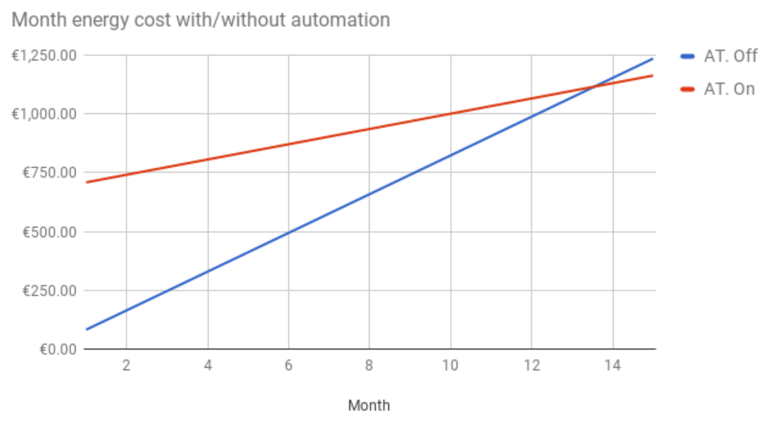


Figure 7.37: Month energy cost with/without automation

Notice that in these scenarios were involved some devices of the room and summing the energy costs of the occupants computers or the aquarium for example, will not affect the above result since the system does not control any of them. However, we calculate an average energy consumption cost for the rest of the devices to understand that over time the initial investment is worth it. The values are mere estimates and should not be taken strictly. This would make sense in a controlled scenario with these devices keeping these consumption costs for a significant period of time. A real case scenario is subject to many variables, such as the use of more devices, the occupants behaviour, the environment temperature and luminosity that change over different seasons. Thus, the presented analysis is according to this case study in specific. Therefore, another scenario with other devices and other variables can have different results.

7.5 Summary

The survey carried out by the office occupants allowed us to establish the functionalities that the system had to guarantee. The analysis of the devices data allowed us to observe that there are several aspects where the system can act to ensure a better comfort in the office and also to reduce the energy consumption. It is also possible to observe behaviours coming from events of natural cause and from the office utilisation. Later, this data can be used, adopting learning techniques, in order to apply rules for energy consumption and increase the comfort in the office environment. We observed that there are several periods of time where the devices consume energy unnecessarily. Thus, it is possible to reduce these periods of time by applying control rules. In the final section of this chapter we presented a costs simulation. This simulation shows us how to determine if it is worth an investment to build such systems. The example presented is applied to a small number of devices and to a controlled case study, and for a real case it would be necessary to take into consideration many other variables. However, the key idea to keep from this analysis is that, the system can actually reduce the number of hours the devices consume energy, even using simple rules, and that implies a reduction of energy consumption.

Chapter 8

Conclusion

This chapter closes the dissertation by summarizing the results (section 8.1 and section 8.1), while highlighting its limitations (section 8.3) and pointing to future directions of this research (section 8.4).

8.1 Summary

We have started our work by departing from the need to implement an architecture to support IoT in BAS that could fit the needs of our case study of Office Automation - the SmartLab. We learnt in the process, thanks to a profound comparative analysis, that a new implementation from scratch would not bring much more advantages than the existing architectures. The existing platforms had, as expected, standard features, however, we have identified some key aspects related to the constraints of our case study that were considerably different. Our process was to start an analysis to understand which main features an IoT platform should provide. Then we searched for different platforms, and we noticed that they presented different architectures and functionalities.

Due to the wide variety and complexity (lack of support, documentation, training, instability) of existing platforms and their differences, we decided to narrow down our search and focus only on the platforms that could provide a well-defined architecture and well documented regarding the description of their characteristics. As a consequence of the said before, as a deliverable of our work, we presented a complete analysis of the key aspects and features that an IoT platform should provide and a comparative study of their architectures and features provided.

The conducted analysis highlighted the fact that some differences in the platform would significantly influence the choice of the platform to the point that we decide to select two platforms. As explained in this document, the selection criteria led to the selection of two platforms that presented similar features and different development and deployment environments.

In the course of our work, we came to the need to answer to the question of how well each of the chosen platforms could deal with the integration of existing devices to deploy the solution - an IoT based BAS. Therefore, we decided to implement a minimum set of functionalities using both platforms and provided a detailed description, in this document, of the differences found and how these differences could influence tradeoffs both due to the case study and the chosen platform. The implementations brought awareness about factors that affect effort and cost that can influence the decision for someone who wants to build an IoT solution. Thus, after our experience using two different approaches, we present the aspects we consider fundamental when choosing a platform.

The second goal of this dissertation was to develop a system concerned with minimising energy consumption costs taking into account the comfort of the occupants in a specific and controlled case study - the SmartLab. We started by the requirements analysis to understand what functional and non-functional requirements the system should ensure. To assess the level of satisfaction of the office's occupants, before (no automation) and after the runtime of the implemented platforms, and how, in their perspective, the system affected their comfort, we proceeded with an initial survey, by means of subject questionnaires. In this way, we wanted to understand whether the occupants were satisfied with the office comfort conditions and what features they would like the system to provide. After analysing the questionnaire results, we noticed that in general the occupants were satisfied with the comfort conditions of the office. Therefore, the new system had to be able to keep this level of satisfaction but in a more efficient way.

The next step was to correctly profile the stakeholders that interacted with the office and what requirements they expected from it. Thus, we elaborated a set of user stories that reflected the requirements expected by the stakeholders to validate with them. We took care not to restrict ourselves to define only the functionalities about comfort and automation. Therefore, we described the expected functionalities of an IoT solution, namely for devices, users and application management. The main reason for this decision is because a description of the expected requirements for configuration and management may influence in choosing a platform. The next step was to describe the functional and non-functional requirements of the system. In the functional requirements, we filtered those that our final system should focus on and we excluded those that were already provided by the platforms. In the non-functional requirements, we choose those that we consider most important considering the key functionalities required by the occupants and our case study. Finally, we designed the system architecture based on the styles and views that went according to the analysis and treatment of the functional and non-functional requirements.

In the implementation phase, it was necessary to evaluate the comfort values, recommended by authorities, medical studies or standards, for a place with the characteristics of our case study and the methods available for control and automation of these variables. Our analysis helped us to understand which methods we should adopt in the functionalities of our system and what mechanisms can simplify their implementation.

At the beginning of the implementation phase, there was an extended period of time for collecting data about the devices installed in the office to better understand the technology setup, office behaviour as a system, and relevance of the extracted data, allowing us to tune the adequate frequency of data collection among other aspects. The data collected without automation would serve as a baseline for future comparison with data with automation. At the end of the implementation phase of the chosen platforms, we made several tests to ensure the system still met the initial requirements. Lastly, we collected data about the same devices for a long period, but this time with automation with control rules. To understand if the system had impact and meet the defined rules we analysed and compared the data provided from the both periods. The data relative to phase without automation, showed us that there were several hours that the devices consumed energy when it was not necessary. The control rules implemented at this phase are superficial but have managed to significantly reduce the number of hours that the devices are consuming energy. Therefore, it was possible to observe a reduction on the energy consumption.

The results obtained helped us to realise that in fact there are several aspects that can be improved in order to reduce energy consumption and maintain occupant comfort. An initial investment in the implementation of these systems may involve different equipments and development effort to achieve the desired solution. However, in the long term it is worth the effort and initial investment on these systems, since they can actually reduce the energy consumption and guarantee good conditions for the office occupants.

8.2 Contributions

The main contribution of this dissertation is the results of our deep study on how to choose adequate technology and implement BAS that use IoT.

Through the analysis of existing architectures and different platforms, it was possible to contribute to a guide, which presents fundamental key aspects that helps with dealing with tradeoffs while choosing the solution to be adopted, as well as the impacts that these aspects may have at cost and effort level in all the development phases. The platforms' comparative analysis and the highlighted features they provide can serve as a comparison document for other future platforms. We also produced an extensive explanation about the concepts of IoT middleware and Fog Computing, their advantages and how they are related to existing IoT platforms.

To deal with the complexity of the technologies involved and the multiplicity of use scenarios non-related with BAS, made us fix a particular case study of Office automation with our SmartLab project. The analysis of requirements and architecture was based on this specific case study, and was designed taking into account non-functional requirements, such as maintainability and scalability. Therefore, we believe that it can be reused as a guide for future applications in similar case studies or even in case studies that involve more components.

The analysis of the results obtained during the scenarios in our case study, with and without automation, creates an opportunity to reason about the case study and design extensions to the system and new applications. During our work, it was possible to abstract valuable office environment values patterns and both devices and occupants behaviours. Thus, it is possible to create rules and learn from these events to provide in the end greater reduction of energy consumption.

