Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2017

Bruno Miguel Monteiro Silva

Criação de um Sistema de Gestão de Gateways para Casas Inteligentes

Creating a Management System for Smart Home Gateways

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Bruno Miguel Monteiro Silva

Criação de um Sistema de Gestão de Gateways para Casas Inteligentes

Creating a Management System for Smart Home Gateways

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Rui Aguiar, Professor catedrático c/agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho ao meu avô pois sei que ele iria adorar estar presente neste momento da minha vida.

o júri / the jury

presidente / president

Prof. Doutor Amaro Fernandes de Sousa
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Diogo Nuno Pereira Gomes
Professor Auxiliar da Universidade de Aveiro

Doutor David Emmanuel Marques Campos
Team Leader na Bosch Termotecnologia, S.A.

agradecimentos /
acknowledgements

Gostaria inicialmente de agradecer ao Prof. Doutor Diogo Gomes (Orientador) pelo apoio e acompanhamento dado ao longo da elaboração desta dissertação.
Agradeço ainda aos meus pais, que permitiram que tudo isto fosse possível por me darem a oportunidade de frequentar o ensino superior, aos meus amigos que ao longo deste tempo todo sempre me apoiaram e motivaram.
Finalmente, gostaria de agradecer à minha namorada por todo o apoio, não só durante o tempo da dissertação mas em todo o percurso académico pelo apoio e motivação que me deu e por sempre acreditar em mim, sem ela tudo teria sido mais dificíl.

Palavras Chave

Automação, Casa Inteligente, Ambiente Inteligente, Internet das Coisas, Gateway, Gestão de Dispositivos, Plataformas Cloud.

Resumo

Esta dissertação enquadra-se no projeto Smart Green Home, que resulta de uma parceria entre a Bosch e a Universidade de Aveiro, e visa criar um dispositivo (Gateway) capaz de interagir com um ambiente domestico inteligente e multi-tecnologias de forma a facilitar a sua integração em instalações existentes e ser o mais possível independente de marcas. Mais ainda, pretende-se criar um sistema de gestão de Gateways em produção por forma a tornar tarefas de manutenção simples e escaláveis. Relativamente a este sistema, ele trás ainda valor acrescentado para o utilizador permitindo o acesso remoto à sua casa e monitorizar e controlar os seus dispositivos, assim como outras capacidades. Para satisfazer os requisitos deste sistema, foi desenvolvida uma solução para ambas as componentes do sistema, a Gateway e o sistema de gestão. Esta solução foi implementada com sucesso e o seu funcionamento validado de acordo com os requisitos. Por último, uma avaliação à solução final implementada, com levantamento das suas limitações, foi realizada e são expostos possíveis futuros melhoramentos para o sistema.

Keywords

Automation, Smart Home, Smart Environment, Internet of Things, Web Platform, Gateway, Device Management, Cloud Platforms.

Abstract

This dissertation was done in the scope of the Smart Green Home project, that was born from a partnership between Bosch and the University of Aveiro, and strives to achieve a device (Gateway) capable of interacting with a smart home environment where multi-technologies are present allowing it to more easily integrate in existing installations along with being vendor independent. Additionally, it will be created a gateway management system to allow their maintenances to become effortless and scalable. Regarding this system, it also provides added value to the user by allowing remote access his home to monitor and control his devices, as well as other features. In order to fulfill this system requirements, a solution was developed for both of the system's components, the Gateway and the management system. This solution was then successfully implemented and its functionality validated according to its requirements. Lastly, an evaluation to the final implemented solution was conducted, and its limitations gathered, exposing this way possible future improvements.

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AB	ApacheBench
AES	Advanced Encryption Standard
AFH	Automatic Frequency Hopping
AP	Access Point
API	Application Programming Interface
AMQP	Advanced Message Queue Protocol
BLE	Bluetooth Low Energy
BSP	Board Support Package
CA	Certificate Authority
CLI	Command Line Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSS	Cascading Style Sheets
DAD	Duplicate Address Detection
DB	Database
DHCP	Dynamic Host Control Protocol
DNS	Domain Name Server
DTLS	Datagram Transport Layer Security
EHS	European Home Systems
EIB	European Installation Bus
EPL	Eclipse Public License
ETS	Engineering tool software
EUI	Extended Unique Identifier
FFD	Fully Functional Device
GAP	Generic Access Protocol
GATT	Generic Attribute Protocol
GPL	General Public License
GUI	Graphical User Interface
HA	High Availability
HAS	Home Automation System
HGW	Home Gateway
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
HVAC	Heating, Ventilation and Air Conditioning
IAM	Identity and Access Management
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IM	Instant Messaging
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IT	Instituto de Telecomunicações
JS	JavaScript
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
M2M	Machine-to-Machine
MQTT	Message Queue Telemetry Transport
MQTT-SN	Message Queue Telemetry Transport For Sensor Networks
MTU	Maximum Transmission Unit
OEMs	Original Equipment Manufacturers

OS	Operating System
OSI	Open Systems Interconnection
OTA	Over the Air
OTC	One Time Code
PAN	Personal Area Network
PKI	Public Key Infrastructure
PLC	PowerLine Communication
QoS	Quality of Service
REST	Representational State Transfer
RF	Radio Frequency
RFD	Reduced Functional Device
RSA	Rivest-Shamir-Adleman
SASL	Simple Authentication and Security Layer
SGH	Smart Green Home
SHA-1	Secure Hash Algorithm 1
SIG	Special Interest Group
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TIM	Traffic Indication Map
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
UID	Unique Identifier
URL	Unique Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YAML	YAML Ain't Markup Language

CHAPTER 1

Introduction

Nowadays technology is surrounding us in every aspect of our life, it is undeniable that in the last years the amount of technology we use everyday has been growing. From small gadgets and wearables to entire buildings and cars filled with technology it is all becoming part of our daily life.

One thing in common with all these devices is all of them consume energy, and the footprint left by them is very noticeable. An analysis of the usage of energy in 2014 [1], showed that the household sector is responsible for almost 25% of the total energy spent in the entire Europe. Even though renewable energies are striving to fight this battle, there is still a long way before it being cost-effective for mass deployment.

In today's market the concern for energy saving is already starting to be present in the consumers mind, however there is still a lot of energy being wasted due to human error, like leaving a light bulb on during the night due to forgetfulness.

Even though consumers most likely are not willing to make huge investments for small energy savings, additional motivation could be present by increasing the comfort in their homes while reducing the human labor. It is targeting this vision that the concept of Home Automation System (HAS) was created. The concept of HAS requires for most of the systems inside a home to be connected, this way allowing the automation of house tasks and the intelligent management of power systems. Currently, these solutions are vendor dependent which means that interoperability between brands is not possible. Because of this, these systems are only available on very expensive and high-end homes, making them available only to a very restricted population group[2]. It is clear that for these systems to reach the majority of the homes cheaper and more flexible solutions need to arise.

In the last decade a new concept, by the name of IoT, has emerged and according to [3], IoT is the point when more "things or objects" are connected to the Internet than people. Still according to [3] this point was reached sometime between 2008 and 2009 meaning, nowadays IoT is well under way and growing. This new concept brought new opportunities for HAS, both in the reduction of costs as well as interoperability between different brands.

As a result of the growth of the number of connected devices, the data resulting from them will increase exponentially. To handle all this data, new platforms and infrastructure will be required in order to allow the processing necessary for turning the data into information so knowledge can be extracted from it.

Equally important is the transport of the data generated from the IoT devices to the connected platforms and vice versa. Based on the information in [4], the authors state that there are two common approaches for managing devices in home networks: direct and indirect management. The direct approach does not require a Home Gateway (HGW), but imposes some constraints regarding the capabilities of the device. In contrast, the indirect approach makes use of a HGW in order to manage the devices locally and only then is the data sent to the connected platforms.

It is within these visions that this dissertation fits itself, by proposing a HGW along with a cloud platform solution that aims to combine the principles of low cost and low power of IoT to achieve not only a better energy efficient home, and thus reducing its environmental footprint, but also one that can provide their occupants with increased comfort and situational awareness.

1.1 MOTIVATION

The main motivation for this dissertation is divided into 2 segments. The first one being the development of a solution that allows an increase in the comfort of the habitants of homes with an HAS. As for the second one and, knowing the environmental impact of energy production [5][6], and that almost one quarter of it is consumed in the household market[1], the need to reduce its footprint is a motivation factor by itself. In addiction, extra motivation comes from the deployment of the concept of IoT, one of the most important research and investment topics for both the industrial and academic world at the present.

This dissertation aims to achieve a solution that includes all of these motivation factors.

1.2 OBJECTIVES

The key objective of this work is to propose a solution for the creation of a HGW that is capable of interacting and connecting all the active systems in a house (e.g. Power Sources, Water Systems, Lighting, Appliances, Security Systems) regardless of the manufacturer or protocol used. This gateway must be able to be managed remotely, for example automatically install security updates through a central cloud platform.

Moreover, this work also aims to develop or integrate software to achieve a solution that allows the control and monitoring of the home. This must be possible over the Internet while also providing strong security to reduce the possibility of the user's home being exposed.

1.3 CONTRIBUTIONS

The work in this dissertation mainly contributes to the Smart Green Home (SGH) project with a solution for the deployment of a HGW in the context of a smart home, using IoT standard protocols to allow the management and control of home devices. The solution was tested and validated with real sensors, while controlling them over the Internet.

This dissertation was presented on a yearly University event called students@deti where students present the projects they have been developing. This event is attended not only by other students and teachers, but also some companies are present at the event.

1.4 STRUCTURE

This document is split into 6 chapters, knowing that the first one, introduction, was already presented the remaining are organized as follows.

- **Chapter 2:** presentation of the state of the art. In this chapter the key concepts for the thematic, will be presented. These will be mainly open-source and standard protocols along with existing software;
- **Chapter 3:** presentation of multiple use cases and requirement analysis, used to assist in the description of the proposed solution. An architecture is also presented and explained in this chapter;

- **Chapter 4:** description of the implementation that was performed for this work. The architecture proposed in Chapter 3 is used to provide a detailed explanation of the implementation decisions that were taken;
- **Chapter 5:** presentation and analysis of the results obtained from testing the implemented solution. In this chapter the test methodology is presented and the results are analyzed under a feasibility perspective;
- **Chapter 6:** presents an overview over the decisions taken during this work, along with an analysis over the initially defined objectives. Additionally, potential improvements and possible future work is also presented.

CHAPTER 2

State of the Art

This chapter is used to provide a description on Domotics, Home Automation System (HAS) and the Internet of Things (IoT), giving an overview of these concepts, their advantages and limitations, and how they are combined to provide better and smarter homes. Additionally some technologies relevant to device management are also exposed as they are pertinent to this dissertation. Also in this chapter, existing solutions and protocols relevant for these concepts are presented. The topics approached in this chapter are all relevant in the development of novel solutions, either choosing from the presented existing solutions, or combining them to achieve a better one.

For each section in this chapter a different topic is approached, exposing previous research developed and its advantages/limitations. This chapter will allow to justify the choices made further ahead regarding technologies and existing solutions.

2.1 DOMOTICS

Domotics is an existing concept on the market that achieved a great popularity in the last decade with the growth of IoT. Historic motivation for these systems range from increasing the comfort and quality of life of its residents to increasing the security of the house itself[7]. The main use case of this type of system is to reduce human labor in the management of the home, but more noble uses can be achieved like assisting the elderly or handicap residents in their daily lives[7].

More recently, new motivations were found and explored namely the reduction of the energy impact of a home in the environment[8]. According to the same author, in European countries 76% of the energy used in houses goes towards Heating, Ventilation and Air Conditioning (HVAC). With this in mind, it is clear that a new way of managing the energy consumption, in order to make it more efficient, is needed. It is stated as well that today's architecture for houses puts an emphasis on passive methods for improving energy efficiency, however a less common approach to residential buildings is an active control of HVAC systems. By actively monitoring and constant sensor gathering, a significant improvement in energy efficiency can be achieved, where the smart house could produce its energy, consume it, store it for later use, and even return excess to the electrical grid.

Even though HVAC systems are responsible for the majority of the energy consumption in a house, other areas could also benefit from a good management to reduce waste and consequently increase efficiency. When looking closely at the daily habits of a person, it is clear that a lot of them are extremely inefficient[9]. One example would be having the air conditioning on with all the windows open. Another common one is having a light on with no people in the room, or while there is still enough natural light outside. While all of these little things can be very difficult for a human to monitor and track, for a smart home it is performed effortlessly.

A good example of a smart home improvement over a *standard* house is given by the previous author[9], where supposing the house is, everyday, empty from 9am to 5pm due to the habitants being at work, the house could, automatically, turn off all sources of power (except the security system and the network) until 4:45pm giving enough time for the house to begin warming up or cooling down in anticipation of its owners arriving. Not only is this extremely convenient and increases the owner comfort but also saves energy in a way not possible without a smart home. Another feature that could improve the savings in these systems is the ability to show the user reports on what is consuming energy or water, for example, giving him more awareness on where to act.

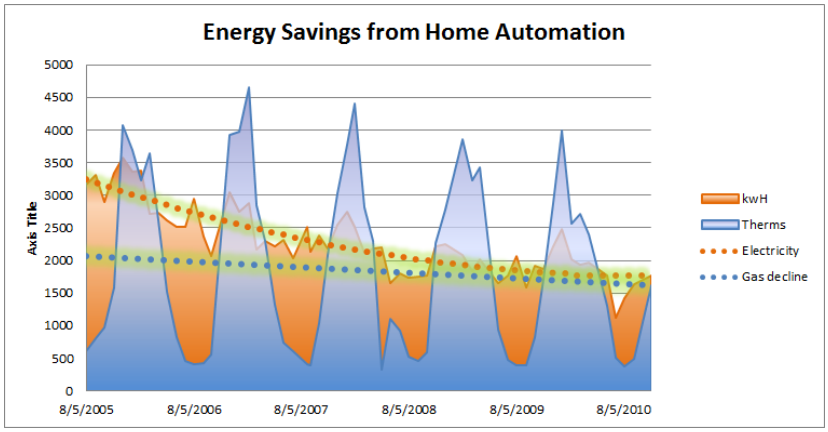

Figure 2.1: Energy savings from home automation [9].

Still the same author mentions a practical experiment conducted about living in a fully automated smart home, where the energy consumption data was tracked and analyzed. In Figure 2.1 it is displayed a chart that shows how the user energy consumption has steadily decreased over time. In order to achieve this kind of energy efficiency his house applied learning algorithms to understand his habits and adjust to his needs. This intelligence inside his house has significantly increased his comfort, while also allowed him to amazingly reduce his energy consumption by over 1000kWh in the period from 2005 to 2010. Besides this example, the author in [9] states that energy payments in houses with smart automation systems reduced over time from 5-15%, this not only saves a good amount of money for house owners, but also reduces the need for energy production thus helping the environment.

To allow the level of intelligence described earlier, a normal approach is to have a centralized controller that is called a Home Gateway (HGW). The HGW connects to all the devices in the house and is responsible for making decisions based on sensors, rules or user input. As an example given by the previous author, the user would input the desired brightness and temperature of a room and with this information, the HGW then requests the sensor values from that room, and based on factors like time of day and outside weather, makes the more efficient decision to satisfy the user request. These systems are highly adaptive to the current state of the environment around them, where looking at the example given, during the afternoon of a sunny day the solution may be to open the blinds and naturally illuminate and heat the room, but on a cold night it would have been to close the blinds and turn on the lights and heating in order to satisfy the user needs.

Although domotics are a definite improvement to the home environment of its inhabitants, there are some issues and problems that are currently being faced by these

systems[9].The bigger issue is the heterogeneity among different devices, manufactured by different companies. Different standards and protocols are used, making a smooth integration between them a very complex process. With the growth of IoT, mentioned further ahead in this document, in section 2.2, one initial approach used to solve this problem was to connect every device to the Internet. This methodology was energy consuming, and better protocols for energy constrained devices were developed and mentioned further ahead on this dissertation as well. Another problem mentioned was the scarcity of Internet Protocol (IP) addresses, this was fixed by introducing Internet Protocol version 6 (IPv6), where the number of IP addresses is greatly increased. Another solution not mentioned by this author, is the use a local HGW this way reducing the number of addresses needed, by using a single IP present on the gateway and all the internal communication being performed using local IPs. One of the other problems mentioned was the ability for these systems to store the home data. While it is true that storage solutions are becoming smaller and cheaper, there are still problems when implementing HAS with low power devices as the HGW. The solution provided in this case was to store the smart home data in cloud platforms, offloading in this case not only the storage, but also the processing of the data to cloud servers that could more efficiently apply the learning algorithms necessary to learn the user habits, this is obviously taken with great skepticism due to security concerns. The last drawback that was mentioned, that is nowadays starting to become outdated, is that these systems, were very expensive and available only to the middle to upper class. Nowadays a HAS for energy conservation is becoming more affordable due to the sensory devices becoming cheaper and the developing of these systems more advanced.

In conclusion, HAS exist for several years now, but with the latest growth of IoT it was open to new functionality and a more efficient operation.

2.1.1 LEGACY DOMOTICS PROTOCOLS

As previously stated, domotics is not a fresh concept being around in the market for over 25 years[2]. During this period multiple technologies have tried to become the market standard, this section presents the most popular protocols used in HAS. It is worth mentioning that the protocols presented below are prior to the IoT era, and should not be mistaken by the protocols developed to be used in the IoT.

2.1.1.1 x10

The x10 protocol was one of the earliest to try and set a standard for home and building automation based on PowerLine Communication (PLC)[2]. Nowadays, x10 is seen as outdated from the perspective of feature set, manufacturer support and robustness. According to [10], x10 is too unreliable and inflexible to be used today as a home-control network technology.

Regardless, still according to the same author, x10 was a pioneer breakthrough technology for its time, that was severely constrained by the existing technology at the moment. An example given by the author on these constrains is, for example, the limit of only 256 devices for each powerline due to the way it addresses devices. Another limitation is the existence of only 16 commands, being that 6 of them are specific for lights, leaving only a small subset for generic devices.

2.1.1.2 KNX

KNX is an open system that claims the ability to cover entirely applications for home and building automation[11]. This protocol follow two main aspects: standardization and certification, thus it requires specific hardware. The KNX protocol gets its roots from 3 other solutions: European Installation Bus (EIB), Batibus and European Home Systems (EHS)[12][2]. The goal of this merge was to create a single European Home and Building automation system standard. Europe is the KNX's main market, where it leads the high-end home automation segment, according to the previous author.

The KNX protocol is supported by numerous devices with all types of characteristics, like systems to control lighting, shading, room climate, energy management and security[13]. The communication between devices in KNX can be done using multiple mediums: twisted pair cables, RF, PLC or IP tunneling, all mentioned by previous authors.

The KNX protocol is an advanced solution and because of that it requires a complex configuration. To help solve this problem the KNX association released two configuration methods that are demonstrated in Figure 2.2. The first method is refereed as the S-mode, and this is designed for large size installations requiring KNX certified planners and the KNX configuration tool named Engineering tool software (ETS). This mode provides the most flexibility and configuration options to the installations[14]. The second mode, named E-mode, is meant for smaller installations, for installers with basic knowledge of the KNX system. In this mode, the devices already come pre-programed with a default set of parameters that offer limited functionality and, with a simple tool each

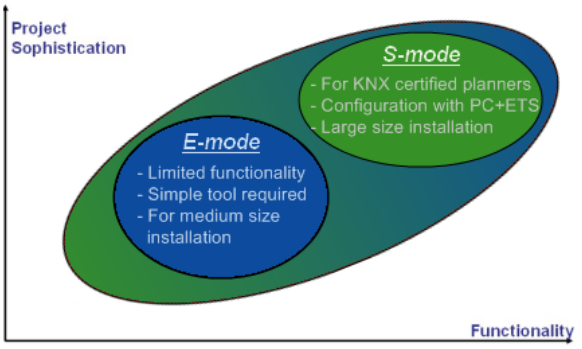

Figure 2.2: KNX Configuration Methods [14].

component can be partially reconfigured to fit the installation.

2.2 Internet of Things (IoT)

The concept of IoT was introduced by Kevin Ashton in 1999[15] in a presentation where he exposed how we, Humans, were the main source of information on the Internet, and as a result most of this information was related to human ideas. Having said this, the author then mentioned his idea of having the computers knowing everything there is to know about the things around us. This way, they would be able to track and count everything, greatly reducing waste, loss and cost. Also said by the author, even though humans were the main source of information on the Internet, the problem is they have limited time, attention and accuracy, and because of this computers needed to be empowered with their own means of gathering information of the world around them.

During the last years, the concept of IoT has evolved, and according to the definition of [3], IoT is the point when more "things or objects" are connected to the Internet than people. Also according to the same author that point was reached sometime between 2008 and 2009.

The constant success and growth of IoT has lead projections, such as [16] visible in Figure 2.3, to believe that by 2020 there will be 34 billion devices connected. Other authors project that by the same year the number of connected devices will be 50 billion[3]. According to the author of [17] the current ratio of devices per person is of 3.64, but considering the population growth[18] and the predictions in [16] this number is expected to go up in 2020 to a little less than 5. Even though this number seems low considering the rising amount of devices that surround us, note that this number considers the entire world population, while in truth, and according to [19], the current

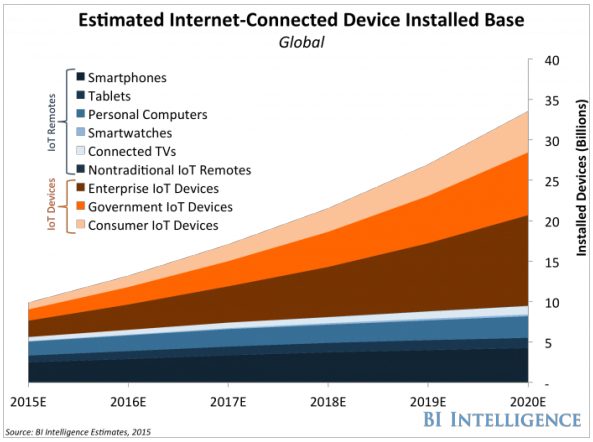

Figure 2.3: Global Estimated Internet-Connected Device Installed Base [16].

amount of people with access to Internet is just 47.1% and the prediction of that number reaching 60% would not happen until 2021. With this in mind, the number of connected devices per person should be much higher.

According to the author in [3], IoT is the first real evolution of the Internet, providing revolutionary applications that will dramatically improve the way people live, learn, work and entertain themselves. This is achieved by expanding the Internet into places unreachable so far. Still according to the same author, IoT will have an even bigger role in today's society. To understand this vision, it is important to first refer the main characteristic that allowed humans to evolve, communication. The current society as we know it today was only possible because of an evolutionary process based on sharing knowledge and wisdom to allow new discoveries on top of what is already known. As the world population increases, it becomes even more important the sharing of knowledge and wisdom to provide a more efficient usage of resources and consequently allow the survival and thrive of mankind. IoT along with all the information it provides by sensing, collecting and transmitting will allow for faster gathering of new knowledge and wisdom, allowing this way for the planet population to continue having healthy, fulfilling and comfortable lives.

Currently IoT is still facing some challenges that are slowing down its development and deployment. As stated by the previous author IoT is facing three main barriers, the deployment of IPv6, sensor energy and the lack of standards. To further explain, IPv6 is going to be essential for IoT seeing that potentially billions of new devices will be introduced in the network all requiring unique IP addresses. More than that, IPv6 allows an easier management of the network due to features like auto configuration and improved security. Regarding the sensor energy consumption, the sensors will need to be self-sustaining in order to allow for long term deployment since it is unimaginable

to replace batteries in billions of devices deployed all around the world. Lastly, the lack of standards is one of the biggest problem with IoT at the moment, and without it the inter communication between devices is impossible and thus invalidating any other efforts.

To conclude, IoT has changed the paradigm of development in some areas, an example of this is the HAS that evolved to new solutions since the adoption of the concept. This is supported by the author of [9] where he states HAS is one of the most exciting developments emerging from IoT. This contribution was mainly with new protocols designed for low power and constrained devices thus allowing faster development, easier integration and larger adoption due to reduced price.

2.2.1 Machine-to-Machine (M2M)

Machine-to-Machine (M2M) is a term used to refer to technologies that allow devices to perform network information exchange without human intervention. According to [20], the beginning of M2M dates back several decades and it was used in telemetry and industrial automation solutions. However, due to the lack of standards in this area, the M2M market is highly fragmented and proprietary. Because of this, it results in the lack of widespread deployment.

More recently, the concept of M2M has been associated with IoT, and although M2M and IoT end goal remains identical they should not be used as the same but rather M2M should be seen as a subset of IoT. To further explain this statement, M2M refers only to the communication between devices, while the concept of IoT is much broader as exposed earlier in this document.

2.2.2 wireless communication standards

One of the core fundamentals of IoT, as the name implies, is the communication between devices. This communication can only be achieved if all devices communicate in the same way with each other. To ensure their interoperability standards had to be established.

This section aims to provide an overview and comparison between the most adopted standards used in IoT. The majority of devices in IoT will require networking capabilities and almost all of them will rely on wireless networks to communicate. Due to this, only wireless communication standards relevant in IoT will be discussed in this topic.

To understand the scope where different standards are designed for, it is important to first mention the Open Systems Interconnection (OSI) layered model. This is a conceptual model that is divided in 7 layers as seen in Figure 2.4 and explained in [21]. When explaining the different standards is important to notice that their scope is not to present an implementation for all layers but rather different standards focus on different layers, and only through combining them it is possible to achieve the full stack for communication between devices. Also to mention that for the 7th layer (Application Layer) there are protocols defined and they are going to be mentioned further ahead in the document.

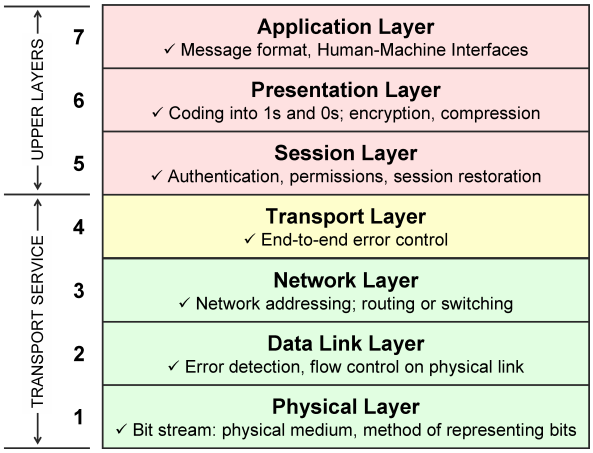

Figure 2.4: OSI model layers [22].

One of the most popular wireless standard was specified by the Institute of Electrical and Electronics Engineers (IEEE)[1] in 1997 [23] with the name IEEE 802.11 also known as Wi-Fi. This standard is a huge success as it is difficult to imagine nowadays a laptop or smartphone without Wi-Fi built in. However the use of this technology in IoT has not been a widespread choice due to the power requirements that it has.

Another very popular wireless standard introduced in 1999 [24] is Bluetooth. This technology is currently maintained by the Bluetooth Special Interest Group (SIG) and in 2016 it announced the fifth version of it, Bluetooth 5.0. Over the years this technology achieved great popularity in short range connections, more commonly connecting smartphones to cars, wearables and other electronic devices.

Although both of these technologies, Wi-Fi and Bluetooth, have solid and mature specifications providing reliable wireless communications, they were not developed with the intent of being used in constrained devices and thus they cannot provide the low power and low resources requirements that IoT needs. With this in mind new specifications were developed such as the ones below.

[1]https://www.ieee.org/index.html

2.2.2.1 IEEE 802.15.4

IEEE 802.15.4 is a standard radio technology developed within the Institute of Electrical and Electronics Engineers (IEEE) by the 802.15 Personal Area Network (PAN) working group for low-power and low-data-rate applications [2]. This standard allows for a maximum data rate of 250,000 bits/s with a maximum output power of 1mW, having its main focus on low-cost and low-complexity hardware. The range on these devices is in the order of a few tens of meters. All of these factors contributed to IEEE 802.15.4 becoming popular for interconnecting smart objects.

An important aspect to note about IEEE 802.15.4 is that it is not a full stack standard, and it only specifies the two lowest layers of the OSI model(Figure 2.4). Because this standard is intended for low-data applications, the maximum packet size is only 127 bytes and after the required headers for the Data link layer the amount of data available to the application ranges from 86 and 116 bytes. The addressing in this standard is done through a 64 bit address that is unique for each device, moreover due to the low-data-rate nature of the standard a 16 bit address can be used instead to identify the device even though it has only PAN validity. When looking at the frequencies used by this standard there are 3 different values used. In the United States it uses 902-928MHz band while in Europe it uses 868-868.8MHz. The third frequency used by IEEE 802.15.4 is available worldwide and is the 2400-2483.5MHz band. This standard specifies two types of devices: Fully Functional Device (FFD) and Reduced Functional Device (RFD). The RFD are more constrained devices and are only capable of communicating with FFD while the latter are fully capable of communicating with all devices.

2.2.2.2 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) is an open standard defined by the Internet Engineering Task Force (IETF) with the main purpose of allowing the use of IPv6 on low power networks and thus enabling constrain IoT devices to be connected to the Internet with a unique reachable address.

The use of 6LoWPAN with a standard IPv6 network is shown in Figure 2.5, and it is possible to observe that the only component needed for this interoperability is an edge router that routes traffic from IPv6 to 6LoWPAN and vice-versa[26]. This router is needed seeing that the 6LoWPAN standard does not define the lower layers of the OSI model instead it is intended to be used with the previously mentioned standard IEEE 802.15.4. In Figure 2.6 a simple comparison is presented between a 6LoWPAN network stack and an Ethernet one, emphasizing the need of the edge router to translate

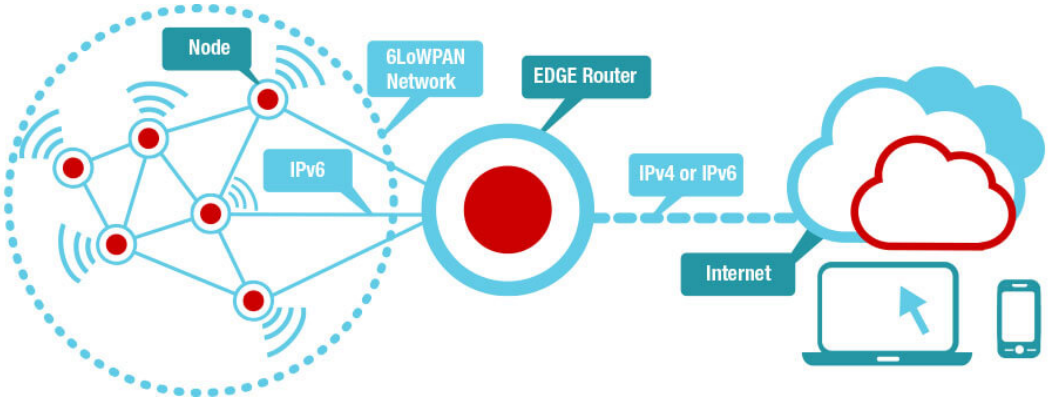

Figure 2.5: Connecting a 6LoWPAN network to the Internet [25].

between the two standards.