8.3 Restrictions

Unfortunately, during the progress of our work, due to unexpected technology constraints, that went against our planning, faced several challenges regarding aspects of the physical equipment. As already mentioned before in this document, there is some equipment used, such as the luminosity sensors, that we later found to be not the most suitable ones for our scenario. Another aspect is that we provide the analysis and logic for control of the office temperature, but it was not possible to test in real devices, due to the absence of air conditioning control devices capable to communicate with our system. The same applies to the aquarium subsystem, it was taken into account during the requirements analysis, architecture design and implementation but due to problems in the aquarium control device, it was not possible to directly control the aquarium with our platform. Additionally, the mechanisms used for presence detection worked but needed a disciplined usage that could be feasible with the gentle help of the occupants (small number), which is a different scenario with a big number of occupants would probably not be possible.

8.4 Future Work

The platforms chosen in our work are ideal to be extended with several applications and functionalities. With a stable case study, and with adequate supporting platforms, we have now a well-formed basis for prototyping solutions and complement the requirements analysis and design of the system architecture. The independence of the control module in the chosen platforms, allows us to run and test various scenarios, with other devices. We foresee that it will be interesting to analyse the advantages provided through the implementation of this system, but following the Fog computing architecture. As said, the independence of the control module simplifies and reduces the effort to adopt the Fog computing architecture. The results obtained about the device states, make it possible to analyse patterns of office utilisation and occupants behaviour through the adoption of machine learning and predictive mechanisms. Thus, this analysis can contribute to a better management of the devices and office policies, which in turn might imply a greater reduction of energy consumption and an increase in the comfort provided to the office occupants. In this case study, there are some limitations, namely aspects that can not be fully automated, such as windows (automatic blinds and window open or close). It would be interesting to adopt gamification methods so that the occupants can help the system to contribute to a reduction in energy consumption. As discussed along the thesis, we believe that adopting control strategies can have an impact on energy consumption and occupant comfort. The work carried out in this dissertation already shows us that it is worth the effort and initial investment for the development of these solutions.

Bibliography

[AF+15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376. issn: 1553-877X. doi: 10.1109/COMST.2015.2444095.

[Awsa] *Amazon WS IoT*. https://aws.amazon.com/iot-platform/. (accessed February 4, 2017).

[Ara+16] T. Ara, P. G. Shah, and M. Prabhakar. "Internet of Things Architecture and Applications: A Survey". In: *Indian Journal of Science and Technology* 9.45 (2016). http://www.indjst.org/index.php/indjst/article/view/106507. issn: 0974 -5645.

[AN05] J. Arlow and I. Neustadt. *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2Nd Edition)*. Addison-Wesley Professional, 2005. isbn: 0321321278.

[Ast+16] N. Aste, M. Manfren, and G. Marenzi. "Building Automation and Control Systems and performance optimization: A framework for analysis". In: *Renewable and Sustainable Energy Reviews* (2016). http://www.sciencedirect.com/science/article/pii/S1364032116307365. issn: 1364-0321.

[Att+16] S. Attitalla, V. Choksi, and M. Potdar. "Web and Cloud Based Home Automation Systems: An Overview". In: (2016).

[Awsb] *AWS Architecture*. https://aws.amazon.com/iot-platform/how-it-works/. (accessed February 4, 2017).

[Bas+12] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. 3rd. Addison-Wesley Professional, 2012. isbn: 0321815734, 9780321815736.

[Bol09] F. Boldissar. *Heat Transfer in Aquariums*. http://www.advancedaquarist.com/2009/7/aafeature1. 2009.

[Bra+05] M. R. Brambley, D Hansen, P Haves, D. Holmberg, S. McDonald, K. Roth, and P Torcellini. "Advanced sensors and controls for building applications: Market assessment and potential R&D pathways". In: *Pacific Northwest National Laboratory* (2005).

[Cae+17] D. S. Caetano, D. E. Kalz, L. L. Lomardo, and L. P. Rosa. “Evaluation of thermal comfort and occupant satisfaction in office buildings in hot and humid climate regions by means of field surveys”. In: *Energy Procedia* 115 (2017). http://www.sciencedirect.com/science/article/pii/S1876610217322166, pp. 183 –194. ISSN: 1876-6102.

[Chi16] M. Chiang. “Fog Networking: An Overview on Research Opportunities”. In: (Jan. 2016).

[Chu+00] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*. Springer, 2000. ISBN: 978-1-4613-7403-9.

[Comcea] I. Comparison. *IBM API life cycle Management*. https://www.ibm.com/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.overview.doc/api_management_overview.html. (accessed September 1, 2017).

[Comceb] I. Comparison. *IBM Decision Optimization Documentation*. https://console.bluemix.net/docs/services/DecisionOptimization/DecisionOptimization.html. (accessed September 1, 2017).

[Comcec] I. Comparison. *IBM Geospatial Analytics Documentation*. https://console.bluemix.net/docs/services/geospatial/index.html. (accessed September 1, 2017).

[Comced] I. Comparison. *IBM Geospatial Analytics Documentation*. https://ibm-graph-docs.ng.bluemix.net/index.html. (accessed September 1, 2017).

[Comcee] I. Comparison. *IBM Identity and Access Management*. https://www.ibm.com/blogs/bluemix/2017/05/introducing-identity-access-management/. (accessed September 1, 2017).

[Comcef] I. Comparison. *IBM Machine Learning Documentation*. https://console.bluemix.net/docs/services/PredictiveModeling/index.html. (accessed September 1, 2017).

[Comceg] I. Comparison. *IBM Streams Documentation*. https://www.ibm.com/support/knowledgecenter/en/SSCRJU_4.2.0/com.ibm.streams.welcome.doc/doc/kc-homepage.html. (accessed September 1, 2017).

[Cor14] I. Corporation. *Affordable Building Automation System Enabled by the Internet of Things (IoT)*. http://www.intel.com/content/www/us/en/internet-of-things/blueprints/iot-building-automation-system-blueprint.html. 2014.

[Des] *Despacho (extrato) nº 15793-F/2013*. https://www.academiaadene.pt/download/pt/despacho-15793-f2013-zonamento-climatico-e-respetivos-dados.pdf. 2013.

[Det+11] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. “From Game Design Elements to Gamefulness: Defining "Gamification"”. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek ’11. http://doi.acm.org/10.1145/2181037.2181040. ACM, 2011, pp. 9–15. ISBN: 978-1-4503-0816-8. DOI: 10.1145/2181037.2181040.

[Dom+16] P. Domingues, P. Carreira, R. Vieira, and W. Kastner. “Building automation systems: Concepts and technology review”. In: *Computer Standards and Interfaces* 45 (2016). http://www.sciencedirect.com/science/article/pii/S0920548915001361, pp. 1 –12. ISSN: 0920-5489. DOI: \url{http://dx.doi.org/10.1016/j.csi.2015.11.005}.

[DBce] S. M. Dominic Betts. *Azure IoT*. https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-what-is-azure-iot. (accessed January 28, 2017).

[Edp] *EDP - Tarifários*. https://energia.edp.pt/particulares/energia/tarifarios. (accessed September 10, 2017).

[Eva12] D Evans. *The Internet of Things How the Next Evolution of the Internet is Changing Everything*. 2012.

[Fac17a] Facebook. *Flux, Application Architecture for Building User Interfaces*. https://facebook.github.io/flux/docs/in-depth-overview.html. 2017.

[Fac17b] Facebook. *React, A JavaScript Library for Building User Interfaces*. https://facebook.github.io/react/. 2017.

[Fer+11] A. Fernbach, W. Granzer, and W. Kastner. “Interoperability at the management level of building automation systems: A case study for BACnet and OPC UA”. In: *ETFA2011*. 2011, pp. 1–8. DOI: 10.1109/ETFA.2011.6059106.

[Fre16a] P. Fremantle. “A Reference Architecture for Internet of Things”. In: (2016). http://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/.

[Fre16b] P. Fremantle. “A Reference Architecture for Internet of Things”. In: (2016). http://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/.

[GS94] D. Garlan and M. Shaw. *An Introduction to Software Architecture*. Tech. rep. Pittsburgh, PA, USA, 1994.

[Gub+13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013). http://www.sciencedirect.com/science/article/pii/S0167739X13000241, pp. 1645 –1660. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2013.01.010.

[Gus+16] K. Gusmanov, K. Khanda, D. Salikhov, M. Mazzara, and N. Mavridis. "Jolie Good Buildings: Internet of things for smart building infrastructure supporting concurrent apps utilizing distributed microservices". In: *CoRR* abs/1611.08995 (2016). http://arxiv.org/abs/1611.08995.

[HAM16] D. HAMILTON. *The Four Internet of Things Connectivity Models Explained*. http://www.thewhir.com/web-hosting-news/the-four-internet-of-things-connectivity-models-explained. 2016 (accessed January 21, 2017).