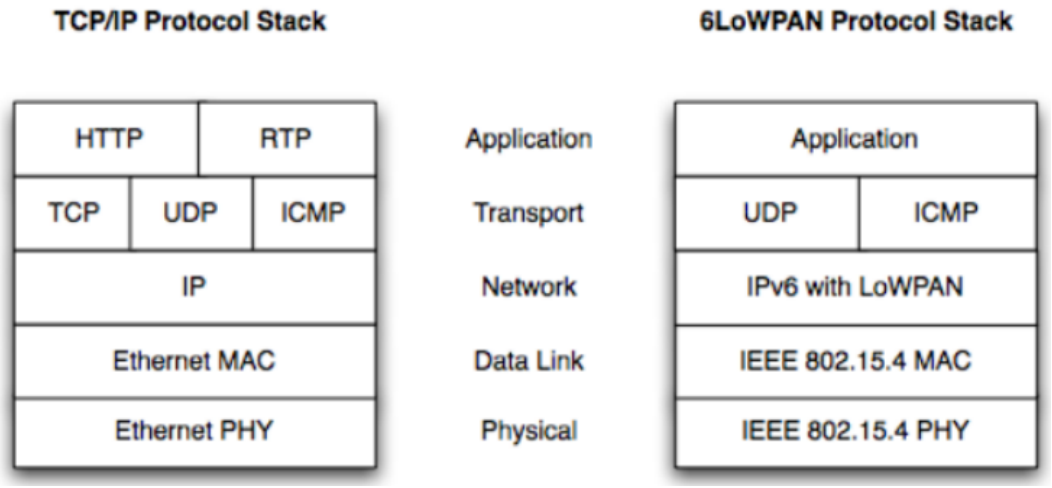

Figure 2.6: The 6LoWPAN stack compared to a standard Ethernet stack [27].

One of the greatest advantages of 6LoWPAN is the ability to natively communicate IPv6 with the network. This eliminates the requirement for gateways like other protocols do that need to keep device and network state and always be synced with all the devices [26]. Because of this, the edge router can remain independent of the application protocol used by the devices hence reducing the processing burden put on it, allowing for a lower cost and lower complexity device.

Because of the low bandwidth natural requirement of the protocol some features had to be implemented in order to allow for IPv6 to be used over such a constrained network. Although 6LoWPAN allows for a full IPv6 Extended Unique Identifier (EUI) address, it can also use a smaller 16 bit short address to reduce the header overhead and lower the memory footprint. Another optimization done by the standard is header compression where it assumes the use of common fields between the IPv6 and User Datagram Protocol (UDP) headers and omits any field that can be derived from the

link layer [26]. Header compression was one of the factors why this protocol can only support IPv6 and not Internet Protocol version 4 (IPv4). Nevertheless, this feature does not force the use of UDP as the transport layer but header compression for Transmission Control Protocol (TCP) is not defined in the standard. Additionally, 6LoWPAN supports fragmentation and reassembly of packets, this becomes crucial as the Maximum Transmission Unit (MTU) of IPv6 is 1280 bytes while the IEEE 802.15.4 is only 127 bytes. Furthermore, the last of the main features of 6LoWPAN is a stateless auto configuration mechanism which enables the devices inside a 6LoWPAN network to automatically generate their own IPv6 address. This mechanism has support for Duplicate Address Detection (DAD).

To sum up, this standard offers an almost native interoperability between an IPv6 and a 6LoWPAN network, while offering major low power benefits. Another aspect this standard does not lack is security where it inherits the strong Advanced Encryption Standard (AES) 128 to provide link authentication and encryption. Furthermore, Transport Layer Security (TLS) is also supported and showed good results when running TCP over 6LoWPAN networks[26]. It also allows for Datagram Transport Layer Security (DTLS) in case of UDP being the transport layer used.

2.2.2.3 ZIGBEE

ZigBee contrary to the last standard is not an open standard, it was developed by the ZigBee Alliance and released to the public in June 2015 [10]. It is a standard that does not define the two lowest layers of the OSI model and instead uses the IEEE 802.15.4 standard, as seen in Figure 2.7.

The ZigBee application layer is defined with three sub-layers: application framework, ZigBee device object and application support sub-layer[29], as seen in Figure 2.7.

The application framework hosts the device applications, defining the services it offers in application objects. Each device can have up to 240 distinct application objects running, for example a light bulb, or a light switch[29].

Following the application framework, there is the ZigBee device object layer that is responsible for performing device management tasks. More specifically, this layer is responsible for defining if a device is a network communication coordinator, or a end device. Additional features of this layer are device and application objects discovery as well as handle binding requests from other devices[29].

Finally, the application support sub-layer is responsible for binding application endpoints and forward their respective messages. Additionally group management is

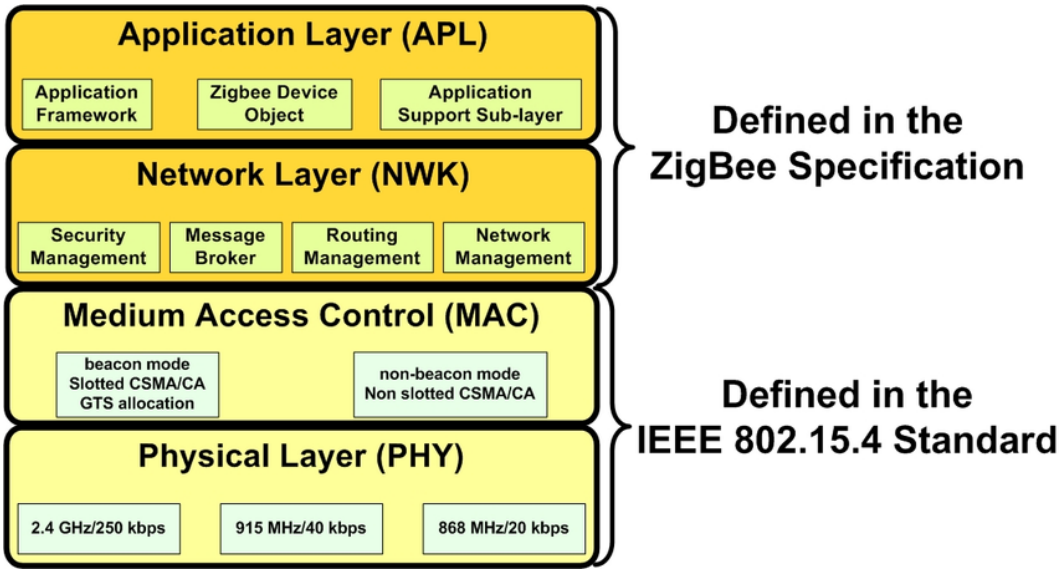

Figure 2.7: The ZigBee network stack [28].

also achieved through this layer using group addressing, where a single address could reach multiple devices[29].

2.2.2.4 Z-WAVE

Z-Wave, as the last standard, is a proprietary solution developed by the Z-Wave alliance. On the other hand, contrary to all the previous mentioned standards, this is not an IP standard and as such it can never communicate with an IP network without a gateway translating the protocols[2].

This protocol relies on a patented low-power RF technology that defines all of its own communication layers, similar to the OSI model[2]. Since this is not an IP based protocol a different device identification is present. To solve this, Z-Wave relies on the network controller, usually also a gateway, to automatically assign unique IDs to connected devices[10].

In order for Original Equipment Manufacturers (OEMs) to develop solutions using Z-Wave, the protocol supplies an API that automatically provides its own certification program[2].

This technology is widely used in home automation such as lighting, temperature and motion sensors, HVAC, energy management, among multiple others. Z-Wave was designed to be plug and play, requiring minimal effort when configuring new devices to the network by the home users.

More recently a new revision of the protocol called Z-Wave plus was released and it

brought improvements regarding not only device battery life, range and bandwidth, but also mechanisms for OTA software updates and easier device inclusion to the existing network[30].

2.2.2.5 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) was released in 2011, as a follow up of Bluetooth 4.0, specifically developed for low-power use cases, allowing it to quickly become a major choice for IoT sensors and actuators. In BLE the number of transmission channels was reduced from the classic Bluetooth, but other characteristic features such as Automatic Frequency Hopping (AFH) are still present making this a robust and reliable protocol[31]. The network stack between the two protocols remained mainly the same as it is visible in Figure 2.8.

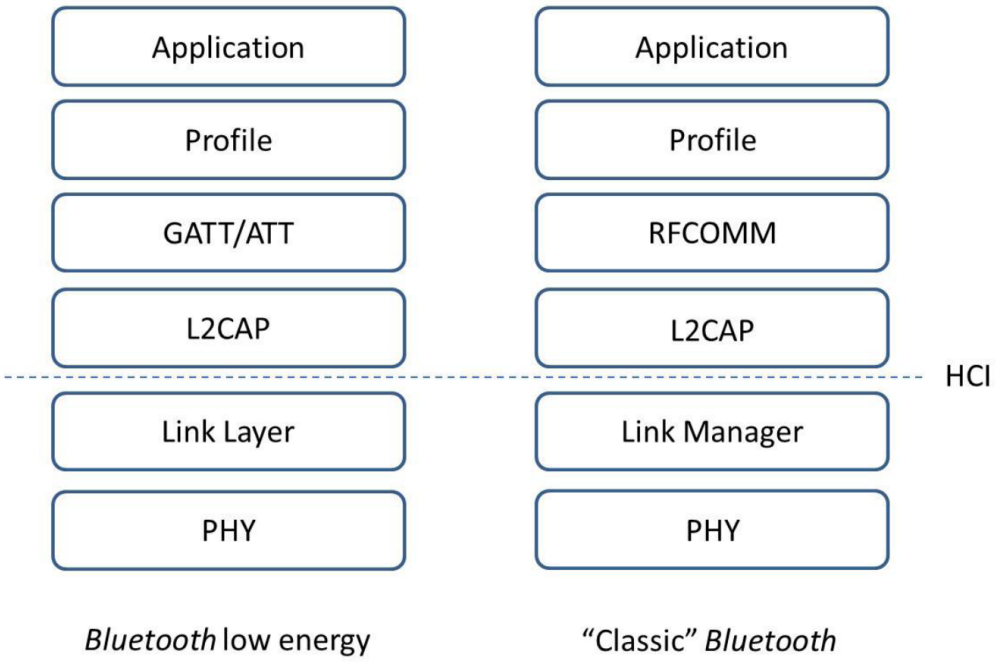

Figure 2.8: Comparison between classic Bluetooth and BLE network stack [32].

Seeing that BLE was not compatible with previous Bluetooth versions, compatibility between devices had to be achieved. From this two class of devices arose: Bluetooth Smart and Bluetooth Smart Ready devices[31]. The former is also called a single-mode device and it is only capable of communicating using BLE. This mode is used for the sensors and actuators where only BLE is required due to its natural constrains. The latter is also called dual-mode and it is capable of communicating with both BLE devices as well as classic Bluetooth devices. This class of device is more fitting for

smartphones, tablets and gateways as they have to communicate with a broader range of devices and their constrains are not as strict.

BLE communication is performed in a client/server model, where the client requests data to the server that provides it. The protocol low-energy is achieved since the devices are in sleep-mode until they receive an advertisement that is sent periodically to scan devices in range, allowing them to find and connect to them. Once devices are connected the data transfer is performed periodically in a server polling mechanism[31].

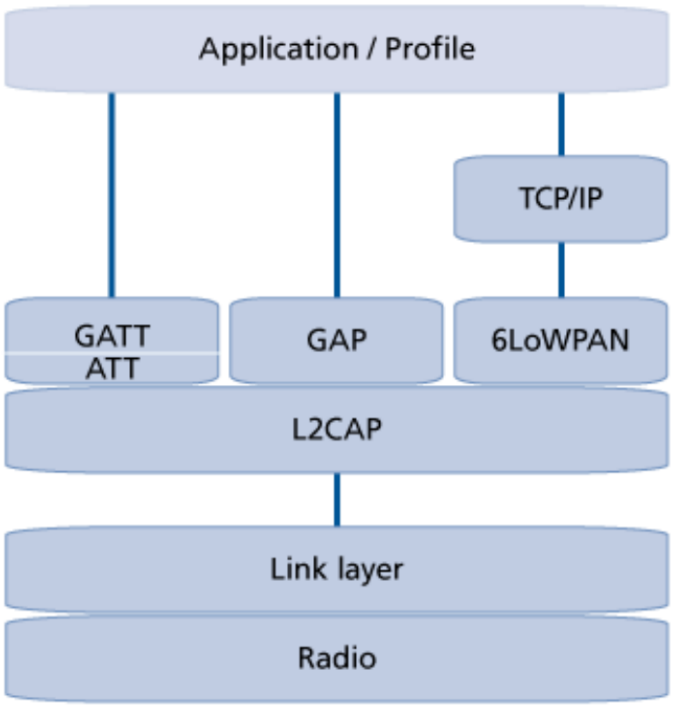

Figure 2.9: BLE network stack [31].

Analyzing the BLE network stack, seen in Figure 2.9, it is visible that after the link layer there is the Logical Link Control and Adaptation Protocol (L2CAP) layer responsible for multiplexing, segment and reassemble data packets when providing them to the higher layers [31]. Above that layer there are three possible choices: Generic Access Protocol (GAP), Generic Attribute Protocol (GATT) and 6LoWPAN. Looking at GAP this layer defines generic methods regarding device discovery and link management when connecting to Bluetooth devices. The GATT provides service discovery as well as methods to allow attributes to be read and written. This is the layer used to develop services and profiles on top of. Finally the 6LoWPAN, previously mentioned in this document, provides an alternative to the GATT layer giving access to TCP/IP communications.

2.2.2.6 IEEE 802.11AH

Wi-Fi HaLow also known as IEEE 802.11ah is a protocol based on traditional Wi-Fi, offering better range and efficiency to the connected devices. This protocol was developed from scratch for IoT and thus all its definition goes towards a low-power communication allowing for sensors and actuators to be deployed on battery. This protocol operates at under 1Ghz allowing it to have ranges in the order of one kilometer with a single Access Point (AP), consequently the bandwidth available at this range is in the order of 150Kbps[33].

Another important feature of this protocol is the scalability it offers by allowing up to 8191 devices to be connected to the same AP. This is achieved by using an hierarchical Traffic Indication Map (TIM) structure[33], where instead of using a sleep-wake mechanism with random wake times provoking large amounts of collisions, as the number of devices increases, the TIM structure allows for the AP to communicate to device groups when they should wake up next[34]. The AP, instead of using the traditional Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), uses a RAW mechanism where it divides the connected devices in groups and allow them all to access the medium concurrently.

To conclude, this protocol also has additional benefits such as a lower association times and end-to-end delay as well as better network capacity when compared to other standards such as IEEE 802.15.4[34].

2.2.2.7 WIRELESS STANDARDS COMPARISON

In this section a comparison between the wireless standards discussed so far is performed. To aid the comparison a table was created (Table 2.1) with the more relevant technologies to IoT. It should be pointed that not all the standards mentioned before are used in this comparison due to not being complete solutions but rather specifications to some layers of the OSI model. Having said this the compared technologies will be ZigBee, Z-Wave, BLE and IEEE 802.11ah. Additionally, two traditional technologies, not meant for the IoT, are also presented as a comparison point Wi-Fi and Bluetooth.

Before proceeding with the comparison, to note that the values present on Table 2.1 for data rate and range are optimal values not always achievable in real world deployment conditions.

When comparing the frequencies used by the protocols it is visible that most of them use the 2.4 Ghz band. Although this is a useful frequency due to being of free use and provides a nice balance between range and data rate, it is widely used by

	Frequencies	Max Data Rate	Max Range	IP Support
ZigBee	2.4 Ghz 915 Mhz (US) 868 Mhz (EU)	250 Kbps	70 m	Yes
Z-Wave	908.42 Mhz (US) 868.42 MHz (EU)	100 Kbps	30 m	No
BLE	2.4 Ghz	1 Mbps	100 m	Yes
IEEE 802.11ah	863-868.6 MHz (EU) 902-928 MHz (US)	150 Kbps	1 Km	Yes
Wi-Fi	2.4 Ghz 5 Ghz	300 Mbps on 2.4 Ghz 1 Gbps on 5 Ghz	30 m on 2.4 Ghz 10 m on 5 Ghz	Yes
Bluetooth	2.4 Ghz	3 Mbps	10 m	No

Table 2.1: IoT Wireless Standards Comparison.

technologies such as Wi-Fi that overloads it. On the other hand, other standards use lower frequencies closer to 1 Ghz, allowing them to have improved range at the cost of the data rates it can achieve. Knowing the own nature of the use of these standards does not require large amounts of bandwidth, this an acceptable trade-off.

Regarding data rate, the protocols not meant for the IoT clearly have an advantage, but this is expected as they are not meant nor need large bandwidth to perform their duty. Instead these protocols provide better energy efficiency allowing them to run on batteries for a longer amount of time. When comparing only the IoT protocols, BLE has the better data rate but is also the less energy friendly protocol out of the other compared.

Looking at the range offered by these protocols, there is a clear winner with ranges units greater than the others. IEEE 802.11ah has the greater range, but is also the newest and more expensive technology. When looking at any of the other IoT standards, all of them have enough range to supply most homes with full device coverage. Moreover, most of these technologies provide ways for signal repetition expanding this way their range.

Finally, when comparing the IP support of the IoT standards most of them have it, expect Z-Wave. Even though, IP support is not a necessity for any standard, the end goal of any HAS is to provide their devices to the user over the network. Knowing this, having IP support facilitates the protocol translation performed by the gateway when exposing the devices. Anyhow Z-Wave gateways are also able to perform this translation and exposing the connected devices to the HAS.

To conclude, all of the discussed IoT standards have its advantages and drawbacks. Despite this, the more common ones used in home systems are ZigBee and Z-Wave.

2.2.3 APPLICATION PROTOCOLS

With the constant growth of connected devices originating from IoT and most of those devices being constrained in terms of resources, one of the current major challenges is efficiently supporting communication between them over also constrained wireless networks, in terms of bandwidth and latency[35].

In this section, a brief description of some application protocols used in IoT will be presented as well as their main features. It is important to know that not all the protocols here presented aim to solve the same use cases, and thus the discussing and comparison in this section will not be to find the better protocol but rather to present the situations where one is more appropriate than others.

2.2.3.1 Message Queue Telemetry Transport (MQTT)

Message Queue Telemetry Transport (MQTT) is a lightweight M2M communication protocol developed by IBM and later turned open source[37]. This protocol is based on a client/server publish/subscribe model as seen in Figure 2.10, and this shows the three main components of the system: publishers, subscribers and a broker[38]. Providing an example for an IoT scenario there is the central MQTT broker that is the server

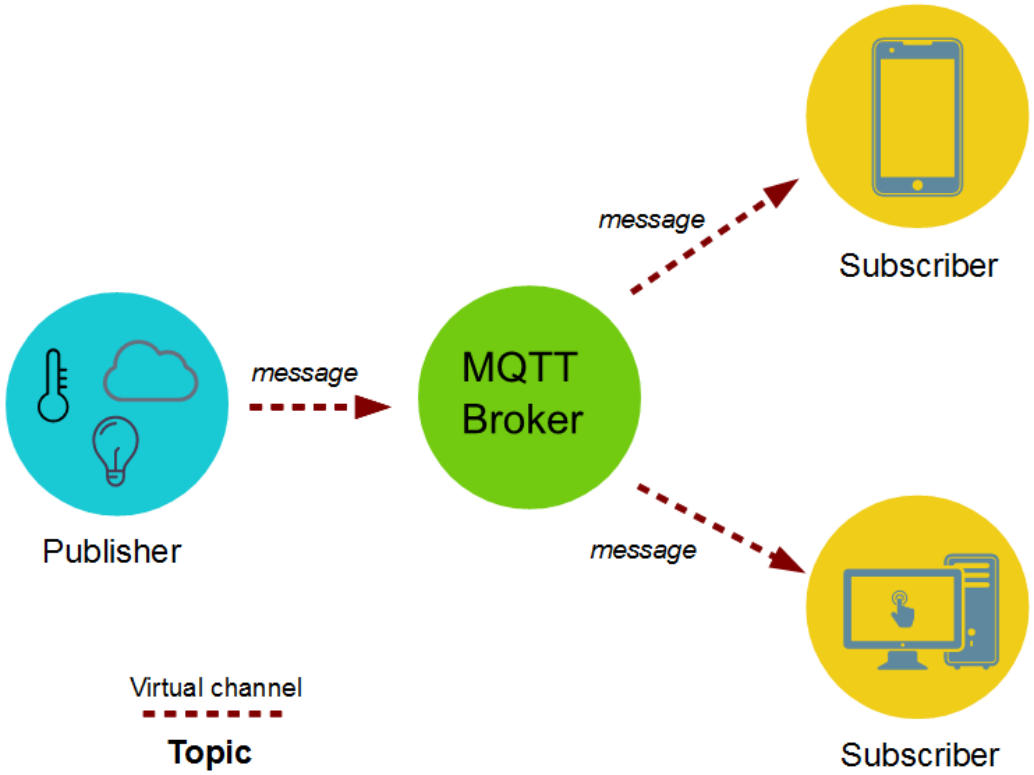

Figure 2.10: MQTT protocol architecture [36].

component of the system. From there, the client that is providing data to the system is called a publisher, and the clients consuming data are subscribers. A client can be simultaneously a publisher and a subscriber for different topics. Given that this protocol is designed to minimize the use of both memory and power, it makes it an appealing choice for IoT devices[37].

Regarding the communication, all the messages are sent to a topic in which the broker then forwards the received message to all the subscribed clients of that same topic[37]. Due to this architecture, MQTT can support communications of 4 different types: one-to-one, one-to-many, many-to-one, many-to-many[35].

MQTT can support Quality of Service (QoS) at the application level, providing three different levels: 0, 1 and 2. Level 0 is also called "Fire and Forget"[37] and it is the more basic level, when no assurance of the message delivery is given. The next level, level 1, is called "Delivered at least once"[37] and it guarantees that the message is delivered, but the message can be delivered multiple times if a broker timeout is reached without receiving the acknowledge message. Finally the highest QoS level, level 2, also known as "Delivered exactly once"[37] guarantees that the message is delivered exactly once to all the receivers. Even though level 2 is the better level, it is also the slowest one and it induces the higher overhead in the communication. This choice must be considered taking into account the network conditions where the application will be running and also the requirements of the application itself.

Another important characteristic of the protocol is the last will message, when upon connection to the broker, clients can register a message to be sent to a topic in case their connection gets abruptly terminated, this way notifying other clients of the fault. This is an easy and integrated fault control mechanism provided by MQTT.

When looking at security concerns, MQTT provides an authentication mechanism for clients using a username and password. Additionally, it also provides mechanisms to encrypt the payload with SSL/TLS.

MQTT works on top of TCP, and although it can benefit from its features it also introduces high overhead and QoS intrinsic to TCP[35]. When deployed on high packet loss networks with limited resource devices this could become problematic due to the amount of traffic it would be generating with retransmissions[37].

An additional problem of MQTT arises from the constant open TCP connection to the broker, that even though it is good in terms of bandwidth save it does not allow devices to enter their sleep modes so often[37].

The final drawback of this protocol comes from the fact that topics are addressed from

a string often long, this is impractical, due to message size, when using in conjunction with the IEEE 802.15.4 standard[37], previously discussed.

Lastly, most of these drawbacks can be overcome by using the Message Queue Telemetry Transport For Sensor Networks (MQTT-SN) protocol[37] that can use UDP instead of TCP to reduce the communication overhead, and it also allows for reducing the message size by indexing the topic instead of always sending the long string.

2.2.3.2 Advanced Message Queue Protocol (AMQP)

Advanced Message Queue Protocol (AMQP) is similar to MQTT in the sense that it is a messaging protocol. However, AMQP was initially designed to the financial industry[38], giving it additional features not present in other lightweight protocols. This protocol is open source, and supported by a lot of tech companies, banks and even the US Department of Homeland Security[40].

In Figure 2.11 an architecture of the AMQP protocol is presented, and from comparing it with the previous protocol, the main difference is observed by the use of queues instead of directly alerting the subscriber clients. This is a huge advantage by not requiring both the publishers and the subscribers to be online at the time of sending the message, as it gets stored in the queues[40].

Among the many features of AMQP, it stand out for being a highly reliable, scalable and manageable system[41]. This protocol makes the process of message passing between services or devices transparent with no concerns from the application regarding message routing. To further extend the flexibility of the routing process AMQP defines four

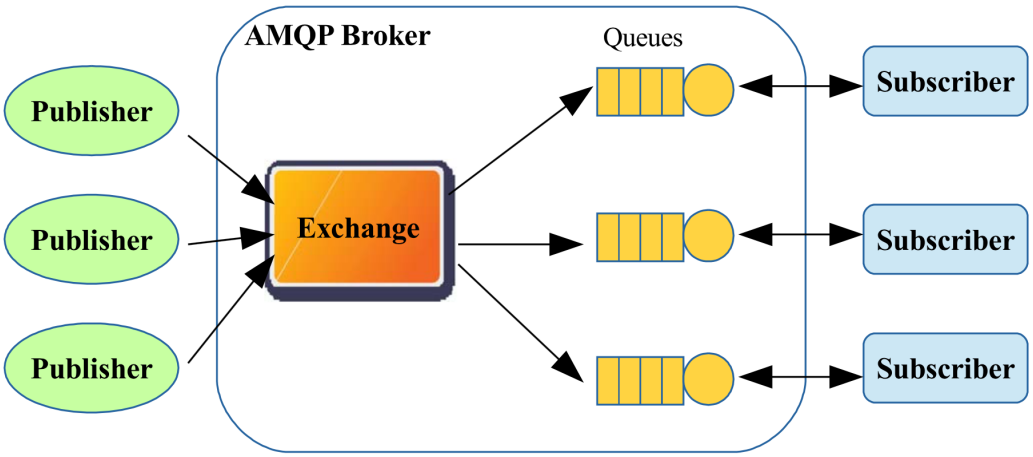

Figure 2.11: AMQP architecture [39].

types of message exchange: direct exchange, topic exchange, fanout exchange and header exchange[41]. The direct exchange method allows for publishers to send messages to the broker that are them assigned to a queue and then forward to a single consumer but it load balances between all consumers of that queue. The topic exchange is similar to MQTT as a message published is then forward to all subscribers of the queues. Fanout exchange is a broadcast mode, where any message is then forward to all queues and received by everyone subscribed to them. Lastly, the header mode uses information in the message header to route it instead of a routing key.

As previously said, this protocol allows for strong reliability, and even allows for all messages to be written to disk using a durability flag. This method allows for message recovery, even after a system failure where the server is rebooted, with no messages lost as long as they are sent with the durability flag. High availability and scalability is also achieved by using clusters of message brokers in a mirroring or distribution topology[41].

2.2.3.3 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is another application protocol designed for the IoT by IETF and it intends to be a replacement solution for Representational State Transfer (REST). REST is a standard interface for communicating on the Internet between clients and servers. It follows a request/response pattern which is synchronous and consequently it blocks the client while it is waiting for the server response. Although this is a mature and broadly used standard, for IoT applications it introduces to much of an overhead for constrain devices to support resulting in an excessive use of power[38]. CoAP comes in as a lightweight solution for IoT devices while still maintaining the

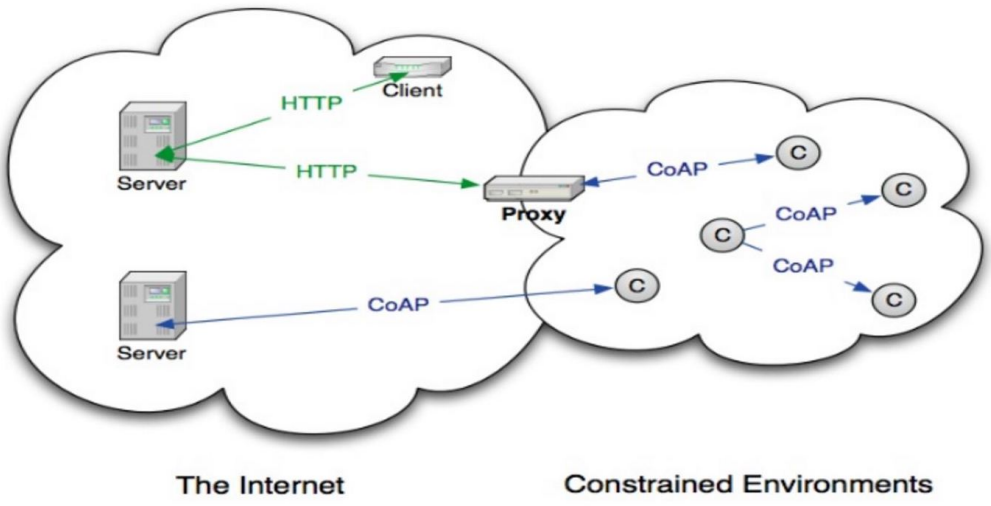

Figure 2.12: CoAP network architecture [42].

request/response pattern unlike the two previously mentioned protocols.

CoAP as seen in Figure 2.12 has a very similar architecture as a normal network does with Hyper Text Transfer Protocol (HTTP), with the perk that it allows communication for much constrained devices. However, the two protocols are not completely compatible, as a proxy is needed between them.

This protocol, to become more lightweight traded the underlying layer from TCP to UDP removing this way all unnecessary overhead. However, it is still able to provide reliability using internal lightweight mechanisms[38].

Internally the CoAP architecture is split into two main sub-layers: messaging and request/response. The messaging layer is responsible for handling failed messages, their duplication and all the reliability factors. The request/response layer handles the communication carrying the message data[38].

CoAP as said is very similar to HTTP and so it also provides similar ways to communicate, using known methods like: GET, POST, PUT and DELETE[43]. The protocol also provides four different messaging modes: confirmable, non-confirmable, piggyback and separate[38]. Confirmable and non-confirmable are the normal communication modes with the difference of one guarantying the delivery of messages and the other not. The piggyback mode is different in the sense that the server response is included in the acknowledge message. Lastly in separate mode the server response is given back in a completely different message.

2.2.3.4 Extensible Messaging and Presence Protocol (XMPP)

Extensible Messaging and Presence Protocol (XMPP) is a protocol originally designed for chatting and Instant Messaging (IM) applications, standardized over a decade ago by IETF[38]. This protocol, due to its purpose, enables near real-time message exchange, presence detection and contacts maintenance[37].

XMPP although operating over TCP has proven to be very efficient over the Internet[38]. Additionally, it provides a flexible communication model allowing the developers to choose between a request/response or a publish/subscribe architecture to their applications[38].