[Hen16] M. Henshaw. "Systems of systems, cyber-physical systems, the internet-of-things… Whatever next?" In: (2016). DOI: 10.1002/inst.12109.

[hor+16] Y. A. horr, M. Arif, M. Katafygiotou, A. Mazroei, A. Kaushik, and E. Elsarrag. "Impact of indoor environmental quality on occupant well-being and comfort: A review of the literature". In: *International Journal of Sustainable Built Environment* 5.1 (2016), pp. 1 –11. ISSN: 2212-6090. DOI: http://dx.doi.org/10.1016/j.ijsbe.2016.03.006.

[Hua+12] L. Huang, Y. Zhu, Q. Ouyang, and B. Cao. "A study on the effects of thermal, luminous, and acoustic environments on indoor environmental comfort in offices". In: *Building and Environment* 49 (2012), pp. 304 –309. ISSN: 0360-1323. DOI: http://dx.doi.org/10.1016/j.buildenv.2011.07.022.

[HA97] S. HYGGE and H. ALLAN. *User Evaluation of Visual Comfort in Some Buildings of the Daylight Europe Project*. 1997.

[Wata] *IBM Getting started with Watson Analytics*. https://community.watsonanalytics.com/wp-content/uploads/2017/04/Tutorial-about-Watson-Analytics-2017-05-10.pdf?cm_mc_uid=09755305807814962356288&cm_mc_sid_50200000=1496342298&cm_mc_sid_52640000=1496342298. (accessed June 4, 2017).

[Inc16a] W. Inc. *WSO2 IoT Server*. http://wso2.com/products/iot-server/. 2016 (accessed January 23, 2017).

[Inc16b] W. Inc. *WSO2 IoT Server Documentation*. https://docs.wso2.com/display/IoTS100/WSO2+IoT+Server+Documentation. 2016 (accessed January 23, 2017).

[Inc16c] W. Inc. *WSO2 IoT Server Documentation Create a new Device Type*. https://docs.wso2.com/display/IoTS300/Creating+a+New+Device+Type. 2016 (accessed May 13, 2017).

[Inccea] W. Inc. *WSO2 Carbon*. http://wso2.com/products/carbon/. (accessed January 24, 2017).

[Incceb] W. Inc. *WSO2 Collecting Data*. https://docs.wso2.com/display/DAS300/Collecting+Data. (accessed June 1, 2017).

[Inccec] W. Inc. *WSO2 Communicating Results*. https://docs.wso2.com/display/DAS300/Communicating+Results. (accessed June 1, 2017).

[Incced] W. Inc. *WSO2 Event Streams*. https://docs.wso2.com/display/DAS300/Understanding+Event+Streams+and+Event+Tables. (accessed June 1, 2017).

[Inccee] W. Inc. *WSO2 Machine Learner*. https://docs.wso2.com/display/ML100/Introducing+Machine+Learner. (accessed June 1, 2017).

[Inccef] W. Inc. *WSO2 Data Analytics Server*. https://docs.wso2.com/display/DAS300/Introducing+DAS. (accessed May 31, 2017).

[Incceg] W. Inc. *WSO2 API Manager Documentation*. https://docs.wso2.com/display/AM210. (accessed September 1, 2017).

[Incceh] W. Inc. *WSO2 Enterprise Integrator Documentation*. https://docs.wso2.com/display/EI611. (accessed September 1, 2017).

[Iot] *IoT Standards and Protocols*. http://www.postscapes.com/internet-of-things-protocols/. 2017 (accessed January 21, 2017).

[Jai+99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data Clustering: A Review". In: *ACM Comput. Surv.* 31.3 (1999), pp. 264–323. ISSN: 0360-0300. DOI: 10.1145/331499.331504.

[Jun+12] M. Jung, J. Weidinger, C. Reinisch, W. Kastner, C. Crettaz, A. Olivieri, and Y. Bocchi. "A transparent ipv6 multi-protocol gateway to integrate building automation systems in the internet of things". In: *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE. 2012, pp. 225–233.

[Ker+16] G. Keramidas, N. Voros, and M. Hbner. *Components and Services for IoT Platforms: Paving the Way for IoT Standards*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319423029, 9783319423029.

[KM15] S. K. Khaitan and J. D. McCalley. "Design Techniques and Applications of Cyberphysical Systems: A Survey". In: *IEEE Systems Journal* 9.2 (2015), pp. 350–365. ISSN: 1932-8184.

[Kha+12] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. “Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges”. In: *2012 10th International Conference on Frontiers of Information Technology*. 2012, pp. 257–260. DOI: 10.1109/FIT.2012.53.

[Kop11] H. Kopetz. “Internet of Things”. In: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. http://dx.doi.org/10.1007/978-1-4419-8237-7_13. Boston, MA: Springer US, 2011, pp. 307–323. ISBN: 978-1-4419-8237-7. DOI: 10.1007/978-1-4419-8237-7_13.

[Kru95] P. B. Kruchten. “The 4+1 View Model of architecture”. In: *IEEE Software* 12.6 (1995), pp. 42–50. ISSN: 0740-7459. DOI: 10.1109/52.469759.

[LS13] M. A. Laughton and M. G. Say. *Electrical engineer’s reference book*. Elsevier, 2013. ISBN: 978-0-7506-4637-6.

[Law08] G. Lawton. “Developing Software Online With Platform-as-a-Service Technology”. In: *Computer* 41.6 (2008), pp. 13–15. ISSN: 0018-9162. DOI: 10.1109/MC.2008.185.

[Lea+14] S. Leal, G. Zucker, S. Hauer, and F. Judex. “A Software Architecture for Simulation Support in Building Automation”. In: *Buildings* 4.3 (2014). http://www.mdpi.com/2075-5309/4/3/320, pp. 320–335. ISSN: 2075-5309. DOI: 10.3390/buildings4030320.

[Lee08] E. A. Lee. “Cyber Physical Systems: Design Challenges”. In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[LS15] E. A. Lee and S. A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2015. ISBN: 978-0-262-53381-2.

[Lee+10] I. Lee, L. Sha, and J. Stankovic. “Cyber-physical systems: The next computing revolution”. In: *Adream, LAAS-CNSR-2010* (2010).

[Li+15] S. Li, L. D. Xu, and S. Zhao. “The internet of things: a survey”. In: *Information Systems Frontiers* 17.2 (2015). http://dx.doi.org/10.1007/s10796-014-9492-7, pp. 243–259. ISSN: 1572-9419. DOI: 10.1007/s10796-014-9492-7.

[LL17] J. Lienhard IV and J. Lienhard V. *A Heat Transfer Textbook*. 4th. Cambridge, MA: Phlogiston Press, 2017.

[Lif] *LifX - Light Bulb Specifications*. https://www.lifx.com/pages/color-1000-info-sheet?geo=false. (accessed September 10, 2017).

[Lil+17] G. Lilis, G. Conus, N. Asadi, and M. Kayal. “Towards the next generation of intelligent building: An assessment study of current automation and future IoT based systems with a proposal for transitional design”. In: *Sustainable Cities and Society* 28 (2017). http://www.sciencedirect.com/science/article/pii/S2210670716302414, pp. 473 –481. ISSN: 2210-6707. DOI: \url{http://dx.doi.org/10.1016/j.scs.2016.08.019}.

[Mar+17] G. Marques, N. Garcia, and N. Pombo. “A Survey on IoT: Architectures, Elements, Applications, QoS, Platforms and Security Concepts”. In: *Advances in Mobile Cloud Computing and Big Data in the 5G Era*. Ed. by C. X. Mavromoustakis, G. Mastorakis, and C. Dobre. http://dx.doi.org/10.1007/978-3-319-45145-9_5. Springer International Publishing, 2017, pp. 115–130. ISBN: 978-3-319-45145-9. DOI: 10.1007/978-3-319-45145-9_5.

[Mav] *Maven Archetype Documentation*. https://maven.apache.org/guides/introduction/introduction-to-archetypes.html. 2016 (accessed May 13, 2017).

[Maxa] L. E. Maxwell. *Facility Planning and Management - Noise in the Office Workplace*. http://dea.human.cornell.edu/sites/default/files/pdf/fpm-notes_vol1_number1s.pdf.

[Maxb] L. E. Maxwell. *Node-Red*. https://nodered.org.

[Mer+09] H. Merz, T. Hansemann, and C. Hbner. *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 3540888284, 9783540888284.

[Min13] D. Minoli. *Building the internet of things with IPv6 and MIPv6: The evolving world of M2M communications*. John Wiley & Sons, 2013.