Another advantage of XMPP is its decentralized architecture, where anyone can run their own server and they transfer information between them until it reaches the destination, similar how email works[37].

Despite this, and even though this protocol also has a Simple Authentication and

Security Layer (SASL), it lacks end-to-end encryption which is a necessity for IoT devices and according to [37] one of the biggest flaws of the protocol.

Another downside to this protocol comes from the fact that it has no support for QoS, and because of this message delivery assurance is not possible. This is a major drawback in IoT, according to [37] even more than on IM.

Lastly, XMPP message format is Extensible Markup Language (XML) that, although it is very extensible and flexible allowing to define any type of data in a structured format, it introduces major overhead in the communication due to the large amount of headers and tags it uses[37].

To conclude, even though XMPP has gained some attention for IoT applications according to [38] it is rarely used. Still, this has lead for the protocol to enhance its architecture and improve itself.

2.2.3.5 APPLICATION PROTOCOLS COMPARISON

This subsection presents a comparison between the previously mentioned IoT application protocols, discussing the strong points of each. To aid this comparison a table was created (Table 2.2).

	MQTT	**AMQP**	**CoAP**	**XMPP**
Publish/Subscribe	Yes	Yes	No	Yes
Request/Response	No	No	Yes	Yes
Transport Layer	TCP (UDP with MQTT-SN)	TCP	UDP	TCP
Security	Username/Password Authentication TLS/SSL	TLS SASL	DTLS	TLS SASL
QoS	Yes	Yes	Yes	No

Table 2.2: IoT Application Protocols Comparison.

Starting with MQTT and AMQP they are both messaging protocols based on the publish/subscribe architecture which makes them viable for devices that need to be active the least amount of time possible to save on power. MQTT is more lightweight than AMQP, but consequently it offers less features. In general MQTT is a simpler and faster to deploy protocol with softer requirements on both the broker as well as the clients and it is because of this chosen most of the time. On the other hand, when a bigger system is intended with security and reliability across the Internet is expected, AMQP becomes an appealing choice.

CoAP in contrary to the two previous protocols follows a request/response architecture which is similar to the well known HTTP model. This is ideal for many applications where a protocol following this model is more fitting. CoAP is also lightweight while still providing security and QoS at the protocol level, this is achieved by the use of UDP contrary to the previous protocols.

Lastly, XMPP as previously said is a powerful and flexible protocol offering many features not given by others, buts lacks on fulfilling the lower power constrains imposed by the IoT devices.

2.3 DEVICE MANAGEMENT

Device management is a very important aspect when deploying a product to the market. Vulnerabilities are always found in software and updates are needed. According to [44] when the software updates responsibility is put on the end-users they tend to delay the process either by having had bad experiences in the past or the lack of awareness of their importance regarding security fixes. To provide the necessary management for the devices two architectures can be used direct or indirect management[4]. These are illustrated in Figure 2.13 and are discussed further in the following sections.

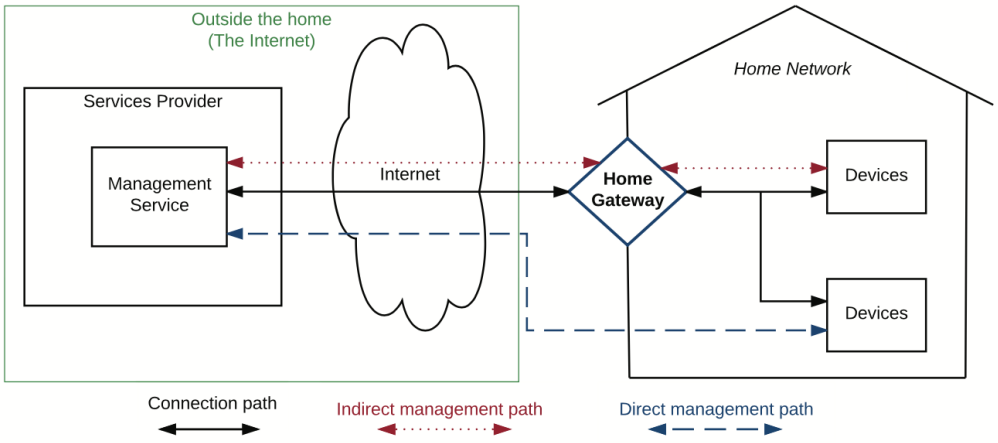

Figure 2.13: Different device management architectures [4].

2.3.1 DIRECT MANAGEMENT

Direct management allows the management service, as seen in Figure 2.13, to take actions directly on the devices without the need for a HGW. This architecture requires

the home devices to maintain a connection to the Internet and to be able to be reachable through it. The communication is done directly, and because of this the HGW can be simpler and can achieve better performance. On the other hand, while simplifying the HGW the devices themselves have to be more capable to be able to handle an Internet connection and the security protocols required. This type of solution is preferred when a greater heterogeneity of devices is present and also when low latency is required, as it eliminates the processing on the HGW.

2.3.2 INDIRECT MANAGEMENT

Indirect management relies on a HGW to communicate to the devices and providing them with access to the management service as seen in Figure 2.13. This approach allows for simpler devices with more constrained resources, and thus cheaper, simpler devices. It also allows for a more protocol diverse network as the gateway can interface between them and the Internet. To the management service the HGW can be seen as a single group of devices this way simplifying the maintenance and configuration tasks, and improving the general scalability of the systems.

2.3.3 DEVICE HARDWARE REQUIREMENTS

To implement an HAS, as said previously, a HGW is needed. In this section some general hardware requirements for such device are discussed, and some example hardware is presented. The HGW needs to be running always on in the users home, and, as such, a low power device is mandatory while still maintaining the required performance to control and automate its entire home. This device would benefit from custom build software images with only the necessary software running on it. This software would be the HAS, an application to manage the device, and the network stack to allow the device to connect to the network. To allow the easier management of a real world production environment, automatic Over the Air (OTA) updates would be required and automatic generation of the embedded images would be ideal. These two aspects are to be discussed further in the next sections.

Hardware boards currently existing on the market that are suited for such tasks are, for example: Raspberry Pi[2], BeagleBone[3] and NanoPi[4].

[2]https://www.raspberrypi.org
[3]http://beagleboard.org/bone
[4]http://www.nanopi.org

2.3.4 BUILDING EMBEDDED IMAGES

The process of being able to deploy the same software to a lot of devices at the same time is complex and when the target devices are on remote locations, stable and consistent software is a requirement. To help solve this problem some automatic embedded image generation systems were analyzed that assisted in creating stable, minimal images ready to be deployed. The next sections will present and compare the two more common popular systems.

2.3.4.1 OPENEMBEDDED/YOCTO PROJECT

The Yocto project is an image generation system, focused on flexibility and device support[45]. This system and OpenEmbedded are, in its roots, the same software where the latter is the build system used by the Yocto project[46]. Yocto differs itself by combining other tools to provide a better, easier and more flexible solution. The Yocto project main goal is to support the widest range of devices possible while providing versatility. This is accomplished by defining its builds from layers which in turn are defined by recipes that define what software to build and how to do it. The layers can be seen as Plug-ins, developed and maintained by the community, where by keeping the software all separated in them allows for low size final images as well as a separation between the core of the build, the Board Support Package (BSP) and the custom packages.

The Yocto project's output is a distribution, and it provides a system to detect changes between builds keeping a cache allowing this way for individual packages to be updated, rebuilt or removed with much faster consecutive builds. This also allows for image creation for different target machines to be much faster that full image rebuilds.

The Yocto project has a new stable release every six months, and provides support with dedicated teams for the current and previous version with regular security patches as well as major updates. Additionally, the community still updates the older releases.

2.3.4.2 BUILDROOT

Buildroot is a image generation system focused on simplicity[45]. It generates a fast and generic root file system image not particularly tailored for any particular use case. When an update is needed, the full image has to be rebuilt, the same is also true for building for different target machines. Buildroot has a new release every three months with the new version containing package updates, new releases and security updates, there is although no support for long term versions.

2.3.4.3 BUILDING EMBEDDED IMAGES SOLUTIONS COMPARISON

When comparing both systems the end goal provided by both can be the same, that is a tailored image for an embedded device. While Buildroot focus more on simplicity, Yocto is more focused on flexibility. Because of this, the Buildroot images can be smaller and faster to build than Yocto's. As an example given in [45] the formers' default build is 2.2 mega bytes and can be completed in 15 minutes, and the latter's default image is 4.9 mega bytes and takes 50 minutes as a fresh build and just over a minute for consecutive builds. Yocto also has a better usable package set with approximately 8400 packages available where Buildroot has just over 1800, with no support for development tools on the target contrary to Yocto.

Regarding the release cycle Yocto's is also better than the Buildroot one, where even though it has a less frequent release cycle, the previous version continues to receive support.

In the end, and according to the previously mentioned authors the decision between systems is not linear and should not be taken without considering the project requirements. An example is given where the Buildroot is given the advantage when stranger architectures are present and smaller images are required, and the Yocto project is given the advantage when multiple different machines must be supported and updates are also necessary. Lastly, the size of the project and their teams should also be considered where Buildroot is more suited for smaller teams with limited resources and Yocto is good for bigger projects where more time and resources can be spend to master the system due to it is steeper learning curve.

2.3.5 Over the Air (OTA) software updates systems

In the previous section systems to generate embedded images were discussed, in this section insight on existing systems to allow the deployment of these images to remotely running devices is given. With a deployed product on the market and potentially thousands or millions of costumers with a device on their premises, it is unimaginable that a software update is performed manually. Moreover, mechanisms need to be implemented to assure no update goes wrong, and if it does that there is the capability of rolling back to a working version. To assure this is true, mechanisms such as A/B partitions[47] can be implemented where there are always two copies of the full file system available, and in case one becomes damaged the system is able to restart from the other partition. Additionally, security mechanisms are very important in these systems to assure the installed updates are legitimate and arrive at the device consistent

with the release by the manufacturer.

Next, some of these systems are going to be presented along with their key features and drawbacks along with a comparison between them at the end.

2.3.5.1 SWUPDATE

SWUpdate is an update system that supports the atomic update of regular Linux images[48]. It has a XML file that contains a description of all sub images currently installed along with their versions. This system is able to operate in both single copy or dual copy mode, the latter is widely used for the safety it provides in having a known good second partition.

SWUpdate also provides a simple web server to allow the upload of update images, however it also has an API that allows it to communicate with any developed software. This system provides a layer to build images with the Yocto project as well it is integrated with Buildroot.

2.3.5.2 MENDER

Mender is an update framework based on the dual copy partition system previously mentioned[48]. It contains end-to-end device management support for updates as well as rollback to the previous version in case it needs to, using the second partition. Mender uses the U-boot[5] bootloader to manage between the different partitions, but as a downside it does not allow for its update nor does it allow for incremental updates, only full os updates.

Mender provides a layer to allow building images with the OpenEmbedded/Yocto Project system supported by Mender itself. It also provides an OTA delivering server with a dashboard, this combined with the device side software to manage and deploy updates, makes the process easier and provides a centralized view of all devices allowing the creation of groups of devices to allow for bulk updating.

2.3.5.3 RESIN

Resin is an update system for Linux systems based on docker containers. It provides both a client application as well as a server. The server software is responsible for building the packages and containers and register them. Meanwhile, the client application, running itself in a container, monitors the device as well as changes in the remaining

[5]http://www.denx.de/wiki/U-Boot/Documentation

containers versions, updating them every time there is a new version of them. All of these actions can be managed from the cloud infrastructure administrated by resin.io[48]. One major drawback of this system is that it can only update container applications, leaving the base Operating System (OS) binaries immutable and thus not updatable.

Resin also provides a layer for OpenEmbedded/Yocto Project to support building the base image for target deployment. Another important feature of this system is the use of RSA keys for establishing the client/server connection and AES 256 for the data encryption.

This system started as a commercial product and them moved to open source in late 2015. It has support for popular embedded boards and thus it grew some popularity.

2.3.5.4 SWUPD

SWUPD is a software update system developed by Intel to meet requirements of two of their internal projects. Its main purpose is the update of software in small increments but at a fast pace. This system uses the term of bundle as the smallest component that can be installed by the system, and these bundles consists of a set of packages and their respective versions to install[48].

SWUPD has available tools for both the client and the server to manage the updates. The client application supports the addition or removal of bundles, the atomic upgrade to a new OS version and the automatic checking of the server for new updates. The server supports the creation of new bundles and new OS releases, and also the hosting of these files for clients to download them. One downside of this system is it assumes the underlaying system never becomes corrupt and thus it has only one partition.

This system provides a layer to support the build of images with the OpenEmbedded/Yocto Project, that already includes the client tools installed. The support for this system is very limited having only Intel working on it.

2.3.5.5 OSTREE

OSTree is an update system similar to the previous discussed solution where it provides atomic updates to an embedded system in a single partition schema assuming there are no problems with the underlying system. It follows a git tree like approach to track the file system changes and allow for updates or rollbacks[48].

This system provides a management tool to set the deployment tree to the desired versions and prepare the system to use the upgraded packages on the next boot.

The support for this system is scarce, however it is still maintained by some members of Gnome and Red Hat. It also had some submissions for an OpenEmbedded/Yocto Project layer but not official.

2.3.5.6 OTA software updates systems comparison

In this section a comparison will be given between the different previously mentioned systems. This comparison will have in mind the opinions of the author in his comparison[48]. To assist this comparison, Table 2.3 was created, showing the key aspects of these systems. According to the author none of the previous systems alone offer the best case possible for updating a system. Both OSTree and SWUPD offer a way to manage fast pace incremental updates, but lack the safety given by the systems with an A/B partition system. The Mender and Resin system are, according to the author, focused on their commercial aspect and not widely adopted in other projects.

	SWUpdate	Mender	Resin	SWUPD	OSTree
Dual-Copy Support	Yes	Yes	No	No	No
Incremental Updates	No	No	Yes (To packages only not OS)	Yes	Yes
Full OS Updates	Yes	Yes	No	Yes	No
Support	Small Community	Only Mender Software	Started commercially, now has some support	Only by Intel and still very low	Moderate by Gnome and RedHat
Image Generation Support	Yocto BuildRoot	Yocto	Yocto	Yocto	Yocto

Table 2.3: Comparison of the OTA update systems.

Looking at the several points discussed individually on the different systems, it is the author recommendation the use of SWUpdate in a dual copy schema with the integration of OSTree in it, even though he also mentions that SWUpdate still needs to have its features extended he claims the software author is open to it. This way the result is a system that is resilient to the unusual partition corruption, but it is also optimized for the more general small and regular updates.

2.4 Securing remote access of the Home Gateway (HGW)

Having remote access to the home is a very appealing feature of a HAS. According to [2], this brings several new use cases to the system and brings new and powerful

business solutions. Some examples stated by the author, when having remote access, are: checking the home while away, receiving notifications for smoke, door/windows, water and movement sensors and controlling the house HVAC prior to arriving home. More noble use cases are also possible with this system for instance assisted living or helping for elderly people to age independently.

In this section, some aspects that need to be taken into account when there is the need of exposing a service over the Internet, will be discussed. Having security as one of the main concerns, due to sensitive data going through the network, some mechanisms to solve this problem will be explained. Furthermore, the need of performance, scalability and High Availability (HA) when deploying a service on the Internet also brings some concerns to the deployment strategy. This will also be mentioned in the coming sections.

2.4.1 HTTP PROXY ACCESS AND LOAD BALANCING

In this section some existing open source products that allow the deployment of a service on the Internet will be discussed.

One important definition used in this context is reverse proxy. This is depicted in Figure 2.14 and it stands for a server that is inserted in the middle of the communication between the client and the application server. This allows for a better access management as all requests need to go through the same control. This also allows for a lower exposure of the application servers, and the ability to transparently load balance requests between them.

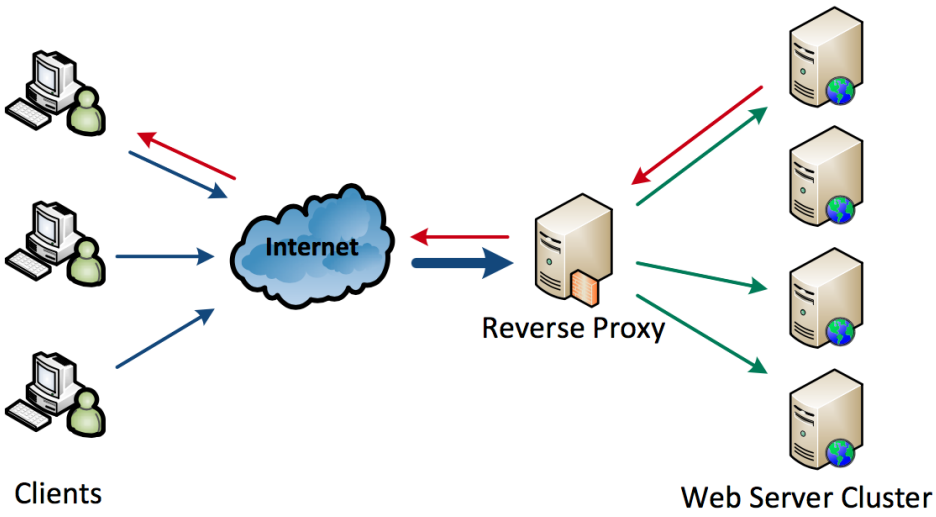

Figure 2.14: System architecture with a reverse proxy [49].

According to [49] the most renowned reverse proxy solutions are Nginx[6] and HAProxy[7], and thus these will be the compared solutions below.

2.4.1.1 NGINX

Nginx original goal is to be a web server but, due to its high efficiency other use cases for it started to be created like load balancing and acting as a reverse proxy[50]. Another major advantage of Nginx is its low resource fingerprint, especially the low memory requirements while handling a large amount of requests[51].

Nginx is open source and highly extensible with plenty of add-ons created by the community[52]. Additionally, there is a non open source version called Nginx Plus that has much more features and is commercially supported[53].

2.4.1.2 HA PROXY

HAProxy is a very high performing solution for load balancing and high availability[54]. It has gained a lot of popularity due to its high performance and it is nowadays used in a lot of very high traffic websites, like for example GitHub[8], Instagram[9], Reddit[10] and many more[55].

HAProxy is open source and has a Plug-in system supported by the community, it also offers commercial support.

2.4.1.3 PROXIES COMPARISON

When comparing both the proxies mentioned previously, Nginx is mainly a web server that can be extended and used as a reverse proxy and load balancer, while HAProxy main goal is to be a reverse proxy and load balancer. They are both very stable, reliable and easy to configure. When it comes to performance, according to the benchmarks performed by [56] there is also not a clear winner with both solutions performing very close. Both of them are open source and support Plug-ins that allow it to extend its base features.

In sum, even though Nginx is a more complete and feature rich solution, when comparing only the reverse proxy and load balancing capabilities there is no winner

[6] https://www.nginx.com
[7] http://www.haproxy.org
[8] https://github.com
[9] https://www.instagram.com
[10] https://www.reddit.com

and both should be looked very carefully when deciding what to use.

2.4.2 Secure Socket Layer (SSL) certificates

Secure Socket Layer (SSL) certificates are usually used to provide data security between a client/server application[57]. A certificate is a combination of multiple fields like owner, validity, issuer, etc. Moreover one of the fields in the certificate is the public key, that using the SSL and TLS protocols allow for secure communication between the two parties. More than providing security in the communication, certificates allow for one party to authenticate the other assuming that an external entity digitally signs the certificate assuring they belong to the rightful owner. The external entity is named a Certificate Authority (CA) and the clients in the connection need to trust it. In this architecture it is crucial that the private key associated with the certificate remains only on the owners machine.

This is a very popular and very successful mechanism implemented nowadays, and has two main use cases to authenticate both the servers as well as the clients. These mechanisms are very important to this work because in a HGW environment is crucial that both sides of the communication are authenticated to minimize the data exposure of the user's home.

2.4.2.1 server certificates

Server certificates are the most common form of certificate authentication used on over 50% of the pages loaded nowadays according to a Google study on the usage of the Chrome browser[58]. This allows for clients to assure they are connecting to the legitimate website while also keeping their data safe. These certificates are issued to a specific domain and are only valid in it[59].

2.4.2.2 client certificates

Client certificates on the other hand are used to identify a user on a system[59] in a unique way. These certificates are mostly used by organizations and companies to enforce stronger authentication to their employers[60]. This authentication mechanism can replace entirely the traditional username and password approach, plus it has additional features such as limiting the user to specific machines to authenticate and easily revoke user access from the CA.

2.5 EXISTING SOLUTIONS

Nowadays, there are on the market various solutions that aim to solve the problem of home automation and smart systems in the home. In this section the most relevant solutions are going to be presented. Some of these solutions are open-source[11], while others are proprietary. For the scope of this dissertation, the open source solutions are more appealing and because of that they are good platforms to base the work on. While all these systems have problems the more common one for the open source ones is the deployment and configuration of these systems are meant for advanced users. On the contrary, the main problem in the proprietary solutions are the lack of third party compatibility. The next sections provides a brief description of the solutions presented and then a comparison between them is performed.

2.5.1 HOME ASSISTANT

Home assistant[12], is a Home automation platform developed using the Python 3[13] language. The goal of this platform is to track and control all devices at home, offering a platform where the user can control and automate the home. This Platform allows the user to set rules on his home, like for example turning the lights on when someone arrives home and it is pass the sunset or turn off all the lights when everybody leaves. These rules can also be overwritten by manual control on a web interface or mobile applications the platform offers, where the user can control all the associated devices at home. This platform is based on components, that work in the same way Plug-ins do, allowing the community to expand its support for new devices and protocols. More recently it was added support for creating and editing automations on the web UI. This project is open source and licensed under the Apache 2.0 license[14].

Home assistant is community driven, with constant updates for bug fixes and feature increase. It was, on December 2016 the 30th most active project on Github[61], and it still is a very active project with more than one release per month.

[11] https://opensource.org/about
[12] https://home-assistant.io/
[13] https://docs.python.org/3/
[14] https://opensource.org/licenses/Apache-2.0

2.5.2 OPENHAB

OpenHAB[15], is a Home automation platform developed using Java[16] as the programming language. The goal of this platform is, as the previous one mentioned , to control and automate all the devices in a home. This platform contains a rule engine that allows the user to create and customize its rules to his needs. This solution has a web-based UI as well as native applications for IOS and Android. Similarly to the previous solution presented, OpenHAB also works with a Plug-in system to allow the integration with new systems and devices. This project is open source and licensed under the EPL-1.0 license[17].

At the time of writing, OpenHAB latest released is 2.1 version that is focused on the deployment on embedded devices and to improve the system usability[62]. To achieve this, the new version has images specifically for the Raspberry Pi, seeing that is one of the most used devices. To solve the other goal a new user interface was introduced to allow the configuration of devices through a UI, also rules can now be defined from the UI.

2.5.3 DOMOTICZ

Domoticz[18], is a home automation platform developed using the C++[19] programming language. Its goal, the same as the others, is to control and automate all the devices in a home. This system's automation rules are based on scripts that run with certain events. This approach, although powerful, can be overwhelming for new users. This solution also has a web interface that is fitted for both desktops and mobile devices. Similarly to the previous solutions presented, Domoticz also works with a Plug-in system to allow the integration with new systems and devices. This project is open source and licensed under the GPL-3.0 license[20].

At the time of writing, the latest version for Domoticz is 3.5877 and the current focus of the project is the support and integration of new devices.

[15] http://www.openhab.org/
[16] **https://www.java.com/en/**.
[17] https://www.eclipse.org/legal/epl-v10.html
[18] https://www.domoticz.com/
[19] http://www.cplusplus.com/
[20] https://www.gnu.org/licenses/gpl-3.0.en.html

2.5.4 SMARTTHINGS

SmartThings[21], is a device manufactured by Samsung with the goal of providing automation and control over a house. This hub has integrated Z-Wave and Zigbee radios[63], that allows it to talk to a vast array of devices. This hub has some local processing capabilities, but for some actions it still requires the cloud. A very useful and unique feature, for devices of this range, is this hub has a battery pack where if the power fails to the gateway, it can still operate and talk to your devices allowing basic functionality on the home.

2.5.5 AMAZON ECHO

Amazon echo is a voice assistant developed by amazon that allows the user to control compatible devices with his voice[64]. This is a very appealing product but is has some major drawback. First of all it has very limited compatibility only having Wi-Fi as a communication technology. Furthermore the devices need to specifically support the Amazon echo. This device also has a mobile application that is used to configure new devices as well as control existing ones. The Amazon echo also requires a constant Internet connection and all data is sent to the cloud. To point that this device has other features not directly related to home automation, like media playback control and calendar/events management, among multiple others[65].

2.5.6 GOOGLE HOME

Google home is a voice assistant developed by Google [66]. It is very similar to the previous mentioned Amazon echo, where it provides control of the devices in the home using voice. It also suffers from the same problem where very specific support has to be added to the devices in order for it to be able to control them. In the same way as the previous solution, Google home also allows more integrations than just home control[67].

2.5.7 APPLE HOME

Apple home is the Apple approach at smart home automation and control[68]. This proprietary solution is available only for Apples's IOS devices, and the support for the home devices needs to be implemented by the manufacturer. This ecosystem also

[21]https://www.smartthings.com

benefits from Apple's voice assistant Siri, and has support for scenes and automations. This system does not have any hub for central control, instead it relies on connecting directly to the Apple devices, adding more performance and resources constrains on the home devices.

2.5.8 VERA

Vera is a company that focus on creating gateways for smart homes where multiple devices from different brands can connect to. For this interoperability to be possible, standards have to be used on both the gateway as well as the devices. Vera offers a range of three gateways with different features and capabilities[69]. All of these gateways have Z-Wave and Wi-Fi connectivity with two model also having ZigBee present. This makes it compatible with a wide range of devices[70]. Although a proprietary device, the approach taken by Vera is closer to the intended in this dissertation.

2.5.9 COMPARING EXISTING SOLUTIONS

In this section a comparison between the so far discussed solutions is presented. This comparison, will be mostly about the open source solutions, seeing the proprietary ones cannot be used as part of this work. Nevertheless, a small comparison between the proprietary solutions is performed at the end of this section.

When comparing the different solutions, one important aspect of comparison is the version and consequently the maturity of a solution. Currently at the time of writing, Home Assistant is at version 0.54 being the newest platform present, but growing at a fast pace. Meanwhile, OpenHAB latest release is a 2.1 version, this is the more mature solution on the market as well as overall the more stable and user friendly. Lastly, Domoticz is currently at version 3.8153 and despite being the highest version number it is not much different from OpenHAB in terms of maturity and stableness, but loses on the user friendly comparison.

Looking at the support given to these platforms, all of them are community supported. This is a positive fact meaning that more people can contribute and evolve the platform with common ideas and features, not relying on the ideas of a single company. Home assistant, as said before, is the newest one of the solutions but also the more supported one, having been on the top 30 open source projects[61]. Due to this fact, it is the platform that, currently, has the most potential in growing and developing new features in the future. When looking at OpenHAB, and their community, it is clear that they are

going in the right direction, in this last 2.0 release their focus was on the accessibility of the users in the platform and it is definitely the point where all these platforms lack. Lastly, Domoticz is also well supported but its focus is towards the integration new devices and fixing bugs. Even though this makes the platform more stable, it does not offer new features or eases the process of deploying it.

Comparing now the development ease for the different platforms, it is important to mention the programming language used in each of them seeing that it is directly correlated to the ease of development. Analyzing Home Assistant, it is written in Python that is one of the easiest and fastest languages to develop in. This is one of the main factors why Home Assistant is the most actively developed solution, of the 3 presented, and has such a short release cycle. Next is OpenHAB that is developed in Java, that is also an easy language to develop in. This, along with being developed for a longer time, made it possible to become the more mature solution out of the 3. At last Domoticz, is developed using C++ which can be said that it is the hardest programming language out of the 3, although it is the one with the better performance. From this results a longer development cycle, and consequently less new features over time.

The next topic of comparison will be the integrations with external components, them being sensors, actuators, or other type of devices. Regarding this topic all the platforms support lots of devices and more important, all of them allow the development of new software to integrate new devices, following a Plug-in pattern. When it comes to the configuration of the home devices, domoticz supports the configuration on the UI with a simple automatic configuration method based on listening for the actions and replicating them. OpenHAB, since the new 2.0 release also supports the configuration of new devices from the web UI, and the discovery and configuration of supported devices. Home Assistant, has started to implement some discovery mechanisms, as well as some UI configuration, still it relies on a YAML Ain't Markup Language (YAML) file for configuration. This makes it, currently, the more limited one regarding this issue.

Comparing another important aspect in the solutions chosen is the ability to automate the house with the definition of rules for the different devices. Regarding this, all 3 solutions have their own mechanisms of handling these rules. Looking first at Home Assistant it uses again a YAML file for the rule definition, although it is starting to allow some rules to be possible to be defined on the web UI. On the other hand Domoticz relies on scripts to apply its automation. These scripts can be regular scripts (Bash, LUA, Python, etc) or blocky scripts that are a more visual approach to define the rules. Lastly, OpenHAB in their new release took a more "user-friendly" approach

by allowing their rules to be defined through the web UI.

As seen all 3 solutions have their own engine for processing rules, the performance of these engines may differ due to multiple factors one being the language they are programmed in which case Domoticz would be better seeing it is written in C++. Although no performance comparisons where done, all the solutions can run on limited hardware, like a RaspberryPi[22], with acceptable performance.

In the last point of the comparison between these platforms their open source licenses will be discussed. Starting with Home Assistant it uses the Apache 2.0 License that is the most permissive license of the 3 solutions. It allows to use the software, change it or add features and commercialize it as long as the original license is included for that part of the code[71]. OpenHAB uses the Eclipse Public License (EPL) that is similar to the General Public License (GPL) but allows to link code under this license to proprietary applications[72]. Domoticz uses the GPLv3 license that is the least permissive license out of the 3 solutions where all the changes or features added to the code must also be released under the same license[73].

The comparison in this section is used to notice the main differences between the different platforms in the aspects presented. In short to conclude, Home Assistant is the newest platform and starting to become very stable and feature rich. It also has the greatest potential to grow in the future. It allows for the easiest expansion due to its ease of coding and to the ease of commercialization because of its open license. OpenHAB was the safe and stable solution for home automation with good support and evolution, but this is starting to not becoming true with the growth of Home Assistant. Lastly, Domoticz is also a stable solution but less used nowadays.