[Moh+11] R. Mohamaddoust, A. Toroghi Haghighat, m. j. Motahari, and N. Capanni. “A Novel Design of an Automatic Lighting Control System for a Wireless Sensor Network with Increased Sensor Lifetime and Reduced Sensor Numbers”. In: 11 (Dec. 2011), pp. 8933–52.

[Moh+16] N. Mohamed, S. Lazarova-Molnar, and J. Al-Jaroodi. “CE-BEMS: A cloud-enabled building energy management system”. In: *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*. 2016, pp. 1–6. DOI: 10.1109/ICBDSC.2016.7460393.

[MP] T. T. Mulani and S. V. Pingle. “Internet of things”. In: *International Research Journal of Multidisciplinary Studies* 2.3 ().

[Mun+13] S. Munir, J. A. Stankovic, C.-J. M. Liang, and S. Lin. “Cyber Physical System Challenges for Human-in-the-Loop Control”. In: *Presented as part of the 8th International Workshop on Feedback Computing*. https://www.usenix.org/conference/feedbackcomputing13/workshop-program/presentation/Munir. San Jose, CA: USENIX, 2013.

[NC15] B. Nakhuva and T. Champaneria. “Study of Various Internet of Things Platforms”. In: 6 (Dec. 2015), pp. 61–74.

[Ngu+17] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. “IoT Middleware: A Survey on Issues and Enabling Technologies”. In: *IEEE Internet of Things Journal* 4.1 (2017), pp. 1–20. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2615180.

[Ope] *openHab Textual Rules*. http://docs.openhab.org/configuration/rules-dsl.html.

[Per+14] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. “Context Aware Computing for The Internet of Things: A Survey”. In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 414–454. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.042313.00197.

[Pra+16] S Pradeep, T Kousalya, K. A. Suresh, and J. Edwin. “IoT AND ITS CONNECTIVITY CHALLENGES IN SMART HOME”. In: (2016).

[Pur+13] S. Purdon, B. Kusy, R. Jurdak, and G. Challen. “Model-free HVAC control using occupant feedback”. In: *38th Annual IEEE Conference on Local Computer Networks - Workshops*. 2013, pp. 84–92. DOI: 10.1109/LCNW.2013.6758502.

[Raj+10] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. “Cyber-physical systems: The next computing revolution”. In: *Design Automation Conference*. 2010, pp. 731–736. DOI: 10.1145/1837274.1837461.

[Ras+03] A. Rashid, A. Moreira, and J. Araújo. “Modularisation and Composition of Aspectual Requirements”. In: *Proceedings of the 2Nd International Conference on Aspect-oriented Software Development*. AOSD ’03. http://doi.acm.org/10.1145/643603.643605. New York, NY, USA: ACM, 2003, pp. 11–20. ISBN: 1-58113-660-9. DOI: 10.1145/643603.643605.

[Raw+15] D. B. Rawat, J. J.P. C. Rodrigues, and I. Stojmenovic. *Cyber-Physical Systems: From Theory to Practice*. Boca Raton, FL, USA: CRC Press, Inc., 2015. ISBN: 1482263327, 9781482263329.

[Ray16] P. Ray. “A survey on Internet of Things architectures”. In: *Journal of King Saud University - Computer and Information Sciences* (2016). ISSN: 1319-1578. DOI: http://dx.doi.org/10.1016/j.jksuci.2016.10.003.

[Ree+15] K. E. M. Reena, A. T. Mathew, and L. Jacob. "An Occupancy Based Cyber-Physical System Design for Intelligent Building Automation". In: *Mathematical Problems in Engineering* 501 (2015), p. 132182. DOI: 10.1155/2015/132182.

[Sen14] S. K. Sen. *Fieldbus and Networking in Process Automation*. Boca Raton, FL, USA: CRC Press, Inc., 2014. ISBN: 1466586761, 9781466586765.

[SB15] D. M. Simmonds and A. Bhattacherjee. "Smart Systems, Smarter Living: An Empirical Study of the Building Automation System in Organizations". In: (2015).

[Soc+14] I. C. Society, P. Bourque, and R. E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014. ISBN: 0769551661, 9780769551661.

[Som10] I. Sommerville. *Software Engineering*. 9th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0137035152, 9780137035151.

[Tat16] R. Tatum. *Where Does the Building Automation System Fit In An Internet of Things World?* http://www.facilitiesnet.com/buildingautomation/article/Where-Does-the-Building-Automation-System-Fit-In-An-Internet-of-Things-World-Facilities-Management-Building-Automation-Feature--16651. 2016.

[Thi15] I. B. Thingom. "Internet of Things: Design of aNew Layered Architecture and Studyof SomeExisting Issues". In: (2015).

[Thi14] ThingSpeak. *Introduction to the "Internet of Things" and ThingSpeak. ThingSpeak Community*. http://community.thingspeak.com/. 2014.

[Wan+17] W. Wang, K. Lee, and D. Murray. "A global generic architecture for the future Internet of Things". In: *Service Oriented Computing and Applications* 11.3 (2017), pp. 329–344. ISSN: 1863-2394. DOI: 10.1007/s11761-017-0213-1.

[Wan+16] Y. Wang, A. Khelil, A. Broering, and D. Anicic. "Big IoT". In: *Analysis of Technology Readiness*. 2016.

[Watb] *Watson IoT*. https://www.ibm.com/internet-of-things/platform/watson-iot-platform/. (accessed February 4, 2017).

[Watc] *Watson IoT APIs*. https://console.ng.bluemix.net/docs/services/IoT/reference/api.html. (accessed May 24, 2017).

[Watd] *Watson IoT Architecture*. https://www.ibm.com/devops/method/content/architecture/iotArchitecture. (accessed February 4, 2017).

[Wate] *Watson IoT Devices*. https://developer.ibm.com/recipes/tutorials/how-to-register-devices-in-ibm-iot-foundation. (accessed May 24, 2017).

[WF+15] F. Wortmann, K. Flüchter, et al. “Internet of things”. In: *Business & Information Systems Engineering* 57.3 (2015), pp. 221–224.

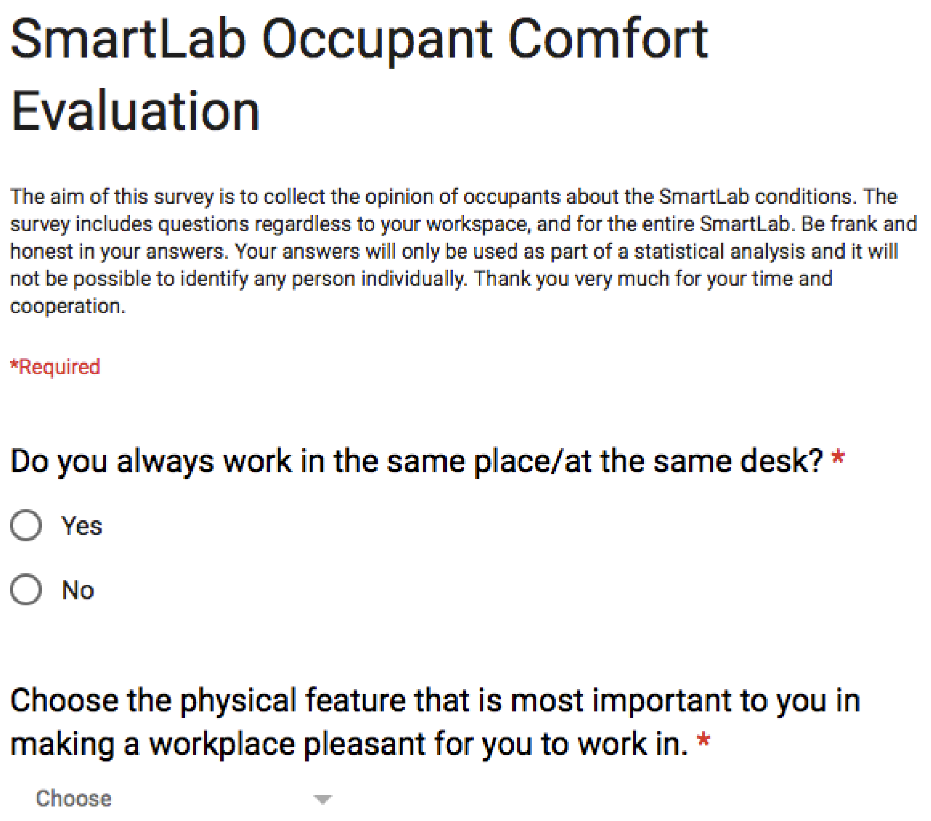
[Xia+12] F. Xia, L. T. Yang, L. Wang, and A. Vinel. “Internet of Things”. In: *International Journal of Communication Systems* 25.9 (2012), pp. 1101–1102. ISSN: 1099-1131. DOI: 10.1002/dac.2417.

[Yan+14] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky. “Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing”. In: *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2014, pp. 325–329. DOI: 10.1109/CAMAD.2014.7033259.

[Zha+16] P. Zhao, T. Peffer, R. Narayanamurthy, G. Fierro, P. Raftery, S. Kaam, and J. Kim. “Getting into the zone: how the internet of things can improve energy efficiency and demand response in a commercial building”. In: (2016).

Appendix A

Questionnaire



SmartLab Occupant Comfort Evaluation

The aim of this survey is to collect the opinion of occupants about the SmartLab conditions. The survey includes questions regardless to your workspace, and for the entire SmartLab. Be frank and honest in your answers. Your answers will only be used as part of a statistical analysis and it will not be possible to identify any person individually. Thank you very much for your time and cooperation.

*Required

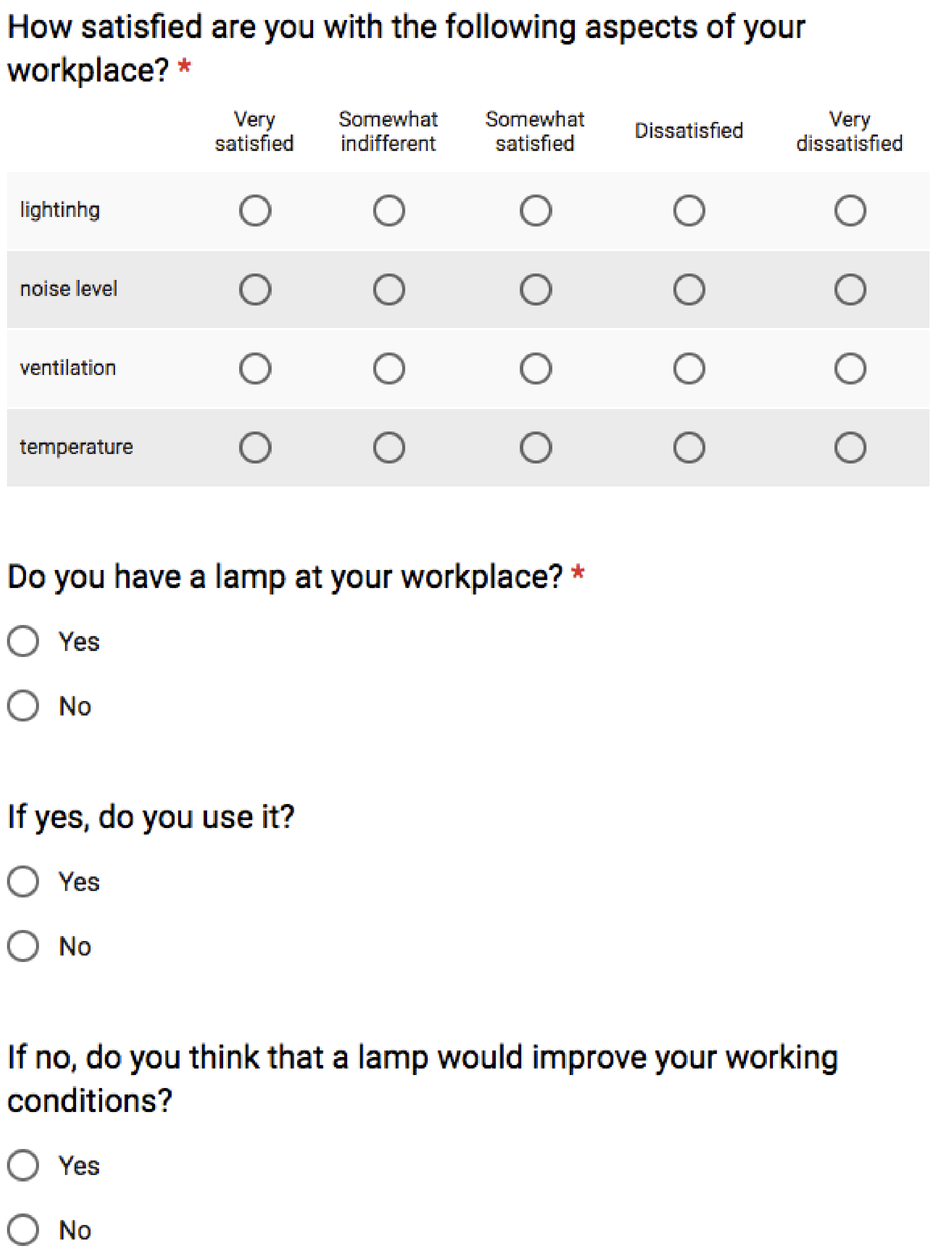
Do you always work in the same place/at the same desk? *

○ Yes

○ No

Choose the physical feature that is most important to you in making a workplace pleasant for you to work in. *

Choose ▾



How satisfied are you with the following aspects of your workplace? *

	Very satisfied	Somewhat indifferent	Somewhat satisfied	Dissatisfied	Very dissatisfied
lightinhg	○	○	○	○	○
noise level	○	○	○	○	○
ventilation	○	○	○	○	○
temperature	○	○	○	○	○

Do you have a lamp at your workplace? *

○ Yes

○ No

If yes, do you use it?

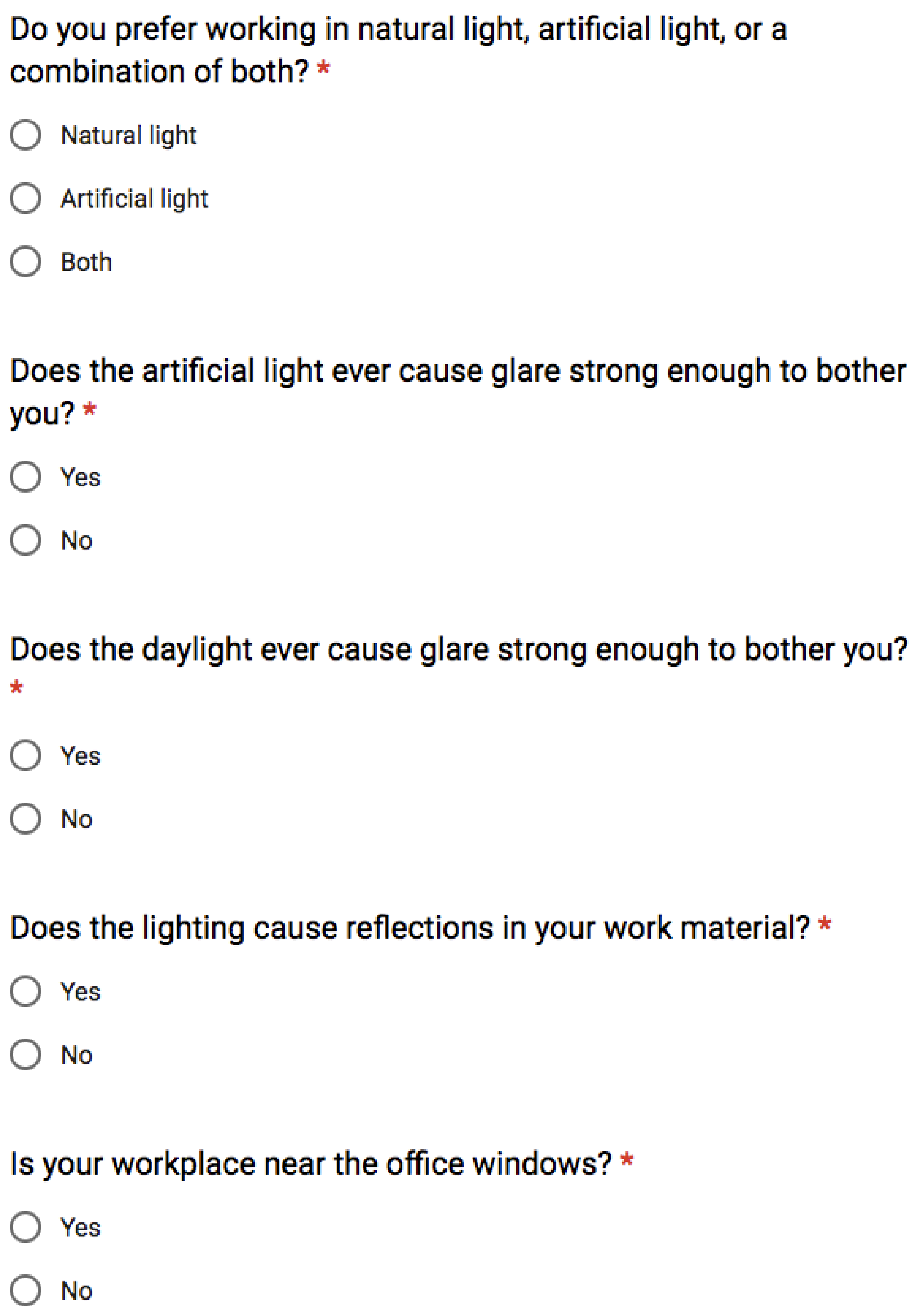
○ Yes

○ No

If no, do you think that a lamp would improve your working conditions?