In short, to give a brief comparison of the proprietary solutions, there are two that are very similar them being the Amazon echo and Google Home where the main focus is the voice control and not exactly the home control and automation. These devices are not meant has an HAS and do not support many of the features required by these systems. However these devices provide a great addiction to existing HAS where their only role is to give user the voice control to their homes, leaving the rest of the tasks to the automation system. Currently one of the main drawbacks of these two devices is the few supported languages and the lack of support for region based features. The Apple home is more complete than the previously mentioned systems in terms of home control and automation. It provides automations and can control the entire home providing it has all compatible devices. However this system requires all Apple devices for control and all home devices need to be Apple homekit compatible,

[22]https://www.raspberrypi.org

vastly reducing the compatible device list. Lastly there is Samsung's SmartThings hub that is, of the proprietary solutions, the more "open" to third party devices, supporting many protocols and having two popular radio standards. It also allows community support, which vastly increases the supported devices. Additionally, automations are also supported, even though some require cloud usage. This leads to the main drawback of this solution that it is not local only, and requires to be connected to the cloud all the time. Still it is a very appealing device, but at the moment it is only commercialized in a very narrow market.

CHAPTER 3

Architecture

In this chapter a solution will be proposed to create a Home Gateway (HGW) capable of being deployed remotely in people's homes and have them managed centrally. This central system will be created with flexibility and scalability in mind allowing it for easier growth and evolution. As for the HGW it must support the most popular communication technologies to allow it to control a broad range of devices. Moreover it must support automation rules efficiently for a home installment as well as a user friendly interface for its users to be able to control their homes.

As for the organization of this chapter it will begin by, in section 3.1, give an overview of the Smart Green Home (SGH) project and its main objectives. Following this section the stakeholders and a brief description of their roles will be presented in section 3.2.

The next two sections of the document, section 3.3 and section 3.4, discuss the main aspects that allow to define the solution, seeing that they describe the interaction the users will need to take with the system as well as the requirements that the system demands. Both these sections are split in two sub-sections, one focused on the gateway aspect and the other on the management system, this allows for a more fine separation on the needs of each.

The final section of this chapter, section 3.5, goes over the proposed architecture for the system with a detailed explanation of every component present in it as well as their interaction.

3.1 OVERVIEW AND OBJECTIVES

The solution presented in this dissertation comes in the scope of the project Smart Green Home (SGH). This project aims to develop a device for the home that allows the communication with a vast array of smart objects in a multi protocol environment. This is achieved by incorporating all major standards used for smart home devices inside the gateway.

An additional system needs to be developed to support the gateway with management tasks as well as remote access. This system needs at any time to support all deployed gateways not only for management tasks, that do not have strict time constrains, but also remote access to control and monitor the users home. This requirement imposes some time constrains on the system, and thus scalability and performance need to be a priority when defining the system's architecture. The system in this work will be referred to as the management system/platform or SGH cloud/platform.

The HGW should contain a HAS capable of providing the home's occupants full control of the house and its devices in an easy but complete way. This system must allow the user to expand its range of home devices in the house by allowing him to add new devices, auto discovering or configure them to become a functional part in the existing HAS. The HAS should also allow the easy creation of automation rules, since this is the core of a smart home environment. Another important feature for a system like this is a house log of all actions that happen in it, as having a history allows the user to identify resource wasting situations as well as detect anomalies. Additionally, all the interaction between the user and the HAS should be conducted by a web UI or a mobile application for easier use. The use of a mobile application also allows for extended features like real time notifications and location based automations.

Likewise, the system backing up the gateways also needs to allow its users to access their gateways remotely in a secure way, where all data going through the network is encrypted thus reducing the user's home exposure. Another essential point present in this system is the management capability of the gateways. This needs to be an extensible system allowing for future integrations and consequently extra features. These features can range from a Plug-in performing a heartbeat that can notify the user if its gateway fails, to a Plug-in monitoring the status of the device and reporting software crashes and other unusual events. The possibilities are varied and support for easy new integrations should be present.

Lastly, this entire ecosystem must support a dynamic business model, making it profitable and allowing the continued support resulting from its new features and overall

platform growth.

3.2 STAKEHOLDERS

Any project has multiple people associated to it with different roles and they all contribute to the outcome of it, being by actively developing it or by simply consuming it. It is, in a project, very important to identify all of its stakeholders along with their expectations towards the same, this way ensuring they do not become unpleased or unmotivated with it. This allows for better results and thus increased project success.

In the SGH project four stakeholders were identified, namely:

- Smart Home Occupants
- Smart Home Owners
- Gateway Manufactures
- Developers Team

Smart home occupants and its owners are the primary stakeholders in this project, the former by being the ones that can take advantages of the benefits added by a smart home in terms of increased comfort and task efficiency, and the latter, in addition to the previous benefits, have extra motivation on improving the global home efficiency and consequently reduced bills as well as easier detection of failure in the house systems or required maintenances.

The HGW manufactures will have a financial benefit over the project, where not only can they earn revenue by the sale of the devices but also for service subscriptions given the added value the management system offers.

Lastly, the developer team who is devoted to create a good, functional product that is able to satisfy the requirements of all other stakeholders and thus making it successful.

3.3 USE CASES

To better understand the features offered by both the HGW and the management system, this section presents some examples of use cases for both components of the overall system. The cases covered range from the user interactions with the gateway, to

the client interactions with the management system, as well as interactions between the gateway and the management system.

3.3.1 Home Gateway (HGW)

In this first section the use cases presented will be only the ones a user can have with his HGW device. These will be divided in two sub-sections namely the use cases regarding the HAS and the use cases regarding the device management.

3.3.1.1 device management

1. Login in to the gateway

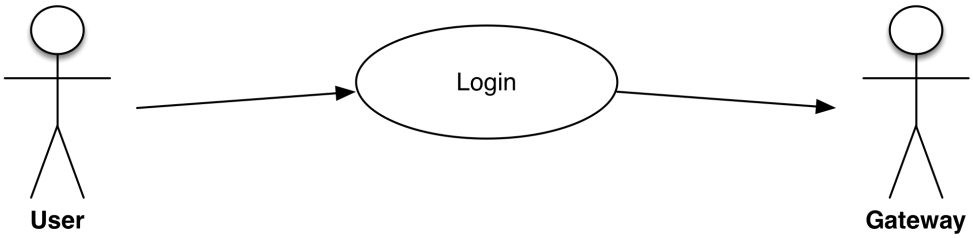

Figure 3.1: Use case diagram: User login to the gateway.

The first use case to consider is when the user first connects his HGW and has to login to it, this is done on a web UI hosted on the gateway itself, and on success it allows the user to access further functionality provided by it.

2. Logout of the gateway

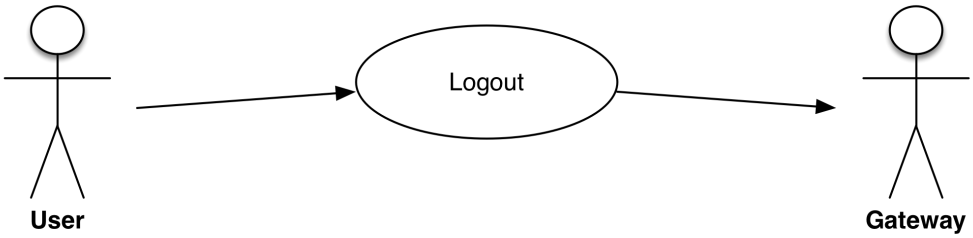

Figure 3.2: Use case diagram: User logout of the gateway.

Following the previous use case, this one presents the contrary action, where the user logs out of the gateway web UI.

3. Register the gateway on the management platform

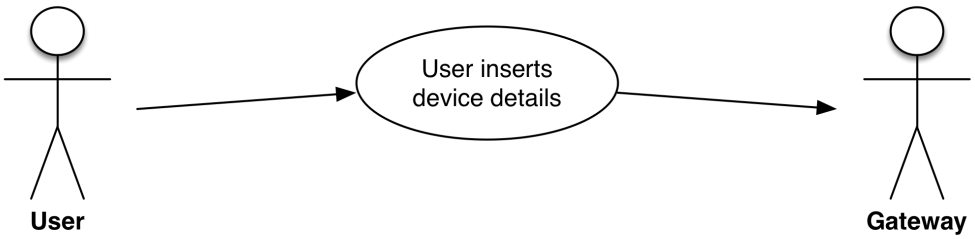

Figure 3.3: Use case diagram: User registers the gateway.

This use case refers to the interaction when the user first connects the gateway in his house and has to register it in the management platform. The registry process must require user interaction to ensure the gateway is linked to the user account. Additionally this process allows for some extra features as adding meta-data about the gateway inserted by the user. This process not only leads to a more information rich HGW, but also helps to mitigate the possibility of rogue gateways.

3.3.1.2 Home Automation System (HAS)

4. Add new home device

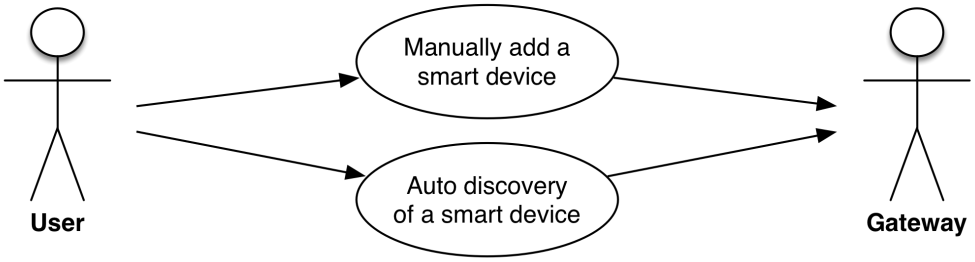

Figure 3.4: Use case diagram: User adds a new device to the HAS.

The user in his smart home may add new devices at any time, due to this fact the system must support the present use case. The HAS must offer, preferably, the auto discovery of new devices only prompting the user for basic information such as location inside the house and a friendly name. In case of device auto discovery not

being available, the system must have a mechanism where the average user can add a new device and configure its basic settings.

5. Add new automation rule

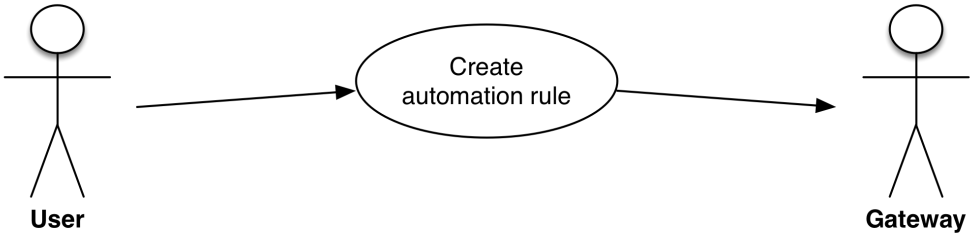

Figure 3.5: Use case diagram: User adds a new automation rule to the HAS.

Every HAS needs to be able to automate tasks through rules. These are personalized rules created by the home's owner to satisfy his, and the house occupants, needs. In order to achieve this the HAS must have a UI to support the easy creation of new rules.

Typical rules contain a trigger, that is the state change that cause the rule to activate, a condition, that may restrict the rule to not be applied if certain situations apply, and finally an action, that signifies what should the rule change in the home environment. Usually the values to be inserted at any of the three sections are device names and their states or values. The automation creator should guide the user with the available options he has at any time during the creation of the rule.

6. Manually control a home device

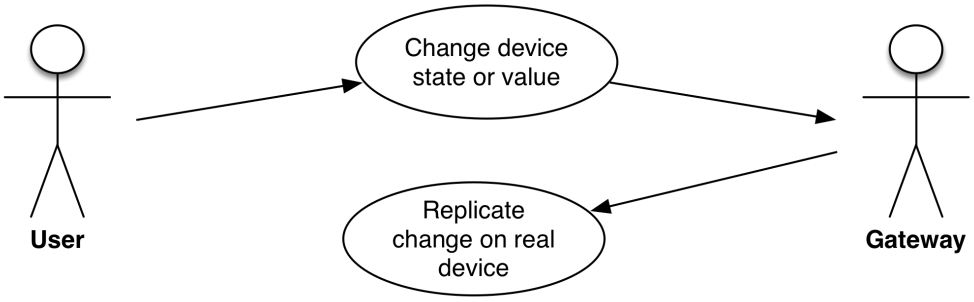

Figure 3.6: Use case diagram: User manually controls a device in his home.

Even though the home may, and should, have automation rules to help its habitants and increase their comfort, there are times where the automation rules will need to

be overridden. To support this use case the system must allow the user to manually control any device at any time, through the provided interfaces, being the web UI or the mobile application.

7. Check house history

Figure 3.7: Use case diagram: User checks the event history of his home.

The user may, at any time, want to check the event history of his home for multiple reasons. Being to check for anomalies or simply to be able to monitor his house. The system must support this and it is in the scope of this necessity that this use case fits.

8. Notify user of events

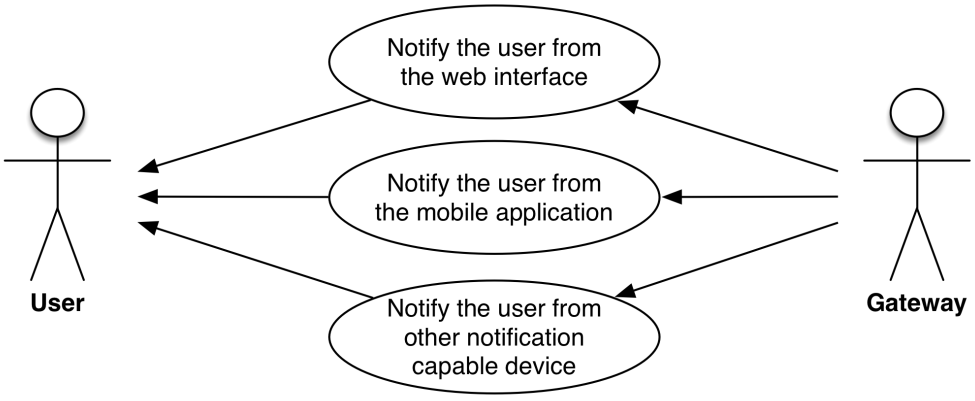

Figure 3.8: Use case diagram: User checks the event history of his home.

In a smart home environment it is crucial to keep the users informed of anomalies as fast as possible. To solve this issue, the best method available is to notify the user and allow him to take action on the situation.

This use case presents three different ways the user can get notified by the HGW. This first case and the more basic one, is a notification sent to the web UI of the

HAS. This is not ideal because it requires the user to be using the web UI in order for him to receive the notification, otherwise it is useless. If the notification needs immediate action, the second solution is to send the notification to the HAS mobile application. This solves the problem of requiring the user to be actively using the system as the mobile application can receive notifications at any time independent of the user's location as long as it is connected to the Internet. This solution is enough when the user is away from home, seeing that nowadays almost no one leaves home without their smartphone. Anyway when the user is home there is still the possibility of him not having its smartphone with him. This is where the third solution to deliver notifications comes in, by using other devices in the house like smart TVs, entertainment systems or any other device capable of delivering information to the user. It gives the user a more convenient way of receiving notifications, giving him the opportunity of acting on or just acknowledging the information.

3.3.2 MANAGEMENT PLATFORM

In this section the use cases presented will be split in the ones where the actor is a user, whether it is a system administrator or a regular consumer of the service, and the ones where the actor is the HGW itself.

3.3.2.1 USER INTERACTIONS

In this section the user cases regarding user interactions with the management system are presented, for both the system administrators and the client users.

9. Login in to the platform

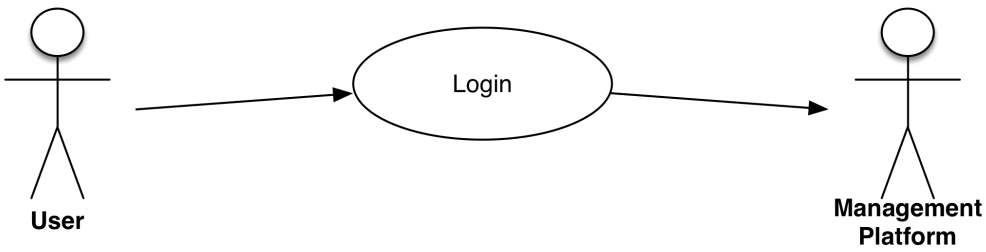

Figure 3.9: Use case diagram: User login to the platform.

This use case considers when the user accesses the online management platform, it allow him to login and, on success, it provides him with further access to the system's

functionalities.

10. Logout of the platform

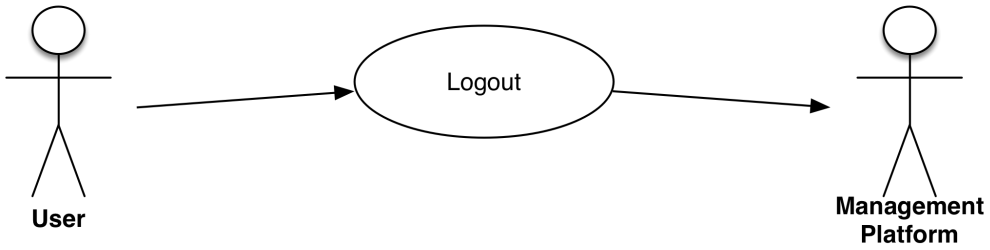

Figure 3.10: Use case diagram: User logout of the platform.

Following the previous use case, this one presents the contrary action, where the user logs out of the management platform web UI.

11. Access the HAS front-end UI

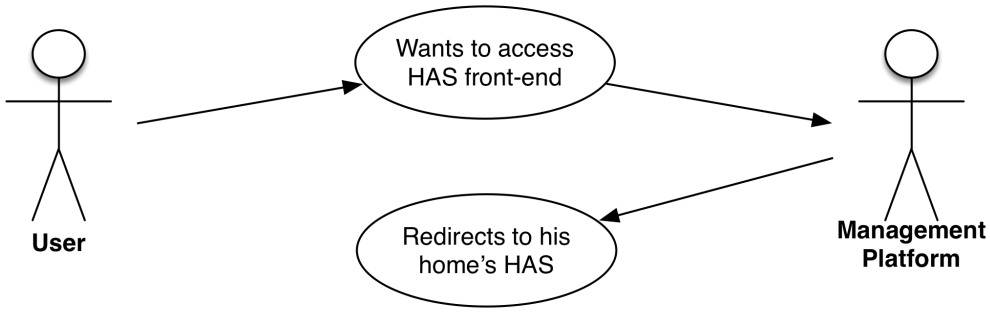

Figure 3.11: Use case diagram: User accesses his home's HAS front-end UI.

When the user is away from home he may want to access his smart home, whether it is to control or simply monitor it this feature must be available. The implementation of this feature needs to offer security mechanisms, seeing that the user's home information will be exposed on the Internet otherwise, resulting in both the home and its occupants privacy compromised.

12. Register a new gateway

Figure 3.12: Use case diagram: User wants to register a new gateway on the platform.

The interaction described by this use case is to be used as complementary to the previously mentioned registry done on the gateway, on use case 3. This two step process helps to validate the authenticity of the gateway, as well as facilitating the process of associating the gateway with its owner, seeing that he must already be logged in to the platform.

3.3.2.2 GATEWAY INTERACTIONS

In this section the gateway interactions with the management system are presented.

13. Register with the management platform

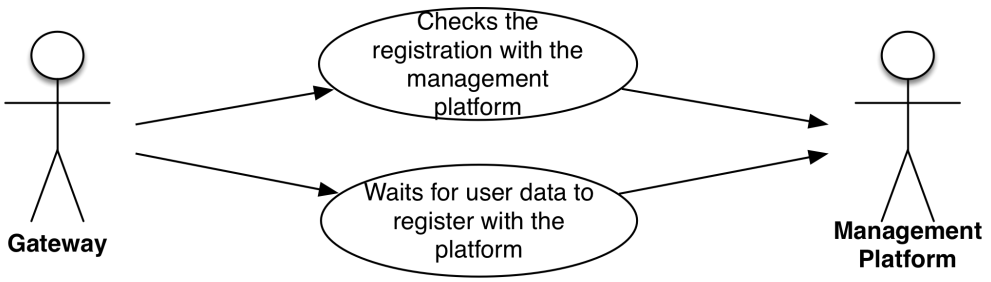

Figure 3.13: Use case diagram: Gateway registers itself on the platform.

Following the two previous registration use cases, this last one combines both of them and it is the one that actually performs the registry. In order to do this, it requires the user to provide information given by the two previous steps and uses that data to validate the operation between the HGW and the management platform. On a successful registry the platform supplies the HGW with the necessary authentication data for future use.

This use case has two possible situations, the previous described one refers to when the HGW is not yet registered to the management platform and has to wait on the user input. The second situation happens when a gateway is already registered in which case it checks its current registration status. This situation does not require user interaction.

14. Authenticate with the management platform

Figure 3.14: Use case diagram: Gateway authenticates itself on the platform.

After the HGW is registered, on any further communication to the management platform the HGW must authenticate itself with the provisioned credentials. This ensures the authenticity of both ends as well as securing the communication.

15. Periodically report to the management platform

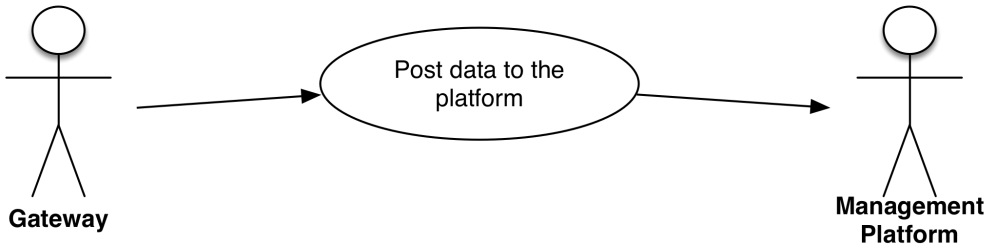

Figure 3.15: Use case diagram: Gateway posts data to the platform.

To ensure the user can be up to date on the status of its gateway, it has to be communicating with the management platform periodically. As it was explained in the previous use case, this communication has to be authenticated and secure.

The more basic example of this communication can be a simple heartbeat to confirm the gateway is in working condition, but more complex and interesting data can be exchanged for example error reports to notify the user or backups of the user's home configuration to assure its safety in case of a gateway failure.

16. Forward HAS front-end to the management platform

Figure 3.16: Use case diagram: Gateway forwards the HAS to the platform.

In order to allow the user to access his house from the Internet the HAS running on the HGW needs to be accessible from outside his network. It is due to this necessity that this use case fits in. The act of exposing a system with full privileges to the user's home raises security concerns, and as such this has to be done using an authenticated secure channel.

17. Check for updates

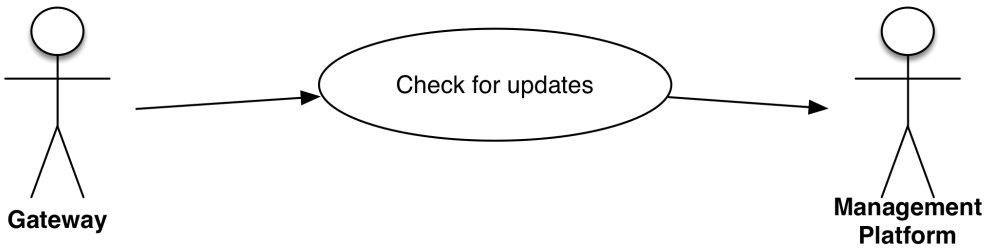

Figure 3.17: Use case diagram: Gateway checks the management platform for updates.

In order to assure the latests features and performance improvements a system needs to be constantly receiving updates. More importantly, are security patches which need to be delivered as soon as possible to the client devices.

If relying on the users to apply the updates, there would be a vast majority of devices not updated. Because of this, their house could become compromised and vulnerable to attacks. Knowing this it was clear that the update process needs to be automatic and transparent, without user interaction. The update system also needs to be resilient to failures and have the least amount of downtime possible.

3.4 REQUIREMENTS

The proposed solution of this work, as previously said, is part of the SGH project and thus it is design for a real-world deployment with real application. For this reason, the requirements presented must be in conformity and fulfill the project's objectives defined in section 3.1. The requirements will be split in two sub-sections, as in some previous sections, to allow for a better distinction between the HGW and the management system.

3.4.1 Home Gateway (HGW)

Automation Rules

One of the most important feature of any HAS is the rule automation system as it allows the user to define how his house should behave given different situations. This is the core of any smart home, and what makes it truly smart, allowing for example the lights to turn on as the user walks into a room and the current ambient light is not enough, and automatically turn them off when he leaves. The previous example is just a small instance of how automations can, in the daily life of users, greatly increase their comfort while reducing the resources usage.

Having described briefly how an automation system is important in a smart home, it is required that this HGW provides one capable of not only performing automations but also allowing the user to create new ones and customize existing ones to his needs.

Performance

From the previous requirement it is know that the HGW needs to allow automation rules in order to be useful for the user. Another very important aspect for this system, in order to be viable to the user, is the performance requirements inherent with the processing required by the HGW. For example, some delay is allowed in automations such as heating and cooling or watering the garden, because their effects do not produce immediate results on the user and so even a few seconds would not be worrisome. On the other hand, when looking at the lights automation, there should by minimal delay between the trigger, being the wall switch or a motion sensor, and the light turning on, as this would be very perceptible and uncomfortable for the user.

Another performance requirement of the HGW is the overall responsiveness of the home monitor and control interface, where it should not feel slow to the user.

Flexibility

The HGW device must support devices from different manufactures using different radio technologies for communication. This level of flexibility on the device allows it to easily fit with existing devices in the user's home as well as when improving the HAS it does not impose as many restrictions thus allowing more freedom to the user in his purchasing choices.

Having a HGW capable of utilizing devices from multiple manufactures as well as with multiple communication standards on a single device also makes it more appealing to the user when comparing the market offerings.

Scalability

Following up on the previous two requirement, flexibility and performance, in the former it was said that the system must allow the user to add new devices to his home as it sees fit. Also, on the latter is was stated that the system must have some performance constrains, where mainly some the automations should not fell delayed by the user.

It is within this scope that this requirement fits itself, as the HGW should allow the user to scale the HAS of his home, by adding new devices, with no noticeable performance degradation.

Failure Handling

Failure handling is also a very important aspect of devices like the HGW, since they will be deployed remotely in people's homes, and consequently presential support will not be available. Furthermore, the technical knowledge of the majority of the home habitants will be very limited or non existing.

To support this requirement the HGW should be able to recover from any software failure automatically, or on the worst case recover with a reboot. The system must easily recover from these failures with the least downtime possible seeing that in a full smart home many critical components will be depending on it.

A very error prone situation for the HGW is during its software update, where it must be able to entirely recover should anything happen during the process. In case of an hardware failure it should notify the user to the occurrence in order for him to replace the device.

User Control

Following the first requirement defined for the HGW, where automation rules were required, this requirement presents an alternative solution to control the home devices, manual user control. As previously said, the main use case for a smart home is the automated use of the home devices, nevertheless there are situations that do not fit in an automation or are of spontaneous origin. To support this, the HGW should allow the user to manually control any home device at any time from, preferably, multiple sources like a web browser or a mobile application.

User Friendly Interfaces

The majority of users of the HGW, and of the system in general, will be non tech-savvy. Due to this fact, all operations that require user interaction need to be intuitive and easily achievable by the common user.

This requirement is linked to the previous one, knowing that the main source of user interaction will be on manually controlling the home devices. For this reason, the HAS UI needs to be easily reached and operated.

Security

The user's home is likely one of the most important assets that he owns, and, in the case of it being a smart home, so is the data it produces and consumes over its network. Because of this, strong security mechanisms need to be implemented in order to keep the house data safe.

In a full smart home all the house systems are connected, and, if remote access is allowed, there has to be a permanent connection to the Internet. This greatly increases the risk of a security breach by having the home habitant's data and privacy compromise. Worst cases could allow for unauthorized persons to enter the house with malicious intents.

To prevent this, the Internet exposure should be minimal with a single, controlled and secure entry-point. In this system the entry-point is the HGW that should be equipped with the necessary security mechanisms and policies to prevent the stated problems.

Automatic Updates

As a follow-up of the previous requirement, this one comes as a very important way to maintain the security levels required by the HGW.

With new vulnerabilities being found over time, security patches need to be issued, and thus a system capable of providing these updates needs to be implemented.

The updates provided to the gateway can contain, not only security fixes but also, new or improved features in order to enrich the user's experience.

3.4.2 MANAGEMENT PLATFORM

Performance

The first requirement for the management platform is the performance constrains it has, where in addition to needing to serve the entirety of the users accessing the platform in a fast and responsive way, it also needs to be able to manage all connected gateways in a way that allows the system's features to fully work without performance compromises.

One major feature where performance is required, is the remote access to the user's home, where the delay felt by user cannot be noticeable, given normal network conditions, when compared to local access.

Flexibility

The management system should be built in a way that it can be used to implement new features over time, providing this way added value to the user.

Additionally, the system must allow a progressive growth where, for example, new hardware devices could be introduced as a HGW and the system must be able to support multiple hardware versions and devices in production. The same goes for software where different hardware devices can be running different versions.

Scalability

With the management system growth both in features and in gateways connected, the system must maintain its performance levels. In order to allow this growth the system needs to scale. The management system developed in this solution must be able to scale on an easy and predictable way allowing for a smooth increase in hardware without any unforeseen events.

High Availability

Similarly to the previous requirement, where additional hardware is introduced to allow the growth in the user base, this requirement is about inserting additional hardware to support a redundant, failover environment.

The concept of high availability, allows for a system to be seen as always online despite of internal hardware fails or maintenance. This is a key concept to be applied in this management system, as the users want to access their homes at any time despite any problems that may be happening with the management infrastructure.

Security

As stated in the security topic of the HGW, this aspect is of major concern when regarding this entire project as sensitive and personal data is involved.

Regarding the management system, the main aspects where data must be absolutely secure are the user account data, all device authentication mechanisms, and the user's home data flow to reach him remotely.

The security in a system like this should be thought from the creation time as an important, integral part and not only as something that must be added when reaching production.

Using this methodology security problems can be avoided, and overall achieve a more secure system.

User Friendly Interfaces

Following what was stated in the similar requirement for the HGW, this system is not to be used by tech-savvy users, and thus all user interaction must be done in a simple and intuitive way, providing this way a good user experience while using the platform.

Remote Access to the User Home

This requirement states one of the crucial features of this system, by allowing users to remotely access their home devices from anywhere at any time.

This access must be done through a secure channel, where the data flowing in it cannot be exposed to the Internet and requires both ends to be authenticated. This feature should always minimize the client's HGW and network exposure to the Internet.

Manage Remote Devices

One of the management platform main requirement is to perform automated maintenance tasks in the deployed HGWs. This again, the same as the previous requirement, should be focus on not exposing the client to the Internet.

When looking at the possible management tasks, there is for example the ability to automatically deploy updates to the devices, this way keeping the HGW always updated thus preventing some security exploits and general errors the software has.

Another use for the management system can be to control the user's access to his home, and grant or revoke it based on factors such as a subscription or abusive access from the user. This can also be used to disconnect gateways from the platform for the same reasons.

Lastly, given flexibility offered by the management system it must support the adding or removing of device features based on commercial preference for different subscriptions models, or because of hardware limitations on that HGW.

3.5 ARCHITECTURE

This section aims to provide an architecture for the Smart Green Home (SGH) project having in mind the previously stated requirements, use cases and objectives.

The solution proposed consists of two major sub-systems, the HGW that will be present at the user's home and the management system that is present at the SGH infrastructure. Both these sub-systems are present in the architecture on Figure 3.18 as they complement each other to provide the user with the required features.

Bellow the architecture Figure, the individual components will be specified, describing their goals and interactions.

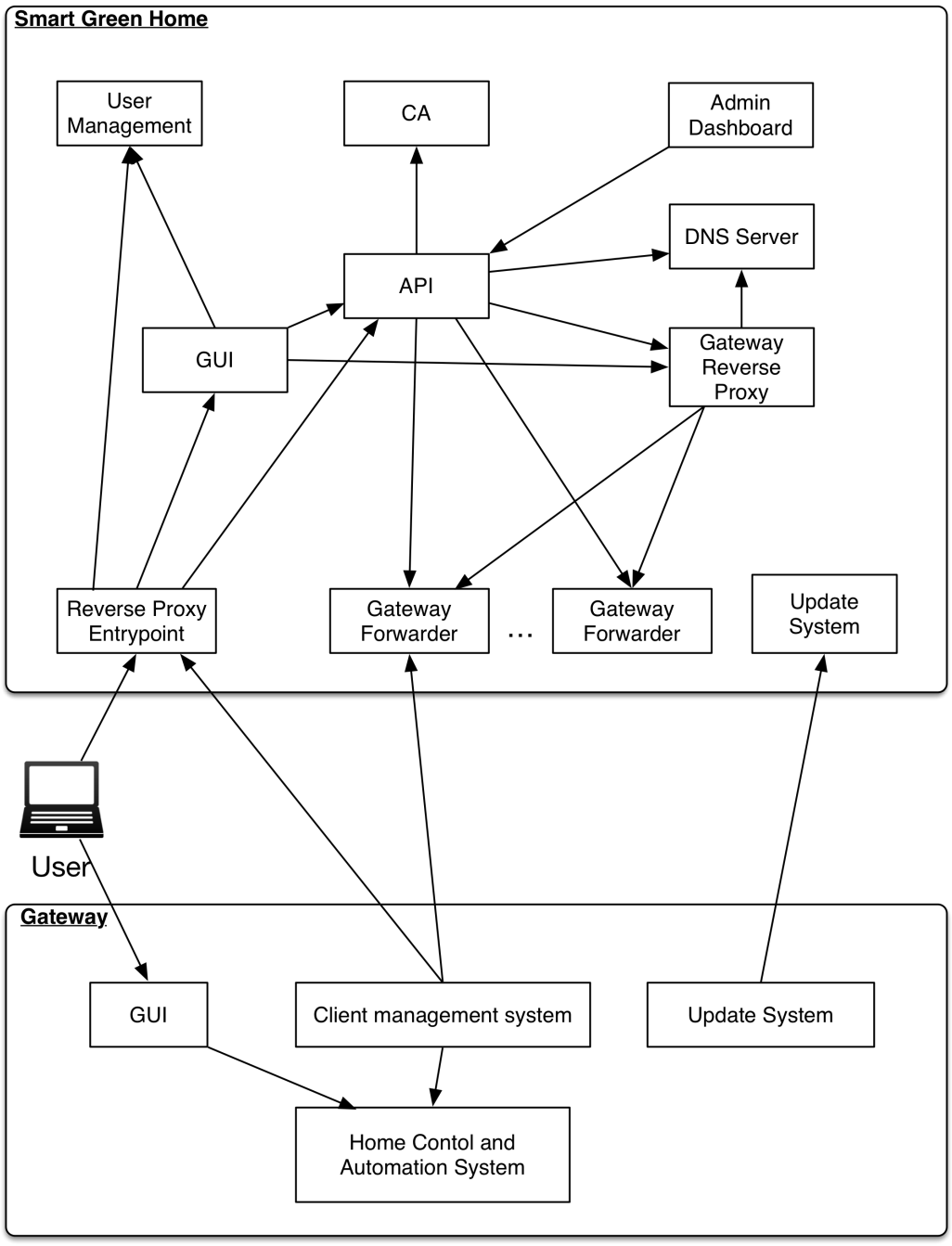

Figure 3.18: Smart Green Home (SGH) System Architecture.

3.5.1 Home Gateway (HGW)

GUI

Users need to be able to interact with the gateway. To satisfy this need a GUI needs to be implemented on the device.

The main goal of the GUI on the gateway is to provide the user with the control and monitor capabilities of his home, allowing him to view and change the state of any home device. Additionally, this should also allow the user to create and edit his home automations as well as manage his connected devices, allowing him to add new ones and remove unwanted ones. Other features that can be offered by this GUI can be a history view off the events occurred at the user's home for information purposes.

Another use of the GUI is to perform management tasks as well as view information about the device itself. Examples of possible management interactions though the GUI are: performing a factory reset, rebooting the device, change the way the device connects to the Internet (Dynamic Host Control Protocol (DHCP), Static IP) and downloading/uploading a configuration backup. These are just examples of possible tasks, the GUI itself should allow for further integrations and improvements over time.

Client Management Platform

The Client Management Platform handles the interaction of the gateway with the SGH platform, securing the communication between them.

This component runs as a background program and is responsible for performing the management features present on the gateway. It must support a Plug-in system to allow for easy future integrations of new functionalities.

Lastly, this component is also responsible for the forwarding of the HAS to enable the remote access to the user.

Update System Client

The HGW should always be updated to the latest possible version, this way receiving security updates, performance improvements and overall bug fixes.

This component is responsible to check a central server and query it for updates. It can them securely download it and apply it to the gateway. All this process should be done without user interaction and with the least downtime possible as well as with safety measures should anything go wrong during the update process.

Home Control and Automation System

The main purpose of the HGW is to connect to the home devices to control and automate them. This a complex component and is expected to perform multiple tasks.

The first and more basic task is device management where this system must have two ways of adding devices. The first one is to automatically discover devices on the network and add them to the HAS upon user confirmation. The second and less user friendly option is when the device has to be manually configured by the user.

After the system has its devices added and configured it needs to continuously check its states for changes while also being able to change them. This state change process has to be done in the least time possible to ensure the HAS internal state matches the real home environment.

Lastly, the HAS needs to able to use the state changes to trigger automation rules, previously defined by the user for his home. These automations can range from simple wall switches to turn the light on to complex events that have multiple triggers as well as conditions associated with the automation result.

3.5.2 MANAGEMENT PLATFORM

User Management

In order for users to be able to remotely access their home and manage their gateways, they need to have user accounts in the SGH platform. This component is present as a solution to this, providing the storage of user data and user authentication and authorization.

This component is used to provide user Identity and Access Management (IAM), to the entire SGH platform allowing it to validate the user between the multiple existing services.

Graphical User Interface (GUI)

All the interactions the users have with the system are processed by this component. It is where the users access the SGH platform home page and from where they can remotely control their HAS.

Additionally, the GUI should be flexible to allow the easy integration of new features to the user. Some examples of basic functionalities are: registering a new gateway with the system, edit details of an existing gateway and edit user profile settings. These are just some simple examples, but more complex and content rich information can be displayed to the user.

Reverse Proxy Entrypoint

This is a very important part of the whole system, being that it routes the traffic arriving at the SGH infrastructure to the correct service running behind it. An architecture of this kind has two main advantages: scalability and security.

Regarding the scalability aspect, a system containing an entrypoint routing the traffic allows for multiple instances of the same service to be running and the incoming requests can be balanced between them, using load balancing algorithms such as round robin or least connected.

When looking at the security advantages from using a entrypoint proxy, there is less exposure to the Internet on the internal services, this way allowing for a less constrained environment on them. Another perk of having this component is the end-to-end security on the communication with the client can end on the proxy instead of on the internal service. This reduces the communication overhead and the overall load on the internal services, allowing them to serve more clients.

Certificate Authority (CA)

Wanting to have end-to-end security and providing authentication to both the servers as well as the gateways requires an entity to help this problem. A CA solves this by issuing certificates to both parties, to which the other services then validate their legitimacy with.

This service needs to provide an interface to automatically generate certificates for new devices, as well as revoke necessary ones.

Main API

All communication originating from the HGW as well as requests from the GUI component are served by this service. It is responsible for providing an interface to consult or modify the current state of the HGWs as well as the services they are allowed to connect to.

Administrator Dashboard

This component provides a GUI focused on performing administrator tasks. It is not meant to be available to the general user, on the contrary, it must be secured to be used only by authorized SGH personal.

This system allows to manually modify the devices state to be used in case of any software error to which manual intervention is required. This is very important, in a deployment scenario, to the costumer support personal. Moreover, this component must allow to revoke gateways or change their current subscriptions plans, influencing their available Plug-ins and features.

Additionally, this system can also provide administrators with real time metrics of the system state, as well as alarms to unusual situations that may occur.

DNS Server

To support such an infrastructure with every HGW requiring to be accessed from the Internet, a Domain Name Server (DNS) server to support the constant adding and removing of entries is necessary.

This service needs to provide an interface to allow for other services to manipulate its entries. Also, it needs to be exposed to the Internet as a name-server for the SGH domain in order for both users as well as other services can resolve the DNS names.

Gateway Forwarder

As said previously, the HGW GUI needs to be forwarded to the user remotely over the Internet. In order to achieve this the server needs to accept connections from the gateways and then share them with the gateway proxy component to make them available to the user.

These servers will have a maximum number of connections allowed, and thus, the existence of multiple instances of this component needs to be possible to allow to scale the system.

Gateway Reverse Proxy

This service is similar to the previously mentioned, and it is responsible for making the HGW available to the user from a single point independent of the real server that is forwarding that gateway.

This approach benefits from the two previously mentioned advantages of having a reverse proxy, being the first the scalability it allows for having multiple servers forwarding the HGW HAS GUI to the user. The second advantage it benefits from is the added security of not requiring the servers forwarding to HGW to be exposed to the Internet.

Update System Server

Constant updates are one of the main factors to keeping a system secure. Having said this, and following the update client component, this service is present in the SGH architecture with the role of hosting the updates, making them available and enforcing them to the devices.

This system must support the deployment of multiple device types possibly with different constrains, hence having the need to manage multiple update images simultaneous and deliver the correct one to each device.

CHAPTER 4

IMPLEMENTATION

On the previous chapter an overview of the HGW and the management system was presented, giving a description of the system's use cases and requirements. Additionally, a possible architecture to satisfy its needs was presented explaining its components and their roles.

This chapter's objective is to describe the implementation route that was taken, specifying the design decisions chosen along with the reasoning behind them. This will be split in several sections, explaining different phases of the development as well as how all of them come together to form the final solution.

It starts by defining, in section 4.1, the implementation objectives with regard to the presented solution in chapter 3 as well as what is not going to be implemented. Following that section, the chosen technologies are presented and why they were preferred over others. Next, in section 4.3, the implemented architecture is presented and compared to the one provided on chapter 3. On section 4.4 and 4.5, a description of all implemented components is provided, explaining their implementation and their role on the system.

The next sections on this chapter intend to go in depth at some important aspects of the system, starting with the interactions between the HGW and the SGH cloud (section 4.6). The next sections go over details on specific parts of the system that are relevant to the understanding of the overall solution.

Lastly, the final section of this chapter (section 4.9) goes over the deployment scenario of this solution that was performed in the Instituto de Telecomunicações (IT) network in order to validate it.

4.1 OBJECTIVES

Although the implementation of the HGW along with the management system would be optimal for validation and testing, it would be unrealistic given the time of this dissertation and the amount of work it would require. Due to this, the goal was to have a functional system capable of a small set of features that could be easily extended in future work. Having said this, the main vision was to have a HGW capable of controlling home devices which was securely accessible from the Internet. Additional management features, namely a safe update system, was also an objective for this work.

Knowing this, and looking at the architecture presented in chapter 3, the component for the administrator dashboard as well as the features in the main API to support it were not implemented for the scope of this work. Despite this, all the features mentioned in the previous paragraph were accomplished and are detailed in the continuation of this chapter.

4.2 ADOPTED TECHNOLOGIES

In this section, existing solutions for some system components will be discussed, as well as the choice that was made presented. This choice will be supported by explaining the different features of the existing solutions and why one was more fitting than the others for the purpose. Nevertheless the choices made in this work are not in any way absolute and definitely other approaches could have been taken. With this in mind, the work done in this dissertation tend not to force itself to any of the below existing technologies allowing for any swap provided the necessary changes.

Certificate Authority (CA)

This project required the use of a Certificate Authority (CA) in order to automatically generate certificates for its servers and more importantly the gateways. Additionally, it would be a requirement that this CA would need to provide an API to allow the automated deploy of new gateways. Having this in mind and after research, multiple softwares were found and analyzed namely:

- OpenSSL
- OpenCA
- Dogtag Certificate System
- XCA
- gnoMint

- r509
- EJBCA
- SimpleAuthority
- CFSSL

After an analysis on these solutions, the option SimpleAuthority was immediately discarded due to the fact that it only offers a GUI interface. From there options such as OpenSSL, OpenCA, Dogtag Certificate System, XCA and gnoMint all offered an option for a Command Line Interface (CLI), despite some also providing a GUI option that was not ideal. Even tough a CLI interface would allow for an HTTP interface to be developed on top to provide the required functionality, additional research was performed and solutions such as r509 and CFSSL were found where both of them offer a CLI as well as a HTTP interface to directly generate and manage the Public Key Infrastructure (PKI). Lastly there was an additional choice, EJBCA that according to their own website is not the easiest to set up and is meant for large production environments.

Knowing the existing options and the project requirements, the solution chosen was the CFSSL as it is a project supported by Cloudflare and prove to serve the purpose.

Reverse Proxy

For the reverse proxy options there were two main existing solutions to consider: HAProxy and Nginx. Both of these products are very competent and capable in their roles, and any of them would fit the project. The choice in this case was Nginx as it was a more familiar product that could be used in multiple scenarios in the project other than as a reverse proxy, such as load balancing (HAProxy is also capable of such) or as the web server for the developed components.

DNS Server

Regarding the DNS server, the requirements were, the same as in the CA, that it would provide an HTTP API to enable it to be easily managed by other software on the system. Knowing this, the considered options were:

- BIND
- PowerDNS
- Dnsmasq

When looking at the given options dnsmasq was discarded first for being a solution aimed more at simpler scenarios mostly combined with an integrated DHCP server as

well. BIND on the other hand is the most used domain name system, and is very well known for its reliability and stability. However, it does not offer a native HTTP API, and even though it was found an adaptation API[74] to provide that functionality to BIND, it was not very complete. This left PowerDNS, a DNS server capable of storing its records and domains on a DB instead of plain files, with the added value of providing a complete API to allow their manipulation. This was the chosen solution as the other were not capable of offering the required features to the extent of this one.

Image Generation System

For this topic of image generation systems there were two main candidates: Yocto Project and Buildroot. Although both of them are capable of generating images, as it was seen in chapter 2 the Yocto project is much more supported in terms of layers, boards and overall software integration. Having this said, the Yocto project was the chosen solution as it was the most fitting for this project.

Image Update System

When looking at the options offered for update systems in the state of the art there are:

- SWUpdate
- Mender
- Resin
- SWUPD
- OSTree

The requirements for this component in the project were the ability to remotely deploy updates to the user's gateways as well as have the comfort given by the dual partition schema, where a device is always bootable despite a failed update.

With this in mind options such as Resin, SWUPD and OSTree were discarded for not providing the necessary mechanisms for a dual partition system. Between the two remaining options, both of them capable of providing a dual partition scheme, the choice was the Mender system. In order to make this choice, both systems were tested with an image generated used the Yocto Project, and the Mender system turned out to be easier to deploy and manage.

User Management

For this project, the authentication and authorization of users with the SGH cloud was offloaded to external entities, specifically Google and Facebook. These services allow for external applications to use their secure authentication mechanisms without having to save user credentials and running the risk of exposing them with possible security flaws.

Both platforms make use of the oauth2 protocol, and consequently it was used to communicate with them on the user management component. This protocol is explained further by the authors in [75] and [76].

Home Control and Automation System

Lastly, the choice on the Home Automation System (HAS) was an important one as all the home devices are going to be controlled by it and it will provide the user with the device management and control. Additionally it needs to provide the user the capability to create and modify automations for his home. Having this said, there were three main candidates for potential HAS:

- Home Assistant
- OpenHAB
- Domoticz

The choice in this case was not as concrete as all three solutions fulfill the project requirements. With this, the choice fell to Home Assistant for being an easily deployed and controlled platform, with an appealing look to the end user. It is a system that is rapidly growing with new features being added with each release. Additionally, due to its programming language, Python, it is easy to develop for. Additional factors such as the familiarity with both the solution as well as the programming language also contributed to the choice of this solution.

Regarding the other two solutions, Domoticz was the least appealing one due to its development being in C++ along with its UI being the least modern and not up to par with the others. OpenHab is developed in java and also fulfilled the required features by this project, but it was not chosen as the HAS for this solution.

4.3 ARCHITECTURE

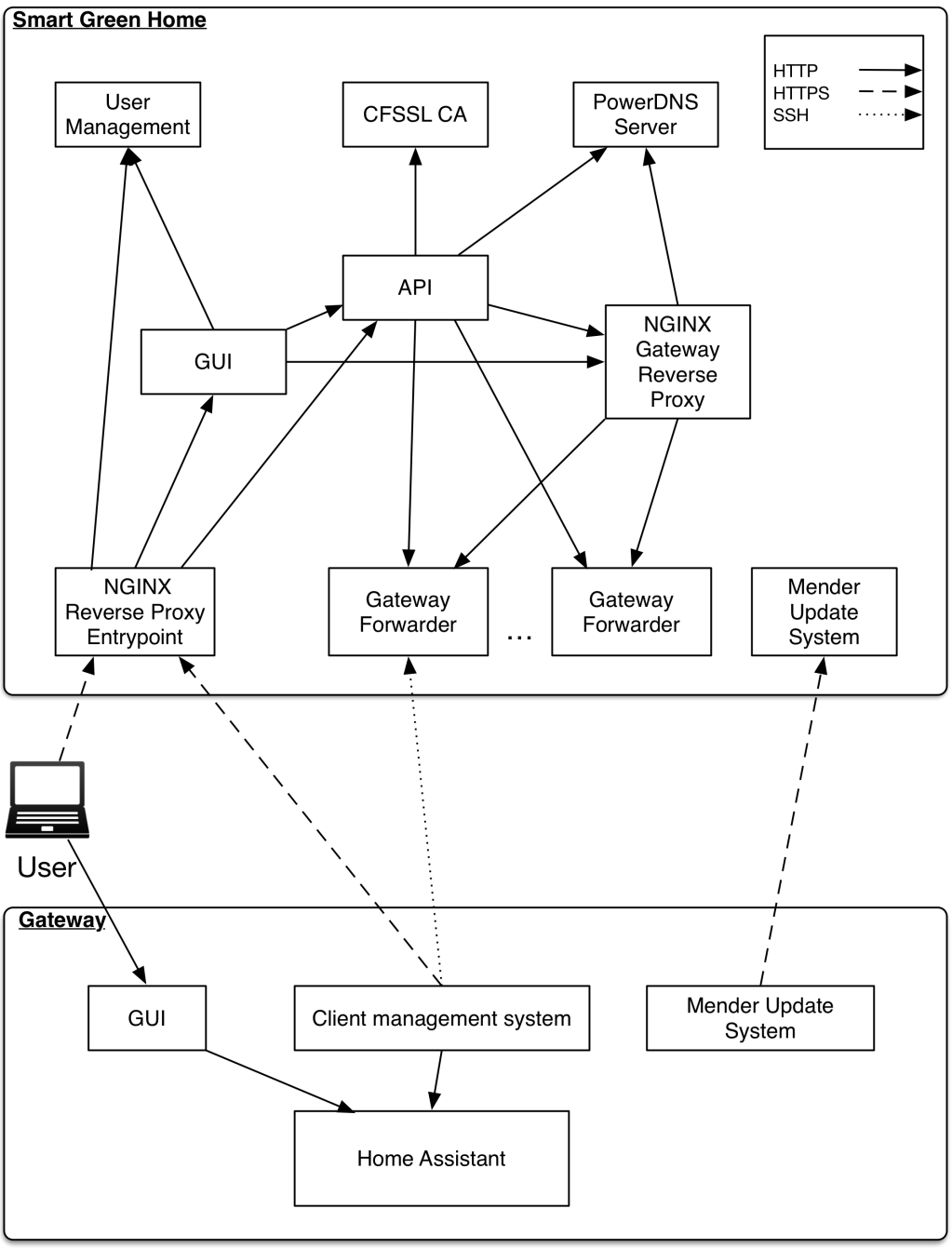

Figure 4.1: Implementation architecture.

Considering the architecture presented in chapter 3, in Figure 3.18, and the objectives and adopted technologies described in sections 4.1 and 4.2 respectively, the resulting architecture is presented in Figure 4.1 that describes the current implemented solution. When comparing this architecture to the one in Figure 3.18, it is clear the removal of the administrator dashboard as previously mentioned. Additionally, the adopted

technologies for each component are already presented where fit. Also important to mention is that all the communication between the different components is performed in HTTP when inside the SGH cloud and in HTTPS when going through the Internet. Moreover, all communication between the gateway and the SGH cloud is mutually authenticated using both client and server certificates.

Regarding the remaining components, they were implemented for the scope of this project to satisfy the necessary requirements, while allowing for an easy integration of new features in the future. The preferred programming language for these components was Python3[1] and when a DB was necessary PostgreSQL[2] was the main choice. The communication between these components was done with HTTP, as stated, and the information transfered was in JavaScript Object Notation (JSON) format.

4.4 Home Gateway (HGW)

This section will go over the components running inside the HGW providing details of how their were implemented and how they fit the full system fulfilling or helping to fulfill the use cases and requirements set for this project.

Home Assistant

Home Assistant[3] was the chosen HAS for this project as previously said. At the moment the latest version of this system is 0.54 and that was the one deployed in the HGW.

To deploy this system the procedure used was the one described in their own web site using the Python package manager (pip) to install it[77]. Additionally, a systemd script was also adapted from their website[78] allowing for easier management of the service as well as an easy way to enable its auto start on reboot or power on.

Moreover, all the requirements and use cases set in chapter 3 can be achieved by this solution, as it provides a web GUI to allow users to manually control devices, create automation rules and consult their history. Having said this, some of the requirements are not yet achieved in the most user friendly manner like adding a new device that is not automatically discovered, or creating complex automation rules.

In Figure 4.2 the main page of Home Assistant is presented, where the user can

[1] https://docs.python.org/3/
[2] https://www.postgresql.org
[3] https://home-assistant.io

control his devices and monitor their states. The page follows a simple structure where devices are organized in groups, as seen in the Figure that has the, automatically created, switch and automation groups. On the top it has the sensors that do not allow user interaction, on the example it has a luminance, a power consumption, a temperature and a motion sensors.

When going over the automations, in Figure 2 of Appendix A it is visible the web page that allows the user to create them. This is still not very user friendly mainly for two reasons, the first the user still needs to know the devices' entity id and possible states to use as a trigger or conditions as well as the command to send to the devices when it is part of the action. The second reason is that complex rules are not allowed to be created and edited on the GUI. A complex rule consists of a normal trigger and action but also multiple conditions with mixed "and" and "or" statements. To add the complex rules, the user still needs to edit the configuration file, a snippet with an example rule to turn a light on when the motion sensor detects movement is presented in snippet 6 of Appendix A. The conditions currently supported are: single conditions, or a list of multiple conditions that are internally seen as an "and" between all of them.

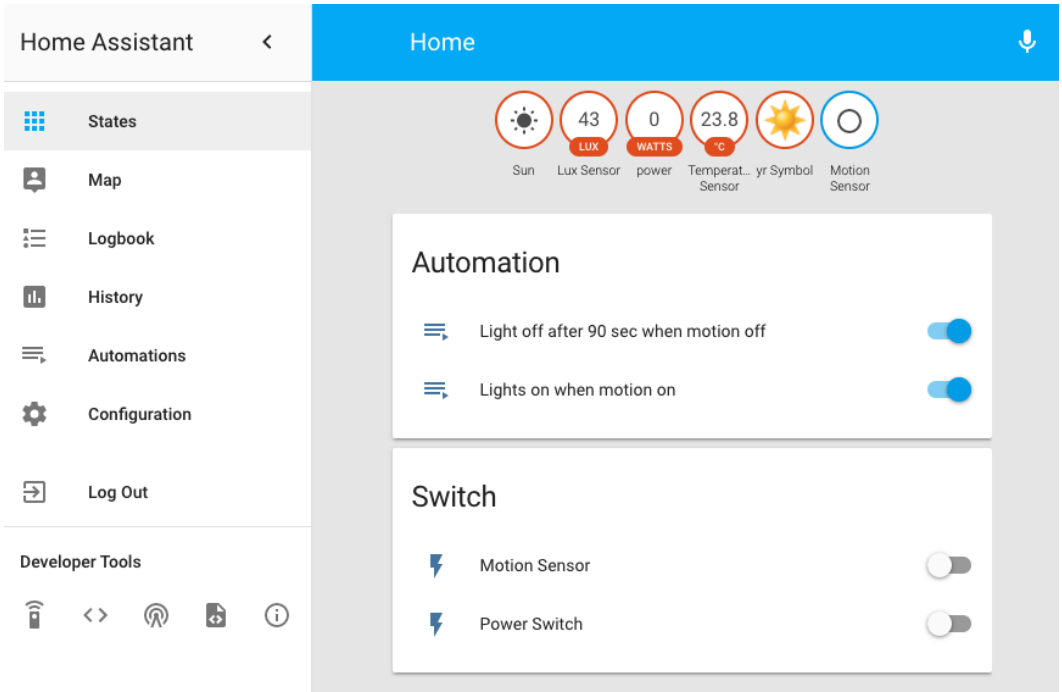

Figure 4.2: Home Assistant homepage.

This system also provides an easy solution for the user to verify the history of the sensors in his home in the web GUI as well as a log of changes to the devices. This is shown in Figure 1 and 3 of Appendix A. Still in Appendix A there is snippet 7 that

illustrates the less friendly approach to adding a device when it is not automatically discovered by the system.

Client Management Software

The software for this component was developed to be responsible for handling the management tasks and communication with the SGH cloud platform. The architecture for this system must be flexible in order to be able to support Plug-ins for future feature addition, as required in chapter 3. To assure this, a similar but simpler approach to the Home Assistant architecture was taken and depicted in Figure 4.3. Additionally, this component also has in its configuration directory files to help persist gateway details as well as its unique certificate and private key. This software runs as a Linux service using systemd to manage it.

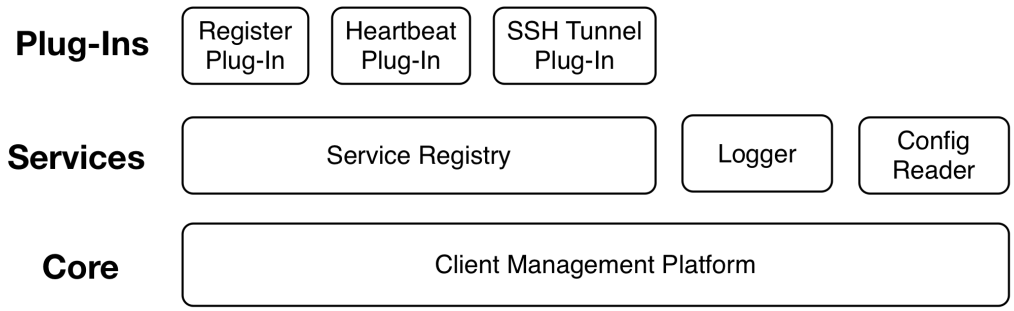

Figure 4.3: Client Management Software Architecture.

Further explaining this component, it was developed in Python3 using the *asyncio*[4] library in its core enabling it to run asynchronously. The core is also the one responsible to provide the interface methods for the Plug-ins and services to execute their tasks.

Moving to the services layer there is a logger component responsible to log the necessary events from all entities running in this system with different severity levels. This is achieved using the standard *logging*[5] library from Python. Still in the same layer there is a component responsible to read a configuration file, parse it and provide the respective configuration to each component. The configuration file is in YAML format and an example of its content is shown in snippet 8 of Appendix B. Lastly, the service registry is the third component running in this layer and is responsible for instantiating the active Plug-ins for the system. To achieve this, it reads the content of the components tag on the configuration file and executes, for each entry, a mandatory

[4]https://docs.python.org/3/library/asyncio.html
[5]https://docs.python.org/3.6/library/logging.html

method, implemented by the Plug-in, with a fixed set of arguments. This method is used for self configuration and to add the respective Plug-in to the main core asynchronous loop.

The third layer is the Plug-in layer and it is the one that gives this system its flexibility. A Plug-in is defined by a mandatory setup method and an additional class to perform its task. The initial method is used to create an object of the class and configuring it with the parameters defined in the configuration file. Finally there is an argument to the setup method that is a function used to add the plug-in class method that performs its main task to the process core execution loop. A Plug-in can also specify library dependencies that are automatically installed prior to the execution of the setup method. For the scope of this work there were developed three Plug-ins: register, heartbeat and ssh tunnel.

Regarding the register Plug-in, it is responsible for assuring the gateway is registered and validated with the SGH cloud. If it is not, it has the mechanisms to register it with the help of the GUI component in order to perform a challenge requiring user interaction, this way helping to prevent rogue gateways in the system. Further explaining of the register process, with all the involved components, will be performed in section 4.6.1.

The heartbeat Plug-in, unlike the last one that runs on the start up of the program and terminates after its task, runs for the entire time the client manager is executing. Its task is to periodically inform the SGH platform of the health status of the HGW.

Lastly, the SSH tunnel Plug-in main task is to open a secure communication channel between the HGW and the SGH cloud platform, allowing for the HAS GUI to be securely forwarded through the Internet. This Plug-in as its name implies makes use of SSH tunnels using Rivest-Shamir-Adleman (RSA) keys to keep the information flowing in them private. Additionally, this Plug-in also has to perform other minor tasks to allow this interaction, but a more detailed explanation of the entire process will be performed in section 4.6.2.

Graphical User Interface (GUI)

To ease the access of the user to the HGW, it was developed a simple GUI capable of providing minimal functionalities as well as some additional debug information. Screenshots of this web interface are available in Appendix C providing a better understanding of the implemented features. To implement this component Python3 was used along with the flask framework[6].

[6]http://flask.pocoo.org

This web interface has a simple authentication mechanism allowing to restrict user access. After the user has authenticated he is presented with the device's homepage where he can change gateway settings, register the gateway with the central platform if it is not already and access the HAS GUI.

4.5 SMART GREEN HOME CLOUD

This section will complement the previous one by going over the components running inside the SGH cloud and providing details of how their were implemented and how they fit the system fulfilling or helping to fulfill the use cases and requirements set for this project.