○ Yes

○ No



Do you prefer working in natural light, artificial light, or a combination of both? *

○ Natural light

○ Artificial light

○ Both

Does the artificial light ever cause glare strong enough to bother you? *

○ Yes

○ No

Does the daylight ever cause glare strong enough to bother you? *

○ Yes

○ No

Does the lighting cause reflections in your work material? *

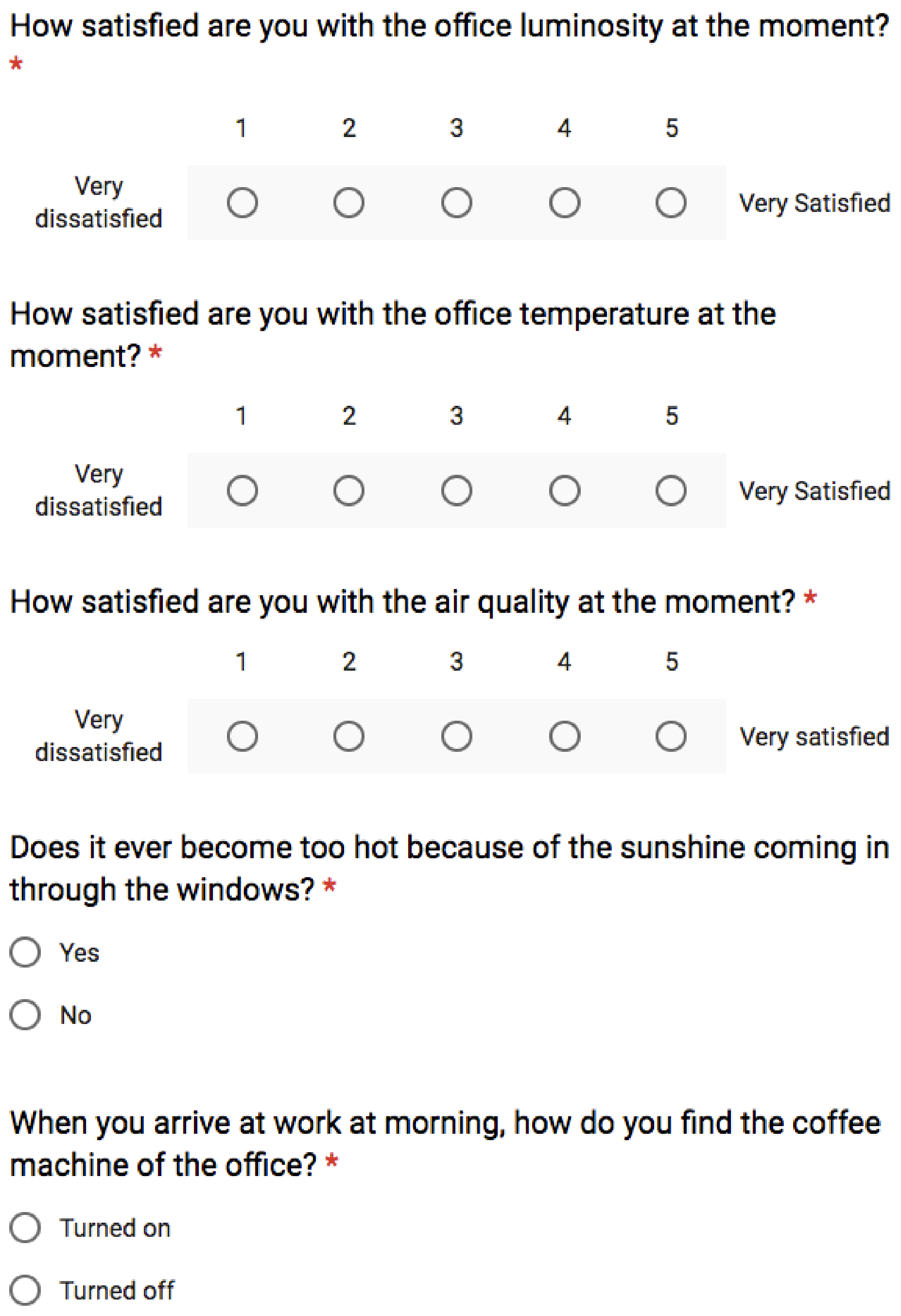
○ Yes

○ No

Is your workplace near the office windows? *

○ Yes

○ No



How satisfied are you with the office luminosity at the moment? *

	1	2	3	4	5	
Very dissatisfied	○	○	○	○	○	Very Satisfied

How satisfied are you with the office temperature at the moment? *

	1	2	3	4	5	
Very dissatisfied	○	○	○	○	○	Very Satisfied

How satisfied are you with the air quality at the moment? *

	1	2	3	4	5	
Very dissatisfied	○	○	○	○	○	Very satisfied

Does it ever become too hot because of the sunshine coming in through the windows? *

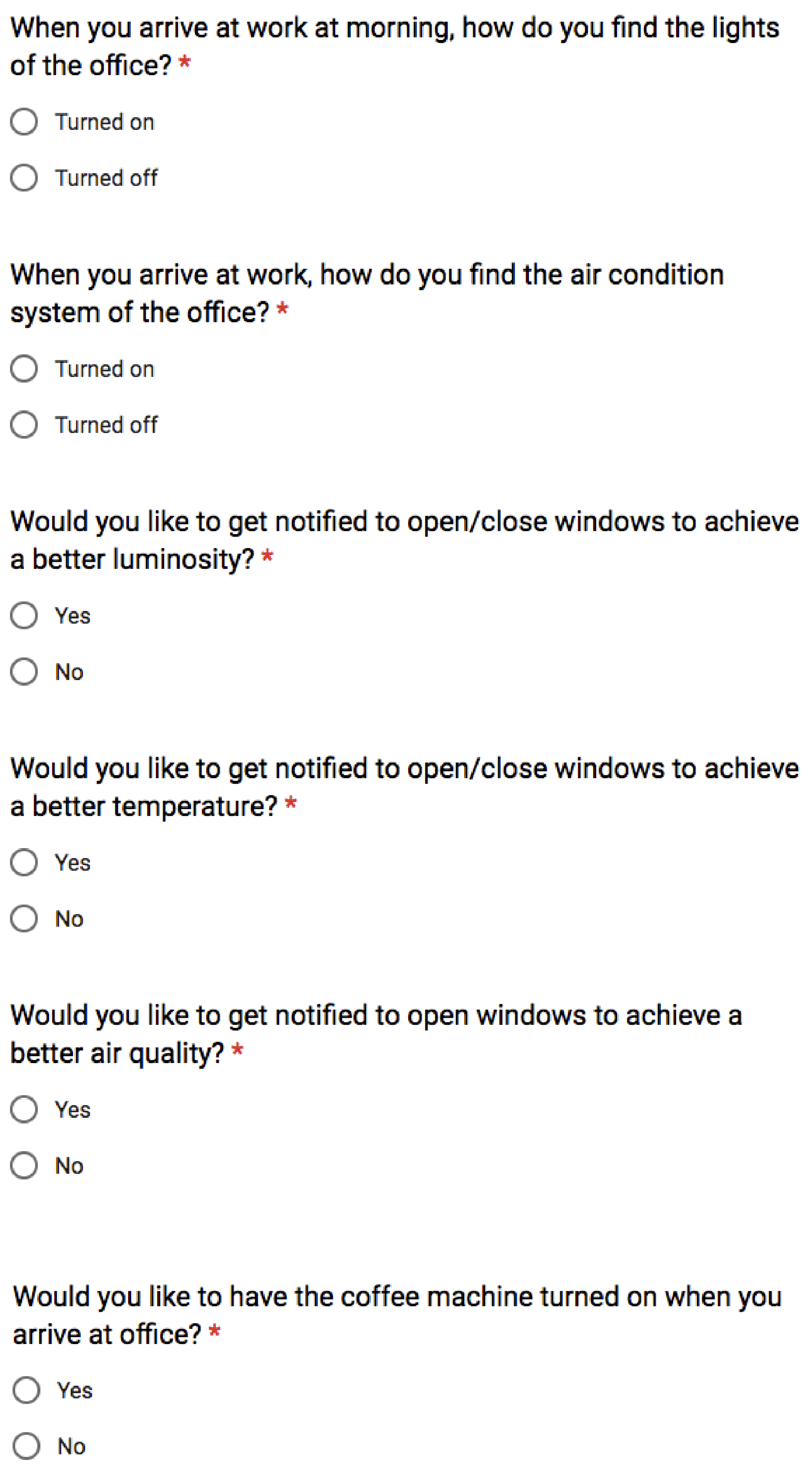
○ Yes

○ No

When you arrive at work at morning, how do you find the coffee machine of the office? *