User Management

The user manager component is responsible for storing the user's information and authenticate them to allow access to the remaining components of this system. In order to do this while keeping safe the user's information and the risk of it leaking low, it was used the oauth2 protocol as stated in the adopted technologies section using the service from Google and Facebook.

To provide this functionality, is was developed this service in Python3 with the flask framework that would provide an interface to communicate with the external services. This service will store some user information in the DB to allow it to be used by the other components in the system. However no security details like passwords are ever used by this system. To allow this, when a user successfully logs in to the external services, an access token is generated and provided to the calling service that then passes it on the user. This token is then used by the component interacting with the user to validate its legitimacy and authorizing the request.

To fulfill the required functionality of this component it provides an API with these methods implemented:

- /**login**: Logs in a user with the help of the external services and provides the access token to the user;
- /**logout**: Logs out the user from the system invalidating that access token making it no longer valid;
- /**validate**: This endpoint is used to validate the legitimacy of an access token;
- /**user**: This endpoint is used to retrieve the user information associated with a given access token.

Graphical User Interface (GUI)

This component is where the user accesses the SGH cloud platform, providing them the functionalities the system has to offer. For the implementation done in this work only a small set of features was developed, where the user, after log in, could remotely access the gateway and register the new one. Additional information such as if the gateway is online or not is also present along with some details of it. This component was developed in Python3 with the flask framework and screenshots are shown in Appendix D.

Certificate Authority (CA)

To ensure the connected gateways are authenticated in all communication they perform, client certificates were used for this system. To be able to deploy the amount of certificates required by this decision, an internal CA was required, and so it was used, as stated in the adopted technologies, the CFSSL solution. This was deployed, with a PostgreSQL DB backend following their documentation on Github[79] and on a blog post from Cloudflare[80], on a docker container[81].

Reverse Proxy Entrypoint

This component is the main entrypoint to the system as all user and gateway interaction comes through it. This is achieved using Nginx as a reverse proxy, providing access to the previously mentioned user management and GUI as well as the main API for the system. This component also contains the server certificates for the previously mentioned services and it is the one responsible for providing the SSL layer between the users and the SGH cloud platform.

Additionally, this component is also responsible for validating and authenticating the client certificates provided by the gateways. To achieve this the requests were split inside Nginx in two locations: one for the ones originating on the gateways, containing the "/gw" prefix in the URL, and the others with the regular URL. The certificate authentication is then performed in all requests as an optional condition this way not forbidding the access to users accessing the GUI and authenticating with the user management service. When certificate authentication is indeed required, two headers with authentication details are added to be used by the destination service. These URLs containing the "/gw" prefix are also rewritten without it as their only purpose is to allow Nginx to distinguish them and add the authentication headers. An example of the relevant part for this configuration is presented in snippet 1. Additionally, in

Figure 4.4 a sequence diagram is presented explaining how the reverse proxy is able to forward the multiple incoming requests to the specific services.

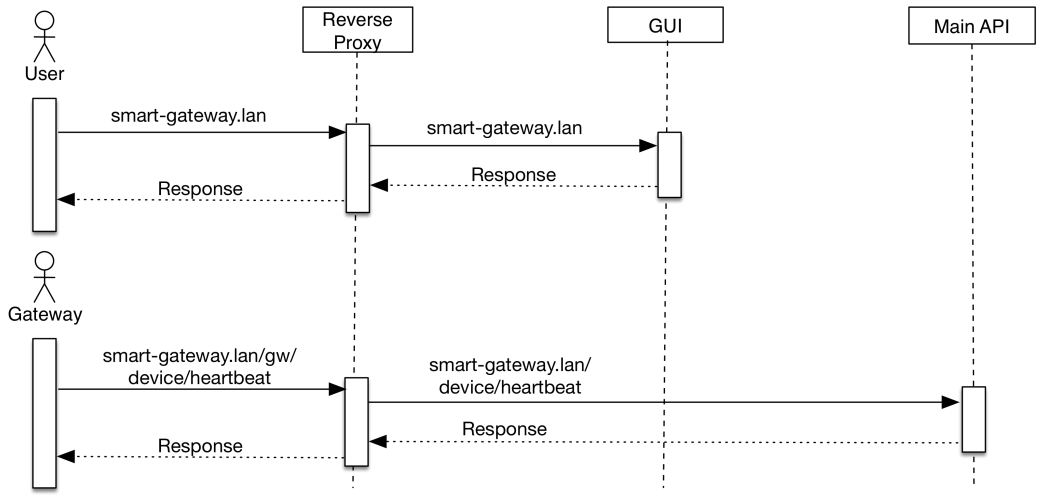

Figure 4.4: Sequence diagram for different requests to the reverse proxy.

```
server {
        listen                  443 ssl;
        server_name     smart-gateway.lan;

        ssl on;
        ssl_certificate /opt/certs/dns/certificate.pem;
        ssl_certificate_key /opt/certs/dns/private-key.key;

        ssl_client_certificate /etc/ssl/certs/ca-cert.pem;

        ssl_verify_client optional;

        ssl_verify_depth 2;

        location /gw/ {

                proxy_set_header X-SSL-User-DN $ssl_client_s_dn;
                proxy_set_header X-SSL-Authenticated $ssl_client_verify;

                proxy_pass http://localhost:5000/;
        }
        location / {
                proxy_pass http://localhost:5001;
        }
}
```

Snippet 1: Example of an Nginx configuration with client certificates.

Main API

This service is the main component of the system as most of the interactions go through it, and it is the one holding and manipulating the system's state. Most of the interaction to other services in the system is engaged by this component, namely the CA, DNS server, Gateway Forwarder and Gateway Reverse Proxy. It makes use of a PostgreSQL backend to store its information using the schema provided in Figure 4.5.

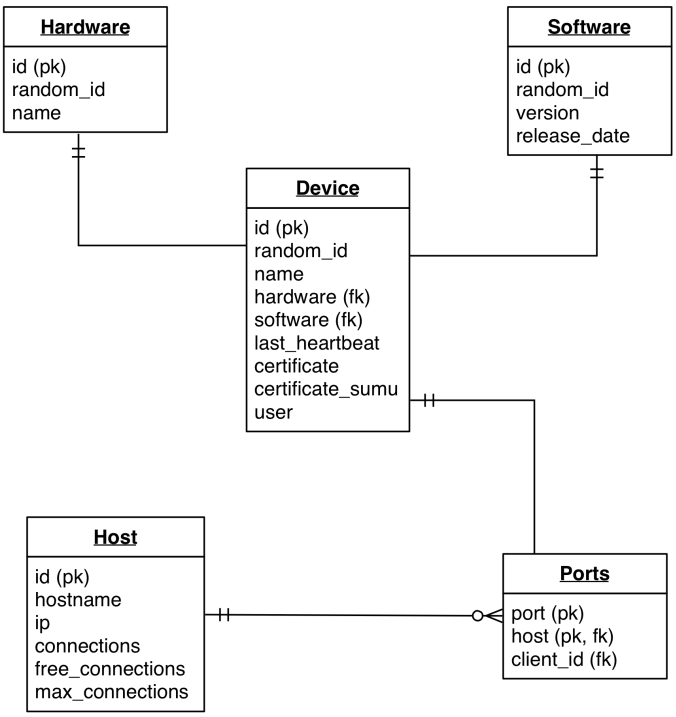

Figure 4.5: Main API DB schema.

This service was developed in Python3 using the flask framework and the following API methods were implemented:

- **/device/get-otc**: This endpoint is used by the GUI component and its function is to generate a new One Time Code (OTC) to be used in the process of registering a new gateway;

  Arguments:

  - None

  Response:

  - Generated OTC (string)
- **/device/consume-otc**: This endpoint is used by the HGW to consume a OTC previously generated. Upon success, the registry process begins where a certificate

is generated by the CA and delivered to the HGW along with other information for it to be able to identify itself with the SGH cloud;

Arguments:

- OTC (string)
- hardware_id (string)
- software_id (string)

Response:

- random_id (string)
- hardware_id (string)
- software_id (string)
- certificate (string)
- certificate_sum (string)
- user_id (string)
- private_key (string)

- **/device/check-registered**: Endpoint used for gateways to check their registration status, useful on boot for gateways to check if their registration is valid or not;

  Arguments:

  - random_id (string)

  Response:

  - random_id (string)
  - hardware_id (string)
  - software_id (string)
  - certificate (string)
  - certificate_sum (string)
  - user_id (string)

- **/device/heartbeat**: Endpoint used for gateways to periodically send a heartbeat message assuring their good functioning to the platform;

  Arguments:

  - random_id (string)

  Response:

  - None

- **/forwarder/get-port**: Endpoint used for a HGW to get a free host and port to where it can open an SSH tunnel and forward its HAS. This method configures, using an API, the DNS server, the Gateway Reverse Proxy and the Gateway Forwarder to allow the connection from the new gateway. When multiple Gateway Forwarders are present, load balancing between them is performed using a simple weighted least connected algorithm;

Arguments:

- random_id (string)

Response:

- random_id (string)
- host (string)
- port (int)

- **/forwarder/free-port**: This endpoint is used by the HGW to release the port that was previously assign to it. This method then notifies the connected components of this change and they remove the entry corresponding to this gateway from the configuration;

  Arguments:

  - random_id (string)

  Response:

  - None

- **/user/gateways**: This endpoint is used by the GUI component to allow retrieving the information about the gateways belonging to a specific user.

  Arguments:

  - user_id (string)

  Response:

  - gateway_list (list)
    - id (int)
    - name (string)
    - hardware (string)
    - software (string)
    - online (boolean)
    - url (string)

The methods implemented in this component were the required to satisfy the needs of the developed Plug-ins. However a flexible approach was taken where new methods can be easily added with minimal effort. This is consistent with the approach taken in the Client Management Software, as they are tightly related.

Moreover, to support the implementation, additional utilities had to be developed namely a decorator to ensure the validity of the certificate provided by the gateway. This is done using an header added by the Nginx entrypoint which is then validated in each method where certificate authentication is required.

Lastly, to support horizontal scalability of this component, as it is a central part of the system and it will receive most of the requests, the calls to its API had to remain

stateless. On the implemented methods this is true for almost all of them as all the information needed is persisted into a DB and synchronization is achieved that way. The exception is the OTC, where for performance and logical reasons storing it in the DB was not an option. For this a redis[7] server with no persistence was deployed allowing for multiple instances of this service to sync their state across multiple calls to the API. For the implemented features this was only used to sync the OTC from when the user retrieved it in the GUI component and the gateway used it to register itself.

DNS Server

To provide access to all gateways using DNS it was clear that a DNS server would be necessary in the SGH infrastructure. As explained in section 4.2, PowerDNS was the chosen solution, and so it was deployed using a PostgreSQL DB following the instructions on their official documentation [82] and [83].

Gateway Reverse Proxy

This component is similar to the Reverse Proxy Entrypoint, in the sense that it also uses an Nginx server as a reverse proxy but in this case it allows to reach the gateways using their unique DNS name. The configuration for each gateway is simple as it only needs to forward the request to the correct Gateway Forwarder. An example entry can be seen in snippet 2.

```
server {
  listen          80;
  server_name     d8669b7907ccc6b0065d342408a7babc4b1cb839.smart-gateway.lan;

  location / {
        proxy_pass http://forwarder-1.smart-gateway.lan:43995;
  }

  location /api/websocket {
        proxy_pass http://forwarder-1.smart-gateway.lan:43995/api/websocket;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
  }
}
```

Snippet 2: Example of a gateway entry in the Nginx configuration.

Along side the Nginx server it was developed a simple API using Python3 and the

[7] https://redis.io

flask framework that would allow the remote modification of the Nginx configuration file. It provides only two endpoints: one to add a new entry and one to remove an entry.

This software runs as a Linux daemon and it was created an installation script that handles the install process including security aspects such as the creation of a dedicated user for this daemon with access only to the folder containing the code and the Nginx configuration file. Additionally, this daemon would need to be able to reload Nginx for it to load the new configuration. To solve this, without giving the limited user sudo access, it is created, during the installation, a file in the "/etc/sudoers.d/" directory with the entry seen in snippet 3. This gives the user sudo permission only to that command this way limiting his access on the host machine.

```
proxy-api ALL=(ALL) NOPASSWD: /usr/sbin/service nginx reload
```

Snippet 3: Content of the file in the "/etc/sudoers.d/" directory.

Gateway Forwarder

This component uses an SSH server with its port forwarding functionality allowing gateways to forward their HAS GUI over the Internet while benefiting from the well known security present in the protocol. To allow this functionality, gateways need to be able to log in this server securely. To achieve this it was used the already present keys, exchanged during the registry process with the certificate. The gateway already had the necessary private key to use in the ssh connection. As for this component it extracts the public key present in the certificate and converts it to the RSA format, inserting it in the "authorized_keys" file. This file contains the public keys for the clients that are allowed to connect, along with some additional settings, an example of it is present in snipper 4. The first two parameters "no-pty" and "no-X11-forwarding" forbids the connection from creating any interaction with the server being by allocating a shell or a graphical environment. The next parameter is "command="""" this avoids the last way that a potential attacker could have to execute commands that was sending the command directly as an argument of the SSH connection not requiring the allocation of a shell and thus bypassing the "no-pty" option. Following, there is the gateway public key, and lastly, where a comment can be placed in the file, there is the gateway Unique Identifier (UID) to allow the easy finding and removal of this entry should it be necessary.

```
no-pty,no-X11-forwarding,command="" ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQClfRpi
EUm+GCg9hG7W7UyvWzFHS5ggbxfasvgRBHjHsG8FpROVpT2EoVeR93RlZ1SucMcdfZO0OnRZ5m1Ykvt8
VnMT5IE+HQv6OSeoIMTMGO/NY1w8187hvEE59wGujqhApIDqRMy8tX/E32HMjLIlxhamj+KVjaYSknU/
jd9l1oKklfNJi9GSDQWsnW1TEHybLCEaJCRB0YPqf4UrlXxA61MjVNbZr8T+coP9qVjpzrEgBpBNECXQ
yTQ5bp45xXZ7NPzLOPPVtdhSRSZj6geNlGTRdnUmzhrf8Em7C58GlJtaSoO/E6lHUK4P+Q/APu5tuRmB
7OIXQTY+dmAlGeBR CNG6KI5TBBHIAZEQH54WUSRQKFYH6YWY3VNM6VNHBWGTGZNJKWJ4CZJZOH3QAKY
W6XFCATFTDAFYUMC
```

Snippet 4: Example of the "authorized_keys" file.

These tasks are performed by a Linux daemon developed for this purpose as well as provide an API allowing the easy addition and removal of entries in the SSH server. This software was developed using Python3 and the flask framework, and in addition an installation script is present containing all the necessary operations including the creation of a dedicated user with privileges only to the code folder and the "authorized_keys" file. Additionally, similarly to the previously mentioned service(Gateway Reverse Proxy) it also needed an entry in a file in the "/etc/sudoers.d/" directory in order to allow it to reload the ssh server with the new configurations.

Update System Server

In order to keep the gateways updated, an update system had to be used, and as stated in the adopted technologies (section 4.2), the chosen solution was Mender. This system was deployed using the official documentation [84] which was also used to help in the image generation for this system, and discussed further ahead in section 4.7.

This solution provides a web GUI allowing an easy upload of newly created images to deploy new versions to the production gateways. Additionally an overview of the connected devices is also present with some information about them. From here manual updates could be triggered as well as group updates, the latter reaching multiple gateways at once. This group functionality is very important in a large scale deployment as it is not ideal to update all devices at once due to the constraint imposed on the network and the update system itself, but updating one gateway at a time is also not desirable. Additionally different hardware devices can have different update images targeting only their respective groups.

Screenshots of this system illustrating its features can be found in Appendix E.

4.6 SYSTEM INTERACTIONS

In this section the most important interactions of the system will be presented. This will show in detail the way they were implemented and their flow, demonstrating how the multiple system's services communicate to achieve the task.

4.6.1 GATEWAY REGISTER

To register the gateway in the system a two step operation is required, this way allowing it to automatically associate to a user account as well as reduce the possibility of rogue gateways registering in the system.

To further explain this process, the sequence diagram in Figure 4.6 was created to illustrate the register process. When analyzing the diagram, the first interaction has to be started by the user accessing the SGH web GUI and, after having logged in, selecting the "Register new Gateway" option, as illustrated in Figure 11 of Appendix D. This will provide the user with a OTC that he should then use on the HGW GUI to continue the registration process. When the user inserts the OTC in the HGW this triggers it to check its validity with the main API and upon success the gateway is added to the system and the information relevant to it returned. This information includes the gateway certificate and private key that it should store to authorize itself in all further communication. This is the only step where the private key is transfered over the Internet and it is secured by the encryption of Hyper Text Transfer Protocol Secure (HTTPS) of the server. The private key is never stored by the server, and it has no further access to it after this interaction. As for the HGW, when it receives this information it process it and saves it into different files for further use in its tasks.

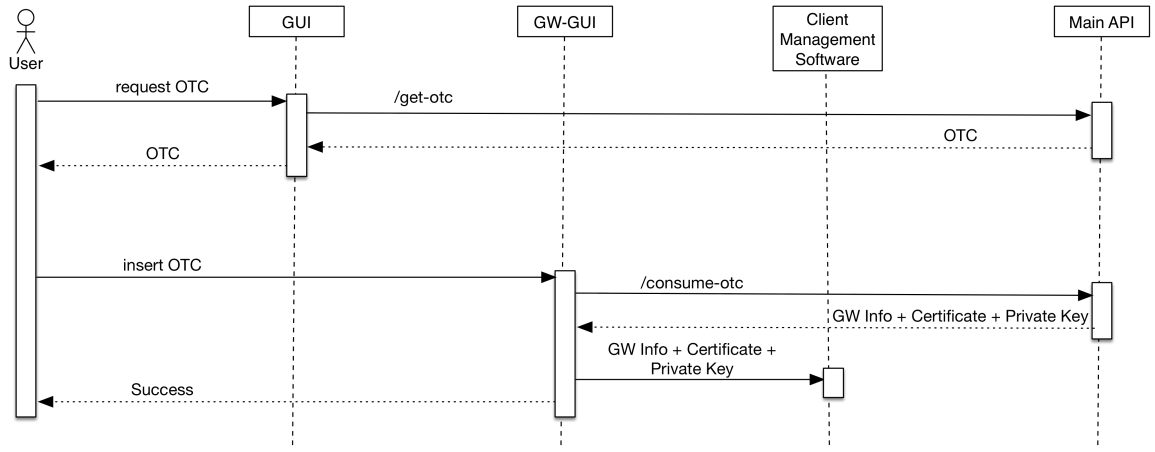

Figure 4.6: Sequence diagram for the registration process.

Additionally, running in the HGW, allowing the registry process, is the register Plug-in whose flow is illustrated in Figure 4.7. When this Plug-in is executed it checks for a device file, that contains information regarding the gateway registration. If this file exists, it checks it for the device UID and then verifies the registration status with the SGH platform. If the server validates it, the registration process is complete and the Plug-in terminates. On the contrary, if this validation fails or the device file does not exist, the Plug-in opens a UDP socket binded to localhost only, where it waits for a registration message coming from the HGW GUI component. Upon receiving a message it processes it, and given no parsing errors, the necessary files are created and the registry process is also given as complete.

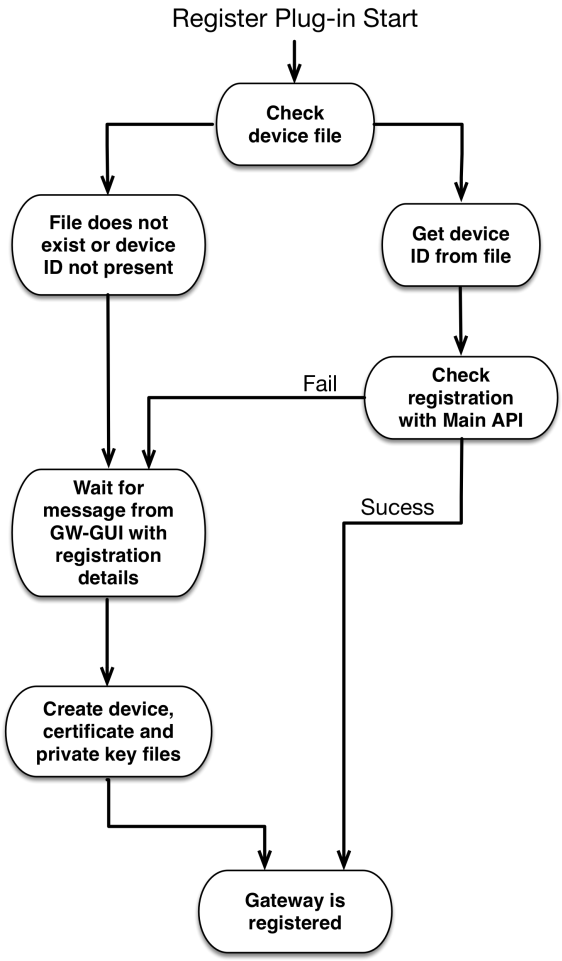

Figure 4.7: Flow diagram for the register Plug-in.

4.6.2 ACCESS THE GATEWAY FROM THE INTERNET

To allow the remote access to the HAS, a secure communication channel has to be established between the HGW and the SGH cloud. As it was previously mentioned, to do this it was used an SSH tunnel, this way taking advantage of the well known and secure protocol.

In Figure 4.8 there is a sequence diagram of the process of opening the SSH tunnel. Analyzing the diagram it is clear that in order to open the SSH tunnel, some configuration has to be performed in multiple services of the SGH cloud in order to achieve this.

The first step of this process is started in the HGW by the SSH Plug-in, where it requests from the SGH platform a host and port to where it should establish an SSH connection.

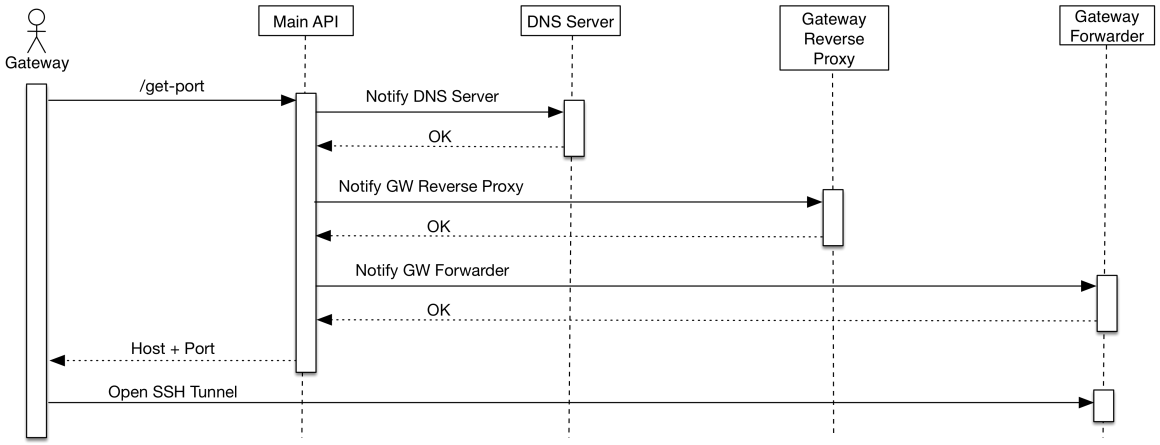

Figure 4.8: Sequence diagram for the process of opening the SSH tunnel.

On the SGH platform, upon receiving a request from a HGW, a set of tasks are performed that are illustrated by the flow in Figure 4.9. The first tasks the API does is to check in the DB if the requesting HGW already has a port assigned to it. If it has the same host and port are returned to the gateway. If this is not true, then the API finds the host with the most available connections, using the DB shown in Figure 4.5 with the "free_connections" field of the "Host" table. After selecting the host, a random unused port from the range of 1024-65535 is chosen. At this point the API has already selected where the gateway will connect, but for this connection to be possible some additional configuration has to be performed.

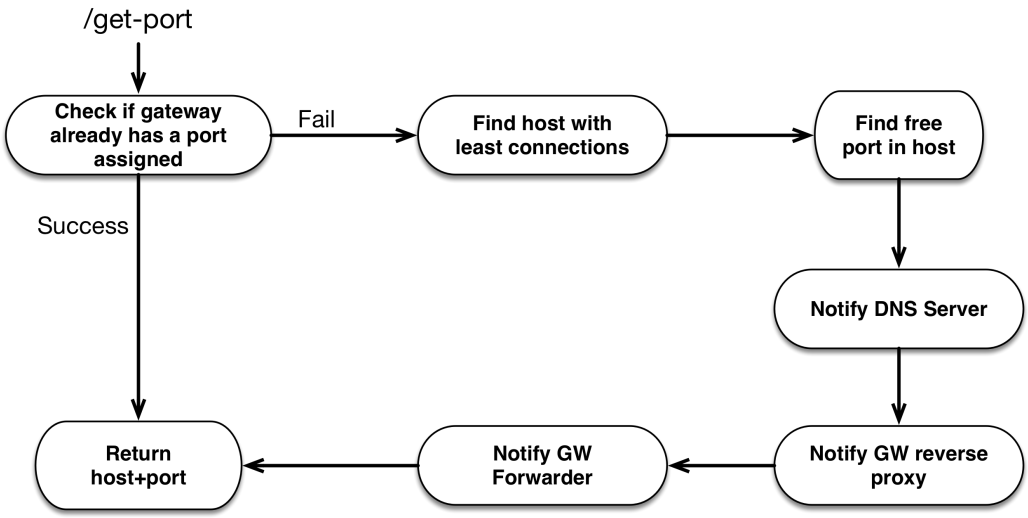

Figure 4.9: Flow diagram for the /get-port API method.

The next steps, still within the same API method, are to notify the DNS server, the HGW reverse proxy and the HGW forwarder for them to perform the necessary configurations. This configuration is performed using the built in or developed APIs running along side these services and described in section 4.5.

To finalize the process depicted in Figure 4.8 the HGW, after receiving the host and port, must connect to them using an SSH tunnel. In order to do this, the Python Paramiko library[8] was considered and tested, but, even though it opened the SSH connection, it was unsuccessful in forwarding the necessary port to the server. The solution was to develop a very simple Python library that could open and close an SSH connection and forward the port to the server. In order to achieve this, and due to the fact that this solution was working using an SSH tunnel opened manually with the CLI, the Python subprocess library[9] was used to replicate the CLI command with configurable arguments. The SSH command used for this scenario is presented in snippet 5. Providing a brief explanation of the command, the first argument specifies the private key to use when connecting to the server. This key is also the one used with the certificate given at the registration. The next argument. "-N", is used to prevent command execution on this connection being only useful for port forwarding. The following argument "-R" is used to specify the port forward. On the given example port 50064 of the server is being binded to the localhost port 8123. This is the default port for the Home Assistant, giving this way access to it from the server. The last parameter is the username and host that the gateway is connecting to, being in this

[8]http://www.paramiko.org
[9]https://docs.python.org/3/library/subprocess.html

example the "forwarder-1.smart-gateway.lan" host and the user "ssh-tunnels" that is the default user for the HGW forwarders servers.

```
ssh -i /home/pi/sg_config/certs/private-key.pem -N -R 50064:localhost:8123
       ssh-tunnels@forwarder-1.smart-gateway.lan
```

Snippet 5: SSH command to forward a local port to a server.

After this process is complete the HGW is reachable from the Internet. The interaction when accessing it is shown in Figure 4.10 by using a sequence diagram to illustrate the multiple components involved in this process. When analyzing the interaction, on the first step the user must connect to the SGH cloud platform and authenticate on the system. After the user is presented with the SGH web GUI shown in Figure 10 of Appendix D, he must chose the gateway he intends to access and click the "Go" button. He is then redirected to a URL in the format "<gw_id>.smart-gateway.lan" that leads to the HGW reverse proxy that according to the used DNS name, redirects him to the respective HGW forwarder and port. Knowing that the port on this server is being used to forward the user's HAS GUI he is presented with it remotely over the Internet.

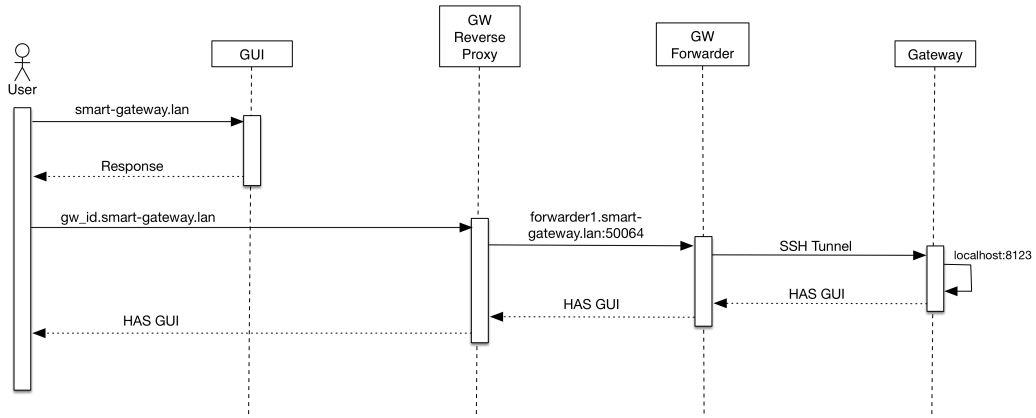

Figure 4.10: Sequence diagram for the process of accessing the gateway remotely.

Furthermore, it is important to mention that in a first approach to this solution a DNS server was not part of the presented architecture. In this case the access to the HGW instead of being done in the format "<gw_id>.smart-gateway.lan" was done in the format "smart-gateway.lan/<gw_id>". This approach simplified the final solution by not having a DNS server and thus not having to maintaining it and keeping it always in sync with the rest of the system. However, the main condition that kept this solution from being used, was that the URL that was being used to access the HAS was

not the root path ("/") but rather a path containing the HGW UID in the beginning ("/<gw_id>/"). This lead to most of the files returning a 404 error, due to Home Assistant being hard coded to use the root path. This is a known issue discussed on their forums[10] that at the time of implementation was not yet solved.

In order to solve the issue discussed in the previous paragraph, the DNS server was introduced, and with it some other problems arose. The main problem was that the HGW UID plus the domain part was to big for browsers to resolve the DNS name. This with the addition of exposing the HGW UID in every DNS query lead to the solution of hashing the HGW UID and using that for the DNS name. This hash process was done using the Secure Hash Algorithm 1 (SHA-1) that provided shorter fixed size DNS names while protecting the HGW UID value.

Finally, to mention that when a gateway gracefully disconnects, it calls the "/free-port" API method that does the reverse process of the "/get-port" method in the sense that it frees the allocated resources from the multiple intervening services.

4.7 IMAGE GENERATION

In this section the process of generating an embedded image to use on the HGW is described. As stated in section 4.2, the chosen solution for this part of the project was the Yocto project. To help with the understanding of this system, some tutorials were followed[85][86][87], but some self learning was also involved as, even though layers for this system are fairly easily found, documentation on how to use them and fixes for errors that appear is scarce. This solution is based on layers that use recipes for automatically download, compile and install the software they are design to provide. For this project the following layers were used :

- meta-poky
- meta-yocto-bsp
- meta-raspberrypi
- meta-golang
- oe-meta-go
- meta-mender-core
- meta-mender-raspberrypi
- meta-mender-demo

[10] https://community.home-assistant.io/t/relative-path-for-home-assistant/2469/9

The first two layers are from the Yocto project, the meta-poky provides the packages for a complete minimal system and the meta-yocto-bsp allows for custom board support to be included in the project. The next layer, meta-raspberrypi, is self explanatory in the sense it adds support for the raspberry pi board. The following two layers, meta-golang and oe-meta-go, provide support for the GO programming language. This is required for the Mender update system client that is installed and running in the generated image. This software is provided by the three last layers, each with its purpose.

Additionally, on a later part of this dissertation, an additional layer containing recipes to install Home Assistant was found but not tested[11]. Even with this layer, for the system to be fully functional, additional software, developed in the scope of this dissertation, needed to be installed and thus a custom layer needed to be developed for that purpose. This was not the main focus of this work and thus only a simple image was generated for testing purposes.

Having described the image generation process, the result was an image containing a simple Linux OS with minimal functionality partitioned according to the Mender system requirements. The Mender client is also installed and connects to the Mender server to allow remote updates. This system contains 4 partitions:

- boot
- rootfsA
- rootfsB
- data

The boot partition makes use of the u-boot bootloader[12] to perform the boot process. The u-boot is also responsible to chose between the rootfsA and rootfsB partitions to boot to. In normal circumstances the bootloader will always chose the rootfsA and only on boot failure scenario will he switch to the other one. This process ensures always a bootable device even across failed updates. Lastly the data partition is responsible for keeping the persistent data across updates as it will not be overwritten.

In conclusion, the process of image generation, and safe updates was demonstrated to work in this project, nevertheless additional work should be performed for full functionality of the generated image.

[11] https://github.com/bachp/meta-homeassistant
[12] http://linux-sunxi.org/U-Boot

4.8 HOME ASSISTANT PLUG-IN FOR RF PLUGS

Although the focus of this dissertation was the development of the SGH cloud platform and the HGW software to interact with it, having a physical device to control from the HAS brings a new level of validation to the work done.

With this in mind, and additionally to the devices controlled with existing Plug-ins from home assistant discussed in the scenarios in chapter 5, a new Plug-in for Home Assistant was developed.

This Plug-in was developed to control the DI-O smart plugs[13] using Home Assistant. These plugs use a RF protocol working on the 433Mhz band known as Home Easy and described in [88]. This is not a well documented protocol, and thus some trial and error was required to achieve a working solution.

To implement this, some additional hardware was required namely an Arduino UNO[14] and an RF receiver and transmitter. Moreover, to aid the process of reading the original signals sent by the official remote and sending replicated ones, two existing Arduino libraries were used: rc-switch[15] and NewRemoteSwitch[16]. The former was adapted with a new protocol option that allowed the sending of the data with the specification of the Manchester encoding[17], used by the Home Easy protocol. The latter was used to read the original remote codes and no modification had to be done.

Regarding the Arduino component of this task, the modified rc-switch library was bundled with a main execution loop for the Arduino to receive the byte code over the serial interface and send it over the RF transmitter with the respective pulse length and timings. This Arduino firmware is available as an open-source Github repository[18].

Complementary to the firmware, a Python library had to be developed to interact with the connected Arduino. This library was developed in a generic way, allowing to implement multiple protocols. However, for the scope of this work, only the Home Easy protocol was implemented. To achieved this a class was developed where all parameters to send are customizable and has a method to output a byte code to be sent to the Arduino. This library is also on a Github repository[19] and hosted on PyPi[20].

[13] https://www.chacon.be/en/smart-home/63-set-of-3-dio-remote-controlled-plugs-5411478547952.html
[14] http://www.arduino.org/products/boards/arduino-uno
[15] https://github.com/sui77/rc-switch
[16] https://github.com/1technophile/NewRemoteSwitch
[17] http://www.erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html
[18] https://github.com/bsilvr/RF433ArduinoFirmware
[19] https://github.com/bsilvr/pyRF433
[20] https://pypi.python.org/pypi

The last piece of this system is the Home Assistant component, that was developed following the project's guidelines to ensure compatibility and future approval of the Plug-in. This was implemented as a generic switch, and it is as such that it is presented in Home Assistant. An example of the required configuration for this component is present in snippet 9 of Appendix F, along with a working example of a switch in Home Assistant using this custom component(Figure 15). This component is also hosted on a Github fork[21] of the official Home Assistant repository.

4.9 DEPLOYMENT SCENARIO

This section will focus on the way the components described in Figure 4.1 were deployed in this project. This is not to be mistaken for the test scenarios presented in chapter 5, as this will only describe how and where the components were installed.

Regarding the HGW deployment, it was used a Raspberry Pi 3[22]. On this system the three software components, all except the Mender client, shown in Figure 4.1 are running with no isolation between them other then each using its own user, having permission only to the necessary files and folders.

When looking at the SGH cloud platform there were many more components to deploy. As this system was developed to allow scalability, no dependencies could be created between components other than the network. Knowing this, the deployment of the SGH cloud was done on the IT infrastructure in an hybrid between docker containers and VMs.

Looking at the components deployed with docker, firstly, all the DBs for this project are containers: the DNS, the CA, the API and the user management. Additionally, the CA software is also deployed in a container along with the user management, the GUI and the main API. For these components a docker file was created to allow the container creation. A final container was created to host the redis server used by the main API.

Even though a full docker deployment was possible, to simplify the process, on components that require file changes due to live configuration updates, Virtual Machine (VM) were used, this way allowing to run multiple services and having direct access to the files. This is evident in components such as the HGW reverse proxy and the HGW forwarder, where additionally to the main program there is an extra daemon running

[21]https://github.com/bsilvr/home-assistant
[22]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

with an API to allow remote changes to the configuration.

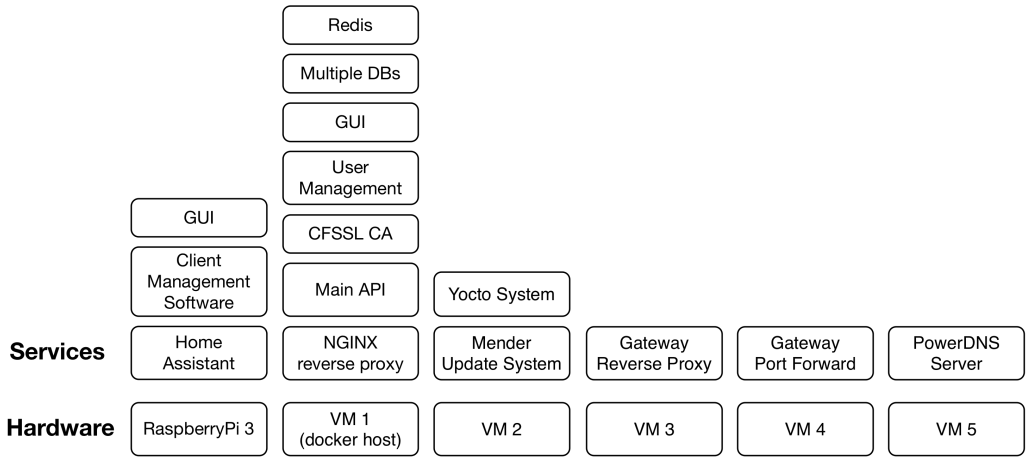

Figure 4.11: Deployment diagram of the used infrastructure and respective services.

To facilitate the explaining of the deployment process, Figure 4.11 shows a representation of the infrastructure used and the respective services running on each.

chapter 5

Evaluation and Results

This chapter evaluates the developed solution previously addressed in chapter 4. This is an important step to the project as it allows to validate both the architecture as well as the work done in this dissertation. Furthermore some metrics of the system are analyzed in order to achieve a performance evaluation of the system.

The first section of this chapter starts by presenting the test scenario used for the the remaining of this chapter. Following, section 5.2 validates the solution by presenting its features in comparison to the requirements defined in chapter 3. The third section of this chapter is dedicated to the performance metrics of this system and the tests that were realized to achieve them. The last section of this chapter goes over some analytic tests that were perform, in order to predict some metrics on a scaled out deployment of this system.

5.1 TEST SCENARIO

This section goes over the final implemented scenario, in the IT infrastructure, used to validate the solution of the Smart Green Home (SGH) project. To help understand the hardware used and how it interacts, Figure 5.1 was created. To point out that, when analyzing the Figure, the deployment inside the SGH Cloud is described in Figure 4.11.

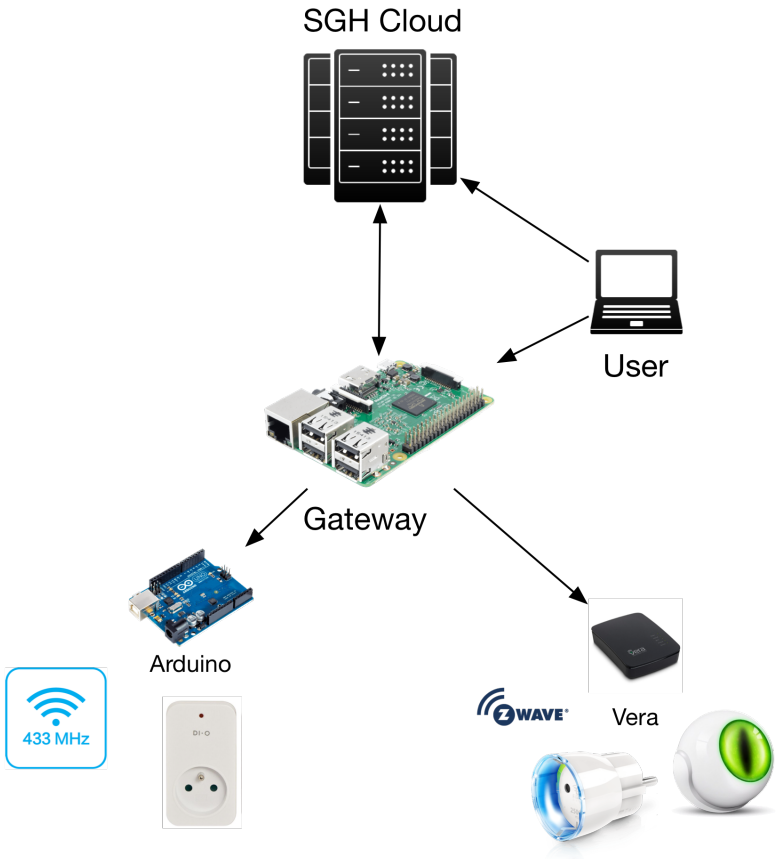

Figure 5.1: Test scenario used to validate the solution.

Looking at Figure 5.1, there is a Raspberry Pi 3 acting as a gateway, as previously mentioned. However, additional hardware is present that was used for testing purposes. Firstly, a Vera gateway[1] was used to provide the gateway with Z-Wave capabilities. Connected to the Vera there were two devices: a motion sensor and a wall plug. The other device connected to the gateway was an Arduino using the custom firmware mentioned in section 4.8. Connected to the Arduino via RF 433 Mhz was a wall plug from DI-O[2].

[1]http://getvera.com/controllers/veraedge/
[2]https://getdio.com/en/smart-lights/17-smart-remote-controlled-plugs-5411478547952.html

This scenario allowed to demonstrate the system working as it is demonstrated in the next section.

5.2 SOLUTION VALIDATION

In this section, the end result of this work is presented and its validity is demonstrated by fulfilling the defined requirements for the project. Looking at Appendix A, where Home Assistant is shown, it is visible that the solution meets the necessary requirements as also stated in section 4.4 in the Home Assistant section. Moreover, in Figure 5.2 the HAS is shown running in the HGW being accessed remotely, as visible by the URL. This URL, as said before, is the result of hashing the gateway ID in the system.

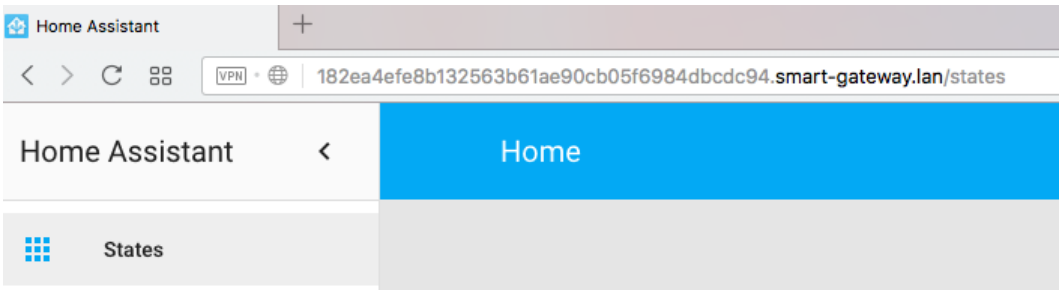

Figure 5.2: Home Assistant URL when accessing externally.

The functionality of the SGH cloud is shown in Appendix D as the screenshots demonstrate the implemented features.

Additionally, figure 5.3 shows a photo of the hardware used and how it all connected to test the solution and help validate it. This hardware was also used in the demonstrations of the project stated in section 1.3. With this hardware an automation was created to turn on a light connected to a wall plug when motion was detected and turn it off after a timeout. The system's state could then be monitored remotely from the web interface.

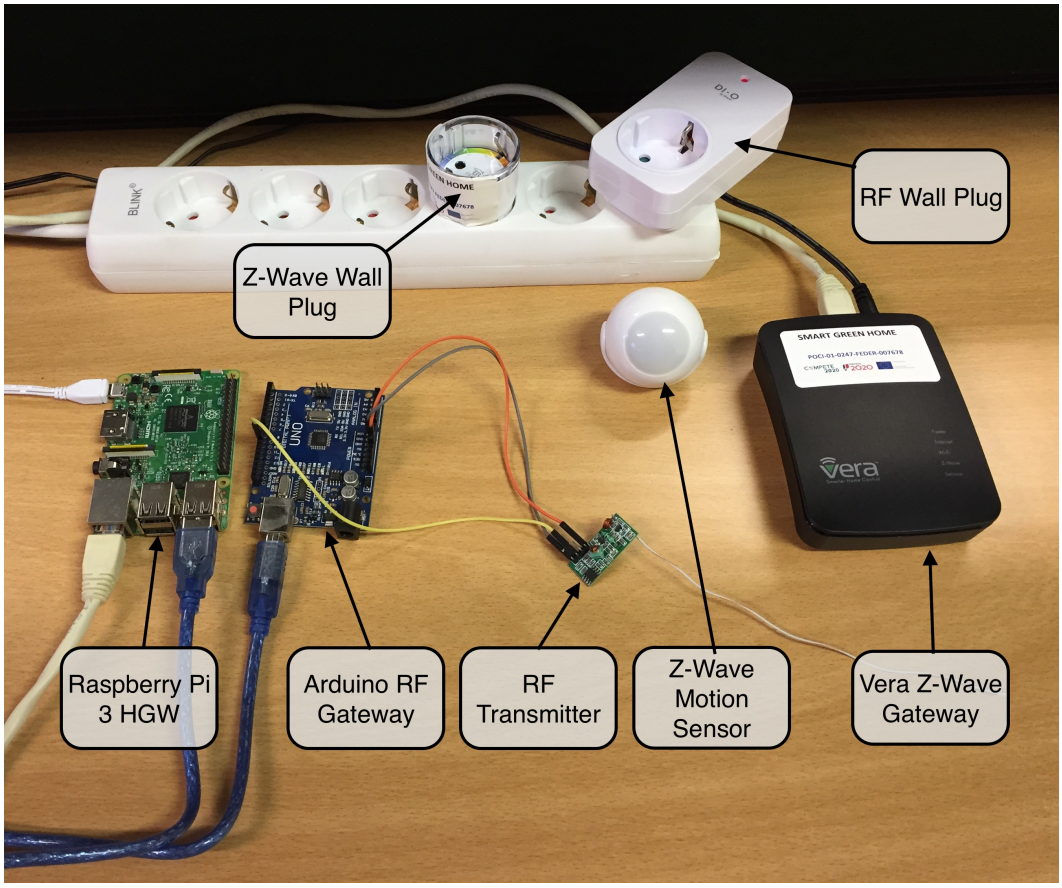

Figure 5.3: Hardware used to validate the presented solution.

5.3 PERFORMANCE RESULTS

In this section, a description of the performance tests done on the platform will be presented and their results discussed. Two main tests will be done, the first being a request-response time to both the Main API as well as the GUI component of the SGH Cloud. The second test is a comparison of the access time to the HAS front-end with the connections originating from both the local network as well as accessing it over the developed SGH Cloud platform.

5.3.1 METHODOLOGY

To conduct these tests a tool from Apache, named ApacheBench (AB)[3] was used. This is a reference tool when carrying out performance tests to HTTP servers, providing good metrics and insight with the results. Given these reasons this is was the chosen tool to conduct these studies.

[3]https://httpd.apache.org/docs/2.4/programs/ab.html

For all the tests done, AB was configured to send 1000 requests to the desired URL. Moreover, three different concurrency levels were used: 1, 10 and 100 concurrent requests. It is important to mention that for concurrency levels greater than one, the AB tool gives the results for all concurrent requests. To further explain, given a 10 concurrency level, the average request time given by AB is for the response of all 10 requests and not a single one. Additionally five runs of each test and of each concurrency level was executed and from there the present results were obtained.

5.3.2 SGH cloud response time

This test, as stated, measures the response time of the SGH Cloud in two developed components, the Main API and the GUI components. These components were chosen due to being developed solutions and not existing ones as well as being services that are part of the core of the system and thus will handle a large part of its requests.

On this test, the origin of the requests was on a different network from the SGH Cloud, but still inside the IT building. Additionally, the request directed at the API was an heartbeat example where no additional services dependencies are present. In the same way, the request to the GUI component was directed at the homepage, having this way no external dependencies as well.

The results presented here will compare the request time of both the API and GUI component. These results are presented in Table 5.1 where it shows the metrics for multiple concurrency levels. A final column showing the average per request is presented, this shows the average time per single request and not for the aggregate of concurrent requests. All values presented are in milliseconds and rounded to the unit. This provides an easier reading and given the small scale, it does not influence the results.

When looking at Table 5.1 and analyzing the results, it is visible that the API request times are consistently below the GUI ones. Seeing that the request path to reach the service is similar, going through the same networks and the same reverse proxy, the justification for this is the response size. While the API responds with a JSON message confirming the heartbeat, the GUI has to send the Hyper Text Markup Language (HTML) page along with its JavaScript (JS) and Cascading Style Sheets (CSS).

Nevertheless, the response time for both these services is acceptable, being below 20 milliseconds in all tests, and even below half that value with higher concurrency

	Minimum (ms)	Maximum (ms)	Average (ms)	Standard Deviation (ms)	Average per request (ms)
API (Concurrency 1)	14	42	16	2	16
GUI (Concurrency 1)	15	45	18	2	18
API (Concurrency 10)	41	200	81	21	8
GUI (Concurrency 10)	44	143	82	17	8
API (Concurrency 100)	177	1143	632	187	6
GUI (Concurrency 100)	227	1241	765	188	8

Table 5.1: Comparison of API and GUI request time for different concurrencies.

levels. These lower values are due to internal caching done by the web server when a large amount of identical requests are executed. The deployment of both of these services was done in docker containers running the default flask development server. The deployment with high performance web servers such as Nginx, could improve these times even further.

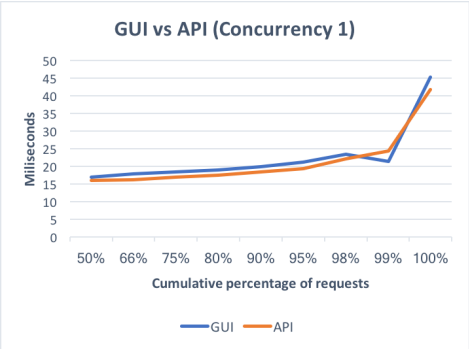

Figure 5.4: Comparison of the cumulative percentage of requests for concurrency 1.

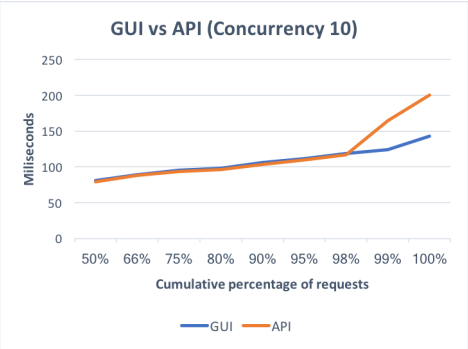

Figure 5.5: Comparison of the cumulative percentage of requests for concurrency 10.

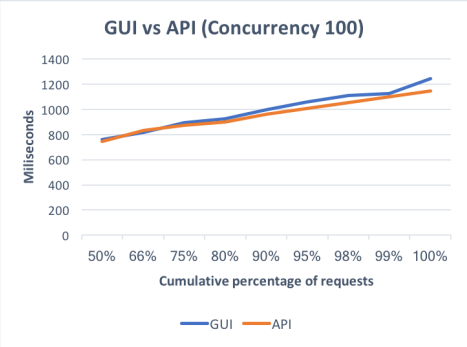

Figure 5.6: Comparison of the cumulative percentage of requests for concurrency 100.

Looking at the graphs in Figures 5.4, 5.5 and 5.6 they show the cumulative percentage of requests served in certain time. Giving an example, Figure 5.4 shows that 50% of the API requests are satisfied in 16 milliseconds. Meanwhile, to guarantee the satisfaction of 99% of requests the estimated value has to be 24 milliseconds.

These results are consistent with what was previous stated where the GUI service takes more time to respond than the API in almost all cases. In Figure 5.5 there is an exception on the 98%, 99% and 100%, this could be caused by some network interference seeing that the average time per request is still lower as seen in Table 5.1.

5.3.3 HGW access local vs remote

This test will compare the access time to Home Assistant being in the same sub network, as a regular user would in his home, as well as through the developed solution, simulating a remote access to the user's home. To test this, Home Assistant was running on the HGW and connected to the same IT network as the testing laptop. This allowed for the local access test as it simulated a home network environment. As for the remote access, the HGW, using the developed components, is connected to the SGH Cloud, running on a different IT network. The testing laptop would then also connect to the SGH Cloud to access Home Assistant, as shown in section 5.2 when the solution was validated.

Having this setup allowed for a remote access test while minimizing the network delay that would be felt by the regular Internet traffic. This traffic would not be controllable and would not reflect the real added delay by using the SGH Cloud to access the HAS. To test the response time, the requests where directed at the Home Assistant home page for both locally and remote.

Analyzing the results, Table 5.2 was created to help summarize the collected metrics

Concurrency Level	Local Access					Remote Access				
	Min (ms)	Max (ms)	Average (ms)	Standard Deviation (ms)	Average per Request (ms)	Min (ms)	Max (ms)	Average (ms)	Standard Deviation (ms)	Average per Request (ms)
1	8	19	8	1	8	10	81	26	22	26
10	37	124	68	8	7	38	129	73	7	7
100	178	763	659	98	7	109	801	711	106	7

Table 5.2: Comparison of request times to the HAS in local vs remote access

and give an overview of the overhead introduced by the SGH Cloud. All values presented are in milliseconds and rounded to the unit, and the same as the previous test an extra column was added showing the average time per request when the concurrency level is greater than one.

Looking at the values in Table 5.2 it is visible, as expected that the local access is always faster than the remote access. An exception is seen on the minimum value with concurrency 100, but that was due to a local access test run that had almost four times the minimum value as all other runs, raising this way the average. Another important aspect to mention about Table 5.2 is the average time per request can induce in error seeing the same values for both local and remote access. This is due to the rounding error, as it is visible by the average of the aggregated of requests that the value is in fact lower in the local access.

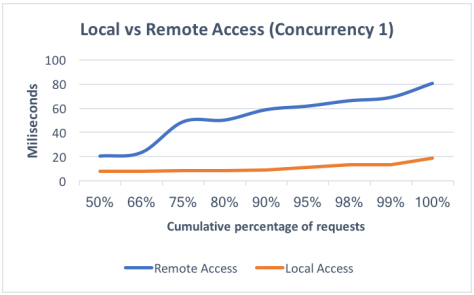

Figure 5.7: Comparison of the cumulative local vs remote access times for concurrency 1.

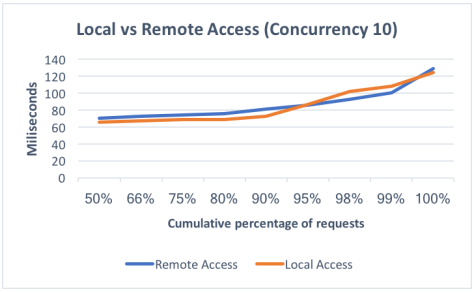

Figure 5.8: Comparison of the cumulative local vs remote access times for concurrency 10.

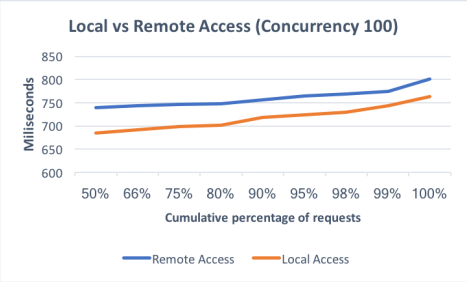

Figure 5.9: Comparison of the cumulative local vs remote access times for concurrency 100.

Considering now Figures 5.7, 5.8 and 5.9, they show the cumulative request percentage in terms of the time they took to be satisfied. These graphs show, again, that the local access time is always lower than the remote access times as expected. Again there is an exception in the concurrency level 10 where the local access is slower to respond to 98% and 99% of the requests. This can be justified by external factors in the network during the time the test was being executed.

Nevertheless, these tests show that the remote access through the SGH Cloud introduces only a small overhead in the communication that is not a deterrent factor given its advantages. The time difference of local vs remote access will obviously increase when an access across the Internet is done. Anyway this is not the end goal of this test as that does not demonstrate the delay introduced by the developed system but rather an out of reach one to both the user as well as the SGH operations.

5.4 ANALYTIC RESULTS

This section will go over two analytic tests performed on this system. These tests are based on some real values but also some mathematic models are used in order to predict a larger scale deployment of this system. The first test will be about the energy consumption of the HGW to understand its impact on a user electricity bill. The second test will go over the SGH Cloud Platform DB and it will estimate its size requirement for various user numbers.

5.4.1 ENERGY CONSUMPTION

This test will measure the energy consumption of the HGW with different CPU loads. This will allow to estimate the necessary savings, to make its use, energy efficient.

The methodology for this test was to measure the HGW energy consumption using a "Kill a Watt[4]" device while introducing different CPU loads. The first CPU load was the idle value while running the software described in chapter 4. Following, a stress test using the sysbench[5] tool, was executed in order to simulate a full system load on the HGW.

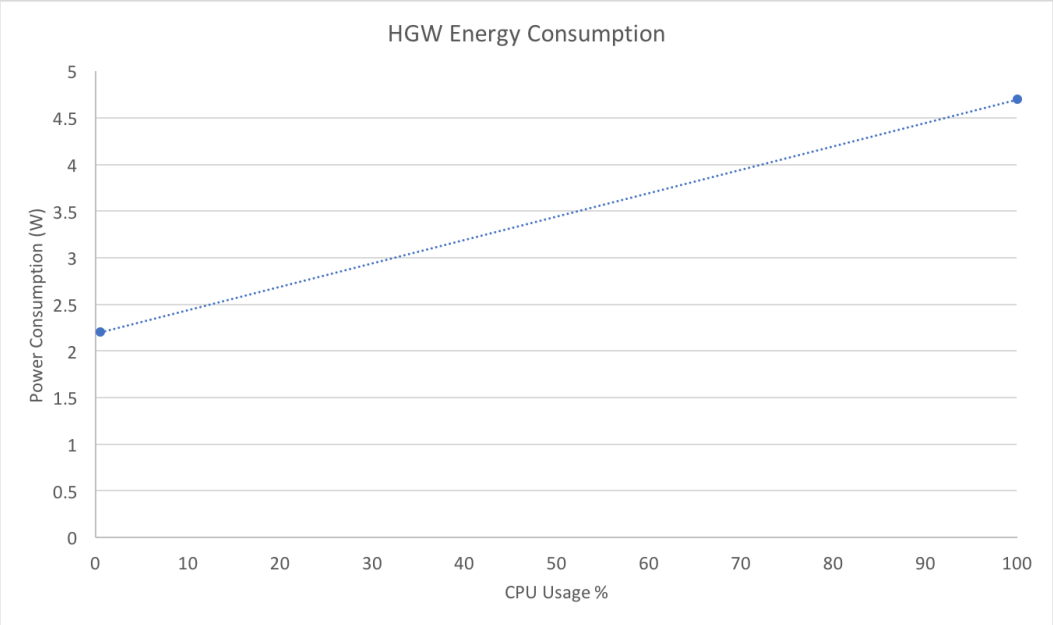

Figure 5.10: Energy consumption for the HGW under different CPU loads.

With the results from this test the chart in Figure 5.10 was created to show the energy consumption of the HGW for different system loads. In this chart the idle and max load consumption was measure on an average of ten observed values. To estimate the energy consumption for the remaining system load values, a linear regression was used.

This test showed the low energy use of the HGW with an idle value of 2.2W and a max of just 4.7W. Assuming different values for the average CPU load of a HGW, Table 5.3 shows the monthly energy consumption in kW/h for this device. Additionally, a cost for each kW/h was obtained from the EDP website[6] and an estimated monthly running cost was calculated. The average kW/h cost was €0.1635 obtained from a 5.75kVA meter on a simple plan.

Looking at Table 5.3 it is visible that even with the gateway always running at full load, which is very unlikely to happen, the monthly energy cost to keep it running has

[4]http://www.p3international.com/products/p4400.html
[5]https://github.com/akopytov/sysbench
[6]https://energia.edp.pt/particulares/energia/tarifarios/

CPU (%)	Energy (W)	Monthly Energy (kW/h)	Estimated Monthly Cost (€)
0.5	2.20	1.58	0.26
10	2.44	1.76	0.29
20	2.69	1.94	0.32
30	2.94	2.12	0.35
40	3.19	2.30	0.38
50	3.44	2.48	0.41
60	3.69	2.66	0.43
70	3.95	2.84	0.46
80	4.20	3.02	0.49
90	4.45	3.20	0.52
100	4.70	3.38	0.55

Table 5.3: Estimated HGW energy running costs.

a small impact in the overall electricity bill. This is important because, for the end user, this device has to be running all the time and if the energy impact was a problem it would be a deterrent factor.