○ Turned on

○ Turned off



When you arrive at work at morning, how do you find the lights of the office? *

○ Turned on

○ Turned off

When you arrive at work, how do you find the air condition system of the office? *

○ Turned on

○ Turned off

Would you like to get notified to open/close windows to achieve a better luminosity? *

○ Yes

○ No

Would you like to get notified to open/close windows to achieve a better temperature? *

○ Yes

○ No

Would you like to get notified to open windows to achieve a better air quality? *

○ Yes

○ No

Would you like to have the coffee machine turned on when you arrive at office? *

○ Yes

○ No

Appendix B

Requirements Tracing

Table B.1: User Management Mapping

User Story Identifier	**Functional Requirements**
US-UM1	FR-UM6
US-UM2	FR-UM2
US-UM3	FR-UM4, FR-UM5
US-UM4	FR-UM3
US-UM5	FR-UM7, FR-UM8
US-UM6	FR-UM1

Table B.2: API Management Mapping

User Story Identifier	**Functional Requirements**
US-AM1	FR-AM1, FR-AM2
US-AM2	FR-AM6
US-AM3	FR-AM7
US-AM4	FR-AM3, FR-AM4, FR-AM5,
US-AM5	FR-AM8
US-AM6	FR-AM9
US-AM7	FR-AM10

Table B.3: Device Management Mapping

User Story Identifier	Functional Requirements
US-DM1	FR-DM3
US-DM2	FR-DM4
US-DM3	FR-DM8
US-DM4	FR-DM1
US-DM5	FR-DM11
US-DM6	FR-DM2
US-DM7	FR-DM9
US-DM8	FR-DM5, FR-DM6
US-DM9	FR-DM7
US-DM10	FR-DM13, FR-DM14
US-DM11	FR-DM10
US-DM12	FR-DM15
US-DM13	FR-DM12

Table B.4: Occupant Comfort Mapping

User Story Identifier	Functional Requirements
US-OC1	FR-OC1, FR-OC2
US-OC2	FR-OC4
US-OC3	FR-OC3

Table B.5: Presence Detection Mapping

User Story Identifier	Functional Requirements
US-PD1	FR-PD5
US-PD2	FR-PD6
US-PD3	FR-PD7, FR-PD8
US-PD4	FR-PD10
US-PD5	FR-PD9
US-PD6	FR-PD11
US-PD7	FR-PD1, FR-PD2, FR-PD3, FR-PD4

Table B.6: Scheduling Mapping

User Story Identifier	Functional Requirements
US-SC1	FR-AQ1
US-SC2	FR-SC1
US-SC3	FR-SC2
US-SC4	FR-SC3, FR-AQ2
US-SC5	FR-SC4
US-SC6	FR-SC5

Table B.7: Suggestions Mapping

User Story Identifier	Functional Requirements
US-SG1	FR-SG3
US-SG2	FR-SG2
US-SG3	FR-SG1

Table B.8: Notifications Mapping

User Story Identifier	Functional Requirements
US-NT1	FR-NT5
US-NT2	FR-NT4
US-NT3	FR-NT1, FR-NT2, FR-NT3
US-NT4	FR-NT6, FR-NT7, FR-NT8
US-NT5	FR-NT9

Table B.9: Aquarium Mapping

User Story Identifier	Functional Requirements
US-AQ1	FR-AQ3
US-AQ2	FR-AQ4
US-AQ3	FR-AQ6
US-AQ4	FR-AQ5

Appendix C

Front-end User Interface

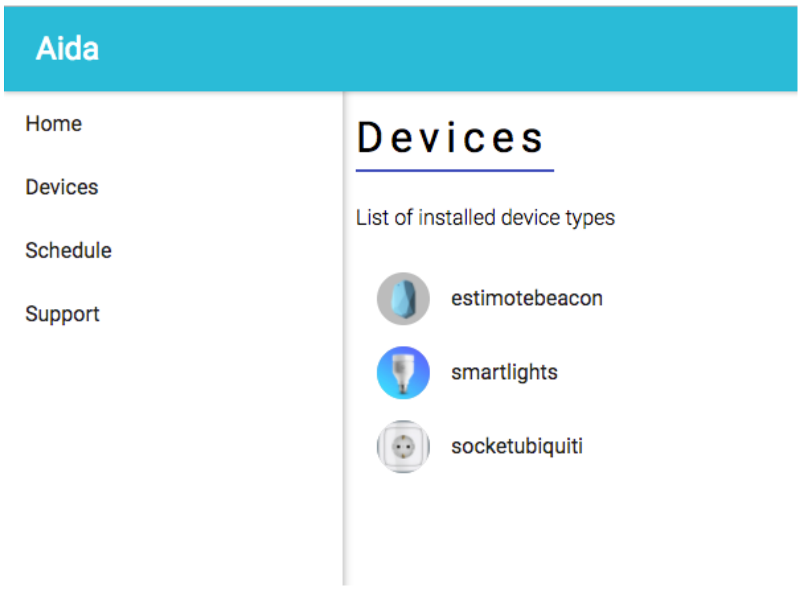
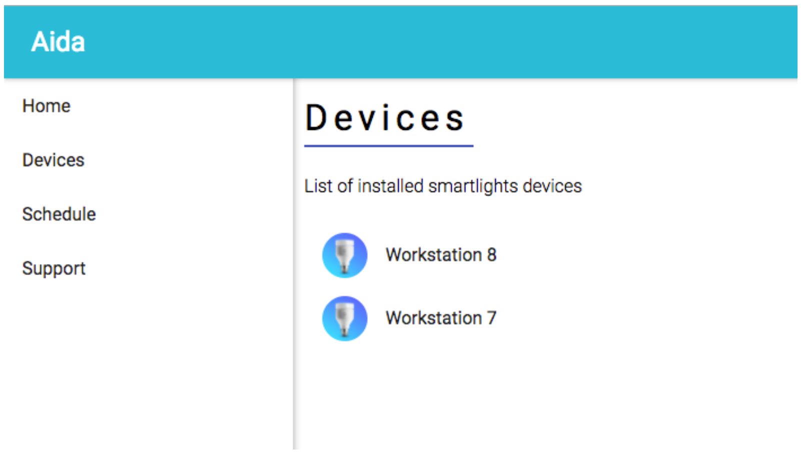


Figure C.1: Front-end User Interface - Desktop view for devices types list



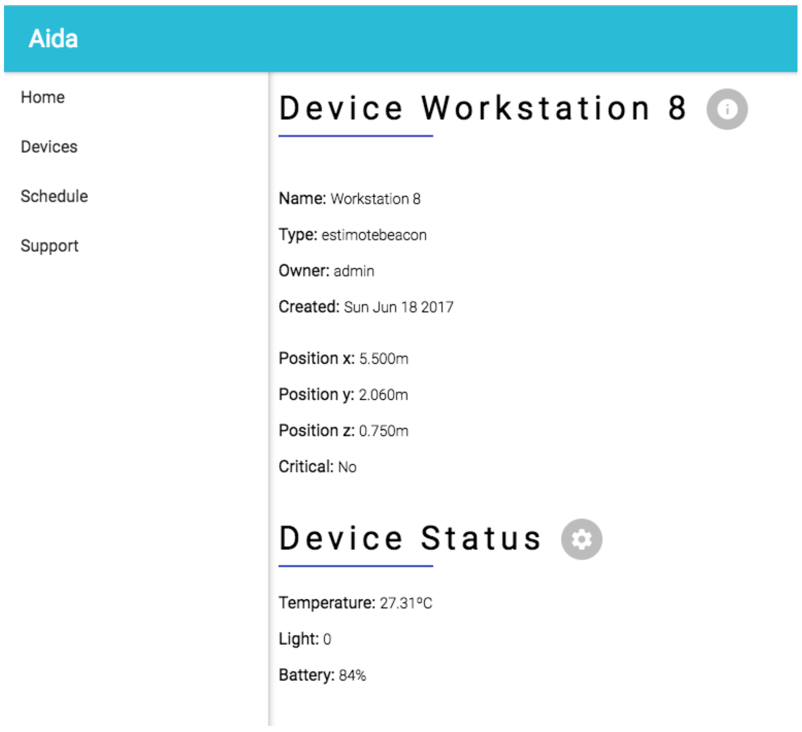
Devices

List of installed smartlights devices

Workstation 8

Workstation 7

Figure C.2: Front-end User Interface - Desktop view for devices list



Device Workstation 8

Name: Workstation 8

Type: estimotebeacon

Owner: admin

Created: Sun Jun 18 2017

Position x: 5.500m

Position y: 2.060m

Position z: 0.750m

Critical: No

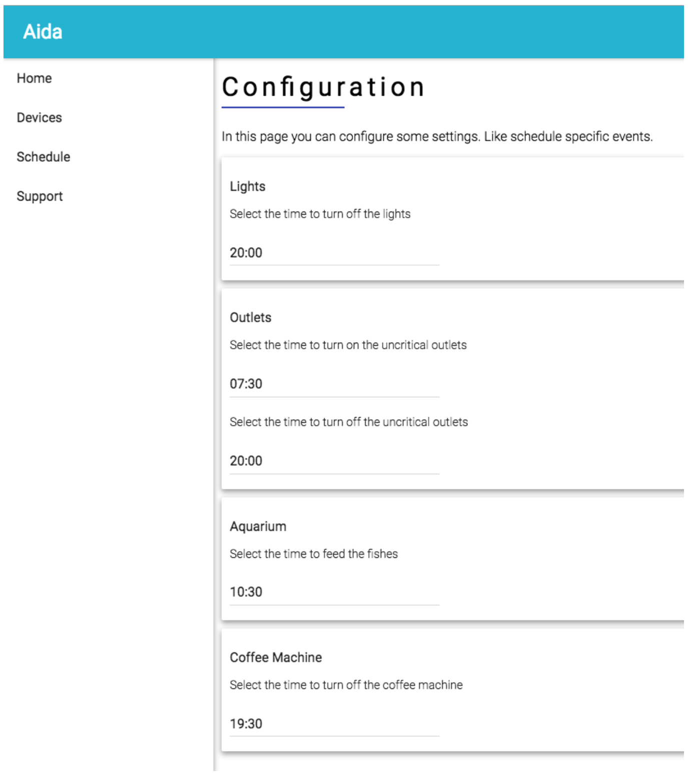
Device Status

Temperature: 27.31°C

Light: 0

Battery: 84%

Figure C.3: Front-end User Interface - Desktop view for device details and control view


Aida

Home
Devices
Schedule
Support

Configuration

In this page you can configure some settings. Like schedule specific events.

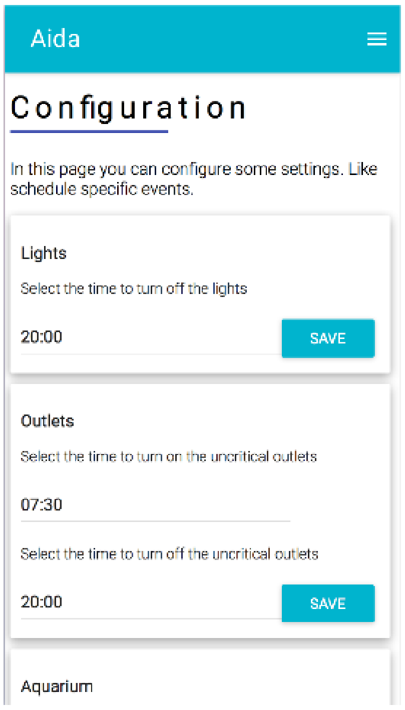
Lights
Select the time to turn off the lights
20:00

Outlets
Select the time to turn on the uncritical outlets
07:30
Select the time to turn off the uncritical outlets
20:00

Aquarium
Select the time to feed the fishes
10:30

Coffee Machine
Select the time to turn off the coffee machine
19:30

Figure C.4: Front-end User Interface - Desktop view for rules view


Aida

Configuration

In this page you can configure some settings. Like schedule specific events.

Lights
Select the time to turn off the lights
20:00 SAVE

Outlets
Select the time to turn on the uncritical outlets
07:30
Select the time to turn off the uncritical outlets
20:00 SAVE

Aquarium

Figure C.5: Front-end User Interface - Mobile view for rules view

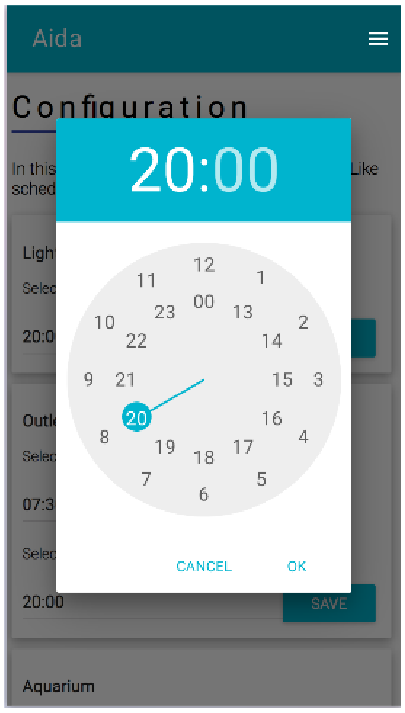
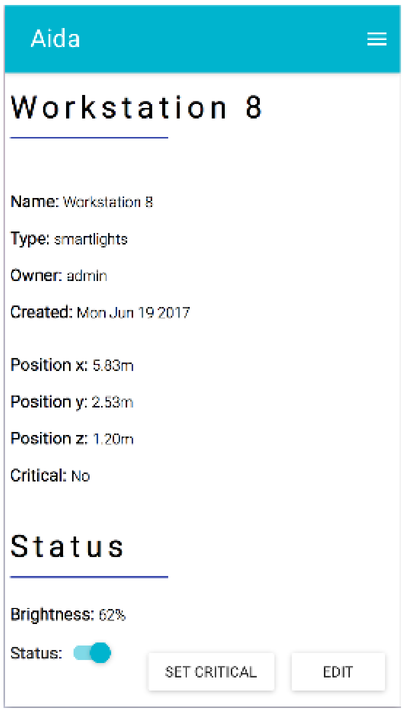


Figure C.6: Front-end User Interface - Mobile view for rule configuration view



Workstation 8

Name: Workstation 8

Type: smartlights

Owner: admin

Created: Mon Jun 19 2017

Position x: 5.83m

Position y: 2.53m

Position z: 1.20m

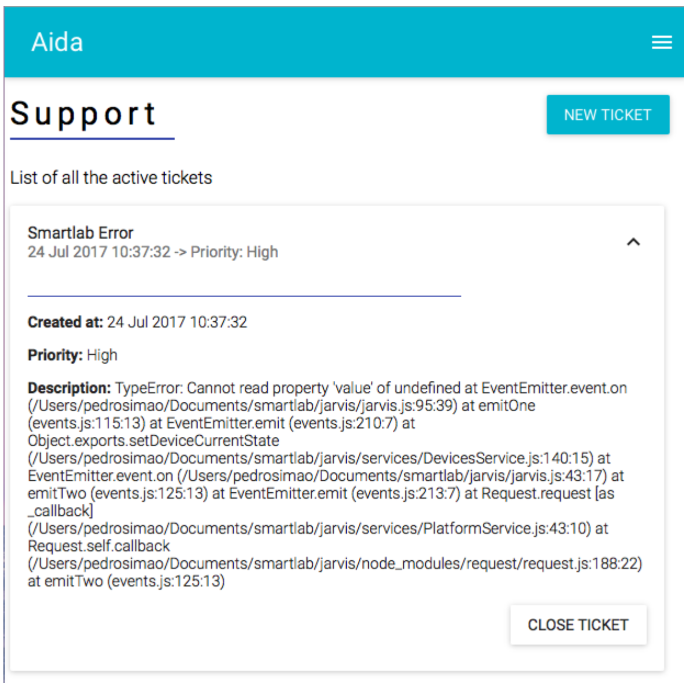
Critical: No

Status

Brightness: 62%

Status:

Figure C.7: Front-end User Interface - Mobile view for device details and control view



List of all the active tickets

Smartlab Error
24 Jul 2017 10:37:32 -> Priority: High

Created at: 24 Jul 2017 10:37:32

Priority: High

Description: TypeError: Cannot read property 'value' of undefined at EventEmitter.event.on (/Users/pedrosimao/Documents/smartlab/jarvis/jarvis.js:95:39) at emitOne (events.js:115:13) at EventEmitter.emit (events.js:210:7) at Object.exports.setDeviceCurrentState (/Users/pedrosimao/Documents/smartlab/jarvis/services/DevicesService.js:140:15) at EventEmitter.event.on (/Users/pedrosimao/Documents/smartlab/jarvis/jarvis.js:43:17) at emitTwo (events.js:125:13) at EventEmitter.emit (events.js:213:7) at Request.request [as _callback] (/Users/pedrosimao/Documents/smartlab/jarvis/services/PlatformService.js:43:10) at Request.self.callback (/Users/pedrosimao/Documents/smartlab/jarvis/node_modules/request/request.js:188:22) at emitTwo (events.js:125:13)

Figure C.8: Front-end User Interface - Desktop view for ticket support