5.4.2 DB size

In this section, an analysis of the SGH Platform DB size will be presented and an estimation of its size growth over time will be performed. The implemented DB schema was presented in Figure 4.5, but for this analysis only the "device" and "ports" tables will be considered when scaling since the other will not be related to how many devices are in production.

The methodology for this test will be to calculate the used disk space for a gateway entry along with its associated forwarding port. Knowing the consumed disk space of one gateway, the scaling will be calculated on a direct proportionality since the DB entries are of fixed length, with mostly auto-generated and known size fields. The only exception to this approach, is the HGW name that is inserted by the user. To solve this an acceptable range will be used and both the minimum and maximum estimations will be presented.

It is important to mention that this test will only present the used disk space by the DB entries and not the full DB size. To accommodate the full DB size, additional considerations had to be taken into account such as the indexes and extra tables created by the DB engine. These will also vary between different DB engines.

Table	Field	Size (bytes)
Device	id	4
	random_id	81
	name	10 to 50
	hardware	4
	software	4
	last_heartbeat	8
	certificate	1411
	certificate_sum	42
	user	19
Ports	port	4
	host	4
	gateway_id	81

Table 5.4: DB fields and respective sizes.

In Table 5.4 there are shown the DB fields along with their sizes per entry. To assist in this calculation the PostgreSQL official documentation was used[7]. As previously stated the "name" field will need to have a size range and thus a minimum of 10 and maximum of 50 characters was used as an acceptable range. Knowing these values and summing all of them, it can be assumed that each gateway will use, in average, 1692 bytes per entry.

Number of GWs	Min DB size (Mb)	Avg DB size (Mb)	Max DB size (Mb)
1	0.001672	0.001692	0.001712
1000	1.672	1.692	1.712
10000	16.72	16.92	17.12
100000	167.2	169.2	171.2
1000000	1672	1692	1712

Table 5.5: DB size prediction for different number of HGWs.

This analysis will allow the estimation of the DB size for future growth as seen in Table 5.5. Looking at this analysis, the raw disk space for the HGW data is reasonable considering that even with a million devices it is only using around 1.7Gb of data. Nevertheless, the actual space used in a real deployment would indeed be larger due to the previously mentioned reasons, such as indexes to allow the fast querying of a DB of this size. Additionally, other tables are present in the DB, but because they are not directly related to the number of devices their impact on a large DB will be

[7]https://www.postgresql.org/docs/9.2/static/datatype.html

insignificant. A possible optimization that would reduce the DB size would be to remove the HGW certificate as an entry, and save it as a file with its name as the respective "gateway_id". This would severely reduce the size to just around 260Mb of data with the same one million devices. Another aspect that this analysis does not focus is the necessary hardware performance to allow multiple concurrent queries to the database when reaching such sizes.

CHAPTER 6

Conclusion

This final chapter of this dissertation aims to provide an overview of the developed work and its final solution. As it was previously discussed in chapter 4 and validated in chapter 5, the end result was a working solution capable of fulfilling the proposed requirements. All this was achieved while maintaining an open-source solution that was also a goal for this project.

Nevertheless, during the development and validation of this solution some issues were found. Firstly, the remote access to the gateways is not authenticated. Even though the hash of the gateway ID is necessary for the URL, that will be present in the DNS queries. Additionally, a server certificate for that gateway is also necessary and is not present in the developed version. This however is easily implemented since the CA has a web API, it would only be necessary for the developed software running along side the Gateway Reverse Proxy Nginx to get a new certificate and configure it to use it.

Another issue related to the HGW HAS forwarding is the assurance the Cloud platform has about the ports the HGW uses. The issues here described are only present in the case of rogue gateways trying to connect or custom firmwares running on them. Still, the use of SSH does not allow, on the server, to lock a user to only be able to forward its connection to a specific port. This is a limitation that is not critical for SSH since it is the client port being forward to the server and not the other way around, which SSH allows to implement that restriction. There is also no assurance the gateway will only open a single connection to the server, allowing the use of unnecessary ports that would become unavailable. The fix of the first issue described in this paragraph, limiting the port the gateway could connect to, would also fix the second, by refusing all other connection attempts. A final issue still related to the HAS forwarding was also

identified, where in an abrupt gateway disconnection the used port was not freed from the platform. This is not a concern when the disconnect is temporary and it actually benefits from not requiring to reconfigure the access as the same port is assigned to that gateway. The problem is when the gateway is permanently disconnected abruptly. The solution to this would be a periodic scrub of the open connections, with removal of all excess ones.

6.1 FUTURE WORK

Regarding future expandability to the system, a few items were found where additional work could improve their use in the overall system. This section aims to provide the next steps to this project in order to improve it and meet extra use cases, contributing to a more stable and complete solution.

With regards to new components in the system, as it was previously stated, there is a need for an administrator dashboard that would allow the SGH team to manage both gateways as well as the Cloud platform itself. This dashboard would be a critical part in a real world deployment for support teams to fix possible issues and inconsistencies with the system. Moreover, this component should also support business models, like multiple gateway plans with different features that should be imposed to the gateways. This would require a better management of the gateway certificates, allowing for their revocation when the user is no longer subscribed, stopping this way further connections from the gateway. Moreover, the certificates could be associated with the current plan the gateway is using restricting this way its features.

Another aspect where the system should improve is the GUIs associated with it, by contemplating more options and settings. Examples of this are to allow, on the gateway GUI, to manage how the HGW connects itself to the Internet, allow for factory reset and more options commonly found on these types of devices. As for the SGH Cloud GUI it should have user profile management, it should contemplate secondary accounts linked to the master one, example of multiple family members accessing the same HAS. Additionally with the business models introduction with the administrator dashboard, the user subscriptions and licenses should also be manageable by the user in the existing SGH Cloud GUI.

Considering the HGW, additional management tasks should be allowed on it by the SGH Cloud. With this, additional plug-ins would be needed, as for example automatic configuration backup to the Cloud, and gateway feature management depending on the

current subscription plan. These are just some examples of possible additions to the gateway feature set.

Looking now at the current implemented OTA update and image generation systems, there are some improvements to be made. These are advance and complex systems that were approached in this dissertation for their importance in a final consumer product. However, they are not ready for a deployment and further attention should be given to them. Analyzing the image generation system first, only a basic Linux image was generated. In order for this system to become viable, a fully functional image with all components installed and configured needs to be generated automatically. In addition to Home Assistant, which a Yocto layer for it[1] was found, there is still the need to create a layer to download, install and configure all the developed software for this dissertation. Going over the update system, the current implemented solution already provides a GUI to manage the deployments of new versions to devices, and even supports dual partition scheme which is a very important aspect when dealing with remote devices not easily reachable. Still the current system only supports full image updates, which means a small change in an update would imply a full image to be downloaded by the client and installed. From this, another problem rises, given that over time device numbers and image sizes tend to increase, with the extra features and content, the updates would become a bigger burden on the SGH infrastructure, thus not scaling well. The solution to this would be to mix the current update system of full image updates for large major updates, combined with an incremental update system, as the ones mentioned in the state of the art, for smaller updates.

Lastly, the SGH Cloud platform could benefit from some deployment improvements, such as using high performance web servers, like Nginx, instead of the default development server for the developed components. Moreover, a mechanism of authentication inside the SGH infrastructure between services would improve its security and, combined with HTTPS, allow for a public infrastructure deployment. In order to allow the system's automatic scalability, an additional monitoring service could be developed to automatically provision new servers and automate their integration with the existing infrastructure.

[1] https://github.com/bachp/meta-homeassistant

References

[1] Eurostat, *Consumption of energy*, 2016. [Online]. Available: http://ec.europa.eu/eurostat/statistics-explained/index.php/Consumption_of_energy (visited on 01/23/2017).

[2] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP*. 2010, pp. 335–351, ISBN: 9780123751652. DOI: 10.1016/B978-0-12-375165-2.00022-3. arXiv: arXiv:1011.1669v3. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123751652000223.

[3] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything", *Cisco white paper*, no. April, pp. 1–11, 2011, ISSN: 09598138. DOI: 10.1109/IEEESTD.2007.373646. arXiv: arXiv:1011.1669v3.

[4] C. Pham, Y. Lim, and Y. Tan, "Management Architecture for Heterogeneous IoT Devices in Home Network", 2016.

[5] Union of Concerned Scientists, *Environmental Impacts of Renewable Energy Technologies*. [Online]. Available: http://www.ucsusa.org/clean-energy/renewable-energy/environmental-impacts#.WIi9braLRHZ (visited on 01/25/2017).

[6] ——, *The Hidden Costs of Fossil Fuels*, 2016. [Online]. Available: http://www.ucsusa.org/clean-energy/coal-and-other-fossil-fuels/hidden-cost-of-fossils (visited on 01/25/2017).

[7] M. Asadullah and A. Raza, "An Overview of Home Automation Systems", pp. 27–31, 2016.

[8] E. Ko, J. Cesi, V. Bachler, H. N. Vu, and H. D, "Smart Home Automation System for Energy Efficient Housing", no. May, pp. 26–30, 2014.

[9] K. Moser, J. Harder, and S. G. M. Koo, "Internet of things in home automation and energy efficient smart home technologies", *Systems, man and cybernetics (smc), 2014 ieee international conference on*, pp. 1260–1265, 2014, ISSN: 1062922X. DOI: 10.1109/SMC.2014.6974087.

[10] INSTEON, "Insteon Whitepaper: Compared", pp. –45, 2013. [Online]. Available: http://cache.insteon.com/documentation/insteon_compared.pdf.

[11] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication systems for building automation and control", *Proceedings of the ieee*, vol. 93, no. 6, pp. 1178–1203, 2005, ISSN: 00189219. DOI: 10.1109/JPROC.2005.849726.

[12] KNX, *What is KNX?*, 2014. [Online]. Available: https://www.knx.org/knx-en/knx/association/what-is-knx/index.php (visited on 02/13/2017).

[13] Siemens, "Communication in building automation", p. 14, 2013. [Online]. Available: www.siemens.com/buildingtechnologies.

[14] KNX, *Configuration Modes*, 2014. [Online]. Available: https://www.knx.org/in/knx/technology/configuration-modes/index.php (visited on 02/14/2017).

[15] K. Ashton, "That 'Internet of Things' Thing", *Rfid journal*, p. 4986, 2009. [Online]. Available: http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf%5Cnpapers3://publication/uuid/8191C095-0D90-4A17-86B0-550F2F2A6745.

[16] J. Camhi, *BI Intelligence projects 34 billion devices will be connected by 2020*, 2015. (visited on 02/08/2017).

[17] C. Buckle, *Digital consumers own 3.64 connected devices*, 2016. [Online]. Available: https://www.globalwebindex.net/blog/digital-consumers-own-3.64-connected-devices (visited on 02/09/2017).

[18] U. C. Bureu, *International Data Base, World Population: 1950-2050*, 2016. [Online]. Available: http://www.census.gov/population/international/data/idb/worldpopgraph.php (visited on 02/09/2017).

[19] ITU and UNESCO, *The State of Broadband 2016: Broadband catalyzing sustainable development*, September. 2016, p. 110, ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3. [Online]. Available: http://www.broadbandcommission.org/Documents/reports/bb-annualreport2016.pdf.

[20] P. Anand, "Towards evolution of M2M into Internet of Things for analytics", *2015 ieee recent advances in intelligent computational systems, raics 2015*, no. December, pp. 388–393, 2016. DOI: 10.1109/RAICS.2015.7488447.

[21] Microsoft, *The OSI Model's Seven Layers Defined and Functions Explained*, 2017. [Online]. Available: https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained (visited on 05/03/2017).

[22] CISCO, *What is OSI Model & the Overall Explanation of ISO 7 Layers*, 2016. [Online]. Available: http://www.cisco1900router.com/what-is-ios-model-the-overall-explanation-of-ios-7-layers.html (visited on 02/14/2017).

[23] B. WOOD, *The Evolution of WiFi*, 2014. [Online]. Available: http://purple.ai/history-wifi/ (visited on 05/03/2017).

[24] Bluetooth SIG, *Our History | Bluetooth Technology Website*, 2017. [Online]. Available: https://www.bluetooth.com/about-us/our-history (visited on 05/03/2017).

[25] TI, *6LoWPAN | Wireless Connectivity*, 2014. [Online]. Available: http://www.ti.com/lsds/ti/wireless-connectivity/6lowpan/overview.page (visited on 05/09/2017).

[26] J. Olsson, "6LoWPAN demystified", p. 13, 2014. [Online]. Available: http://www.ti.com/lit/wp/swry013/swry013.pdf.

[27] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. 2011, vol. 43, ISBN: 9780470747995. [Online]. Available: http://nolegendhere.blogspot.pt/2012/04/sessio-7.html.

[28] E. Tovar, M. Alves, and A. Koubâa, "IEEE 802.15.4: a federating communication protocol for time-sensitive wireless sensor networks", no. February, 2006.

[29] P. Domingues, P. Carreira, R. Vieira, and W. Kastner, "Building automation systems: Concepts and technology review", *Computer standards & interfaces*, vol. 45, no. November, pp. 1–12, 2016, ISSN: 09205489. DOI: 10.1016/j.csi.2015.11.005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0920548915001361.

[30] Z-Wave Alliance, *Z-Wave Plus^{TM} Certification*, 2017. [Online]. Available: http://z-wavealliance.org/z-wave_plus_certification/ (visited on 06/26/2017).

[31] M. Andersson, "Use case possibilities with Bluetooth low energy in IoT applications -White paper", p. 16, [Online]. Available: https://www.u-blox.com/sites/default/files/products/documents/BluetoothLowEnergy-IoT-Applications_WhitePaper_(UBX-14054580).pdf.

[32] K. Torvmark, "Three flavors of Bluetooth ® : Which one to choose? The current state of Smart", no. January 2013, 2014.

[33] K. Shah, J. Notor, T. Godfrey, B. Rolfe, Y. Seok, B. Tuncer, A. Mody, and F. Khatibi, "Smart Grid Task Group – Sub 1 GHz White Paper", 2016.

[34] N. Ahmed, H. Rahman, and M. Hussain, "A comparison of 802.11ah and 802.15.4 for IoT", *Ict express*, vol. 2, no. 3, pp. 100–102, 2016, ISSN: 24059595. DOI: 10.1016/j.icte.2016.07.003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2405959516300650.

[35] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network", *2016 international conference on selected topics in mobile and wireless networking, mownet 2016*, 2016. DOI: 10.1109/MoWNet.2016.7496622.

[36] F. Azzola, *MQTT protocol tutorial: how to use mqtt in iot projects*, 2016. [Online]. Available: https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html (visited on 06/28/2017).

[37] P. Kayal, "A Comparison of IoT Application Layer Protocols Through A Smart Parking Implementation", PhD thesis, North Carolina State University, 2017.

[38] T. Salman and R. Jain, "Networking Protocols for Internet of Things", pp. 1–28, 2013. DOI: 10.1002/9781119173601.ch13. [Online]. Available: http://www.cse.wustl.edu/$%5Csim$jain/cse570-15/ftp/iot_prot.pdf.

[39] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *Ieee communications surveys and tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015, ISSN: 1553877X. DOI: 10.1109/COMST.2015.2444095. arXiv: arXiv:1011.1669v3.

[40] J. O'Hara, *Connecting Bussiness for Value*, 2014.

[41] V. John and X. Liu, "A Survey of Distributed Message Broker Queues", 2017. arXiv: arXiv:1704.00411v1.

[42] RF Wireless, *What is CoAP IoT protocol | CoAP Architecture,message header*, 2012. [Online]. Available: http://www.rfwireless-world.com/IoT/CoAP-protocol.html (visited on 06/29/2017).

[43] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)", Internet Engineering Task Force (IETF), Tech. Rep., 2014, pp. 1–112.

[44] A. Mathur, "A HUMAN-CENTERED APPROACH TO IMPROVING THE USER EXPERIENCE", PhD thesis, University of Maryland, 2016.

[45] N. Willis, *Deciding between Buildroot and Yocto*, 2016. [Online]. Available: https://lwn.net/Articles/682540/ (visited on 05/26/2017).

[46] OpenEmbedded, *Openembedded*, 2017. [Online]. Available: http://www.openembedded.org/wiki/Main_Page (visited on 05/26/2017).

[47] Android, *A/B (Seamless) System Updates | Android Open Source Project*, 2017. [Online]. Available: https://source.android.com/devices/tech/ota/ab_updates (visited on 05/26/2017).

[48] K. Group, "Device side Software Update Strategies for Automotive Grade Linux", 2016. [Online]. Available: https://konsulko.com/wp-content/uploads/2016/09/Device-sideSoftwareUpdateStrategiesforAGL.pdf.

[49] W. Yuan, H. Sun, X. Wang, and X. Liu, "Towards Efficient Deployment of Cloud Applications Through Dynamic Reverse Proxy Optimization", 2013. DOI: 10.1109/HPCC.and.EUC.2013.97.

[50] Nginx, *What is NGINX? - NGINX*, 2017. [Online]. Available: https://www.nginx.com/resources/glossary/nginx/ (visited on 05/29/2017).

[51] Datadog, *How to monitor NGINX*, 2015. [Online]. Available: https://www.datadoghq.com/blog/how-to-monitor-nginx/ (visited on 05/29/2017).

[52] Nginx, *NGINX 3rd Party Modules | NGINX*, 2017. [Online]. Available: https://www.nginx.com/resources/wiki/modules/ (visited on 05/29/2017).

[53] ——, *Compare Features in NGINX Plus and NGINX Open Source*, 2017. [Online]. Available: https://www.nginx.com/products/feature-matrix/ (visited on 05/29/2017).

[54] HAProxy, *HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer*, 2017. [Online]. Available: http://www.haproxy.org/#desc (visited on 05/29/2017).

[55] Datadog, *Monitoring HAProxy performance metrics*, 2015. [Online]. Available: https://www.datadoghq.com/blog/monitoring-haproxy-performance-metrics/ (visited on 05/29/2017).

[56] Observing, *WebSocket loadbalancer battle*, 2015. [Online]. Available: https://github.com/observing/balancerbattle (visited on 01/01/2001).

[57] M. L. Das and N. Samdaria, "On the security of SSL/TLS-enabled applications", *Applied computing and informatics*, vol. 10, no. 1-2, pp. 68–81, 2014, ISSN: 22108327. DOI: 10.1016/j.aci.2014.02.001. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2210832714000039.

[58] Google, *Transparency Report – Google*, 2017. [Online]. Available: https://www.google.com/transparencyreport/https/metrics/?hl=en (visited on 05/29/2017).

[59] J. Edge, *Client Certificates vs. Server Certificates – What's the Difference? | Symantec Connect Community*, 2013. [Online]. Available: https://www.symantec.com/connect/blogs/client-certificates-vs-server-certificates-what-s-difference (visited on 05/30/2017).

[60] K. Johnson, *What is Client Authentication and Why Do I Need It?*, 2016. (visited on 05/30/2017).

[61] *Open-Source Leaderboard*, 2016. [Online]. Available: https://ossmetrics.com/leaderboard (visited on 01/30/2017).

[62] K. Kreuzer, *openHAB 2 has arrived*, 2017. [Online]. Available: http://www.kaikreuzer.de/2017/01/23/openhab2/ (visited on 02/01/2017).

[63] Samsung, *Meet the Samsung SmartThings Hub – SmartThings Support*. [Online]. Available: https://support.smartthings.com/hc/en-us/articles/205956900-Meet-the-Samsung-SmartThings-Hub (visited on 05/24/2017).

[64] G. CLAUSER, *What Is Alexa? What Is the Amazon Echo, and Should You Get One? | The Wirecutter*, 2017. [Online]. Available: http://thewirecutter.com/reviews/what-is-alexa-what-is-the-amazon-echo-and-should-you-get-one/ (visited on 05/24/2017).

[65] Amazon, *Amazon.com: Alexa Skills*, 2017. [Online]. Available: https://www.amazon.com/b?ie=UTF8%5C&node=13727921011 (visited on 05/24/2017).

[66] Google, *Google Home – Made by Google*, 2017. [Online]. Available: https://madeby.google.com/home/ (visited on 05/24/2017).

[67] ——, *Features – Google Home – Made by Google*, 2017. [Online]. Available: https://madeby.google.com/home/features/#?filters=answers,entertainment,manage,plan,home,fun%5C&feature= (visited on 05/24/2017).

[68] Apple, *Apple Home*, 2017. [Online]. Available: https://www.apple.com/ios/home/ (visited on 05/25/2017).

[69] Vera, *Smart Home Controllers Vera*, 2017. [Online]. Available: http://getvera.com/controllers/ (visited on 10/03/2017).

[70] ——, *Vera Compatibility*, 2017. [Online]. Available: http://getvera.com/compatibility/ (visited on 10/03/2017).

[71] Kevin, *Apache License 2.0 (Apache-2.0)*, 2014. [Online]. Available: https://tldrlegal.com/license/apache-license-2.0-(apache-2.0) (visited on 11/18/2017).

[72] ——, *Eclipse Public License 1.0 (EPL-1.0)*, 2014. [Online]. Available: https://tldrlegal.com/license/eclipse-public-license-1.0-(epl-1.0) (visited on 02/06/2017).

[73] ——, *GNU General Public License v3 (GPL-3)*, 2014. [Online]. Available: https://tldrlegal.com/license/gnu-general-public-license-v3-(gpl-3) (visited on 02/06/2017).

[74] A. Clark, *A RESTful json api to BIND DNS*, 2014. [Online]. Available: https://github.com/ajclark/bind-restapi.

[75] A. Parecki, *OAuth 2 Simplified*, 2012. [Online]. Available: https://aaronparecki.com/oauth-2-simplified/ (visited on 07/13/2017).

[76] Z. Dennis, *Choosing an SSO Strategy: SAML vs OAuth2 | Mutually Human*, 2013. [Online]. Available: https://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/ (visited on 07/13/2017).

[77] Home Assistant, *Installation on your computer - Home Assistant*, 2017. [Online]. Available: https://home-assistant.io/docs/installation/python/ (visited on 07/11/2017).

[78] ——, *Autostart using systemd - Home Assistant*, 2017. [Online]. Available: https://home-assistant.io/docs/autostart/systemd/ (visited on 07/11/2017).

[79] CloudFlare, *CFSSL*, 2017. [Online]. Available: https://github.com/cloudflare/cfssl.

[80] N. Sullivan, *How to build your own public key infrastructure*, 2015. [Online]. Available: https://blog.cloudflare.com/how-to-build-your-own-public-key-infrastructure/ (visited on 07/18/2017).

[81] D. Hiltgen, *Build a container with cfssl in it*, 2015. [Online]. Available: https://github.com/dhiltgen/docker-cfssl.

[82] PowerDNS, *Installing PowerDNS*. [Online]. Available: https://doc.powerdns.com/md/authoritative/installation/ (visited on 07/18/2017).

[83] ——, *Generic PostgreSQL*. [Online]. Available: https://doc.powerdns.com/md/authoritative/backend-generic-postgresql/ (visited on 07/18/2017).

[84] Mender, *Create a test environment | Mender documentation*. [Online]. Available: https://docs.mender.io/1.0/getting-started/create-a-test-environment (visited on 07/18/2017).

[85] CNX Software, *12MB Minimal Image for Raspberry Pi using the Yocto Project*, 2013. [Online]. Available: http://www.cnx-software.com/2013/07/05/12mb-minimal-image-for-raspberry-pi-using-the-yocto-project/ (visited on 07/25/2017).

[86] Jumpnow Technologies, *Building Raspberry Pi Systems with Yocto*, 2017. [Online]. Available: http://www.jumpnowtek.com/rpi/Raspberry-Pi-Systems-with-Yocto.html (visited on 07/25/2017).

[87] Mender, *Building a Mender Yocto Project image*. [Online]. Available: https://docs.mender.io/1.0/Artifacts/Building-Mender-Yocto-image (visited on 07/25/2017).

[88] Wikia, *Advanced Protocol | Home Easy Hacking Wiki*, 2015. [Online]. Available: http://homeeasyhacking.wikia.com/wiki/Advanced_Protocol (visited on 07/25/2017).

[89] Home Assistant, *MQTT Light - Home Assistant*, 2017. [Online]. Available: https://home-assistant.io/components/light.mqtt/ (visited on 07/11/2017).

Appendix A: Home Assistant Configuration and Web UI

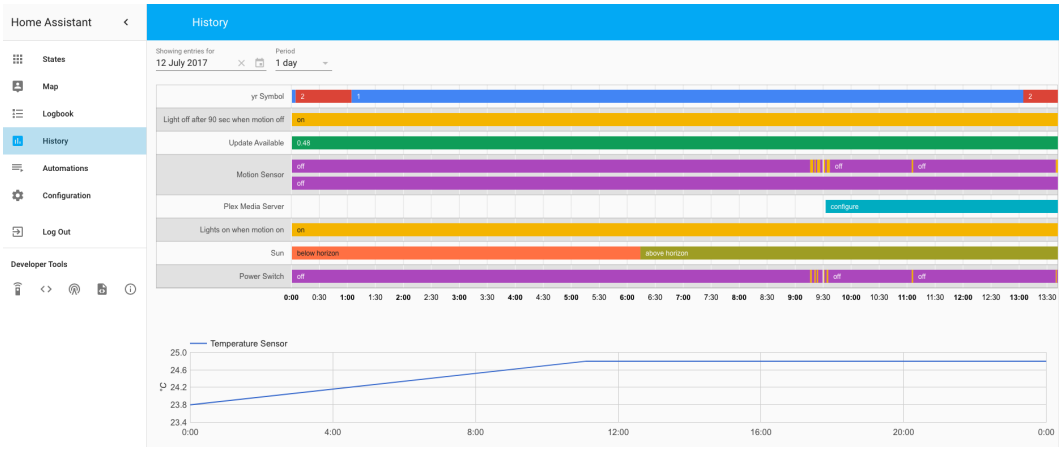

Figure 1: History of devices states in Home Assistant

```
automation:
  - id: light_on_when_motion
    alias: Lights on when motion on
    trigger:
      - platform: state
        entity_id: binary_sensor.motion_sensor_3
        to: 'on'
    action:
      - service: switch.turn_on
        entity_id: switch.power_switch_6
```

Snippet 6: Example of an automation rule to turn the light when motion is detected

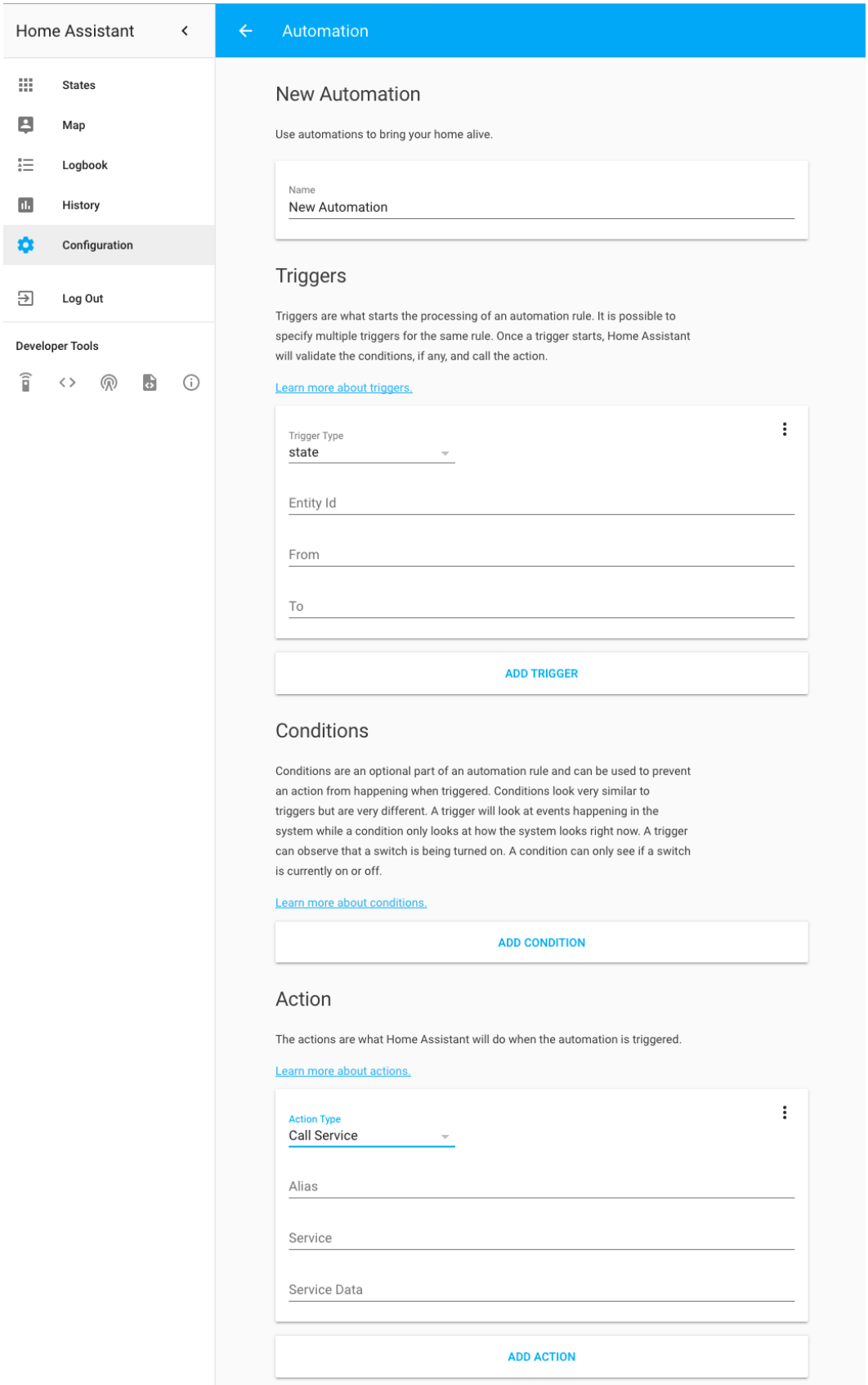

Figure 2: Create an automation rule in Home Assistant

```
light:
  - platform: mqtt
    name: "Office Light RGB"
    state_topic: "office/rgb1/light/status"
    command_topic: "office/rgb1/light/switch"
    brightness_state_topic: "office/rgb1/brightness/status"
    brightness_command_topic: "office/rgb1/brightness/set"
    rgb_state_topic: "office/rgb1/rgb/status"
    rgb_command_topic: "office/rgb1/rgb/set"
    state_value_template: "{{ value_json.state }}"
    brightness_value_template: "{{ value_json.brightness }}"
    rgb_value_template: "{{ value_json.rgb | join(',') }}"
    qos: 0
    payload_on: "ON"
    payload_off: "OFF"
    optimistic: false
```

Snippet 7: Example of an MQTT light configuration (obtained from [89])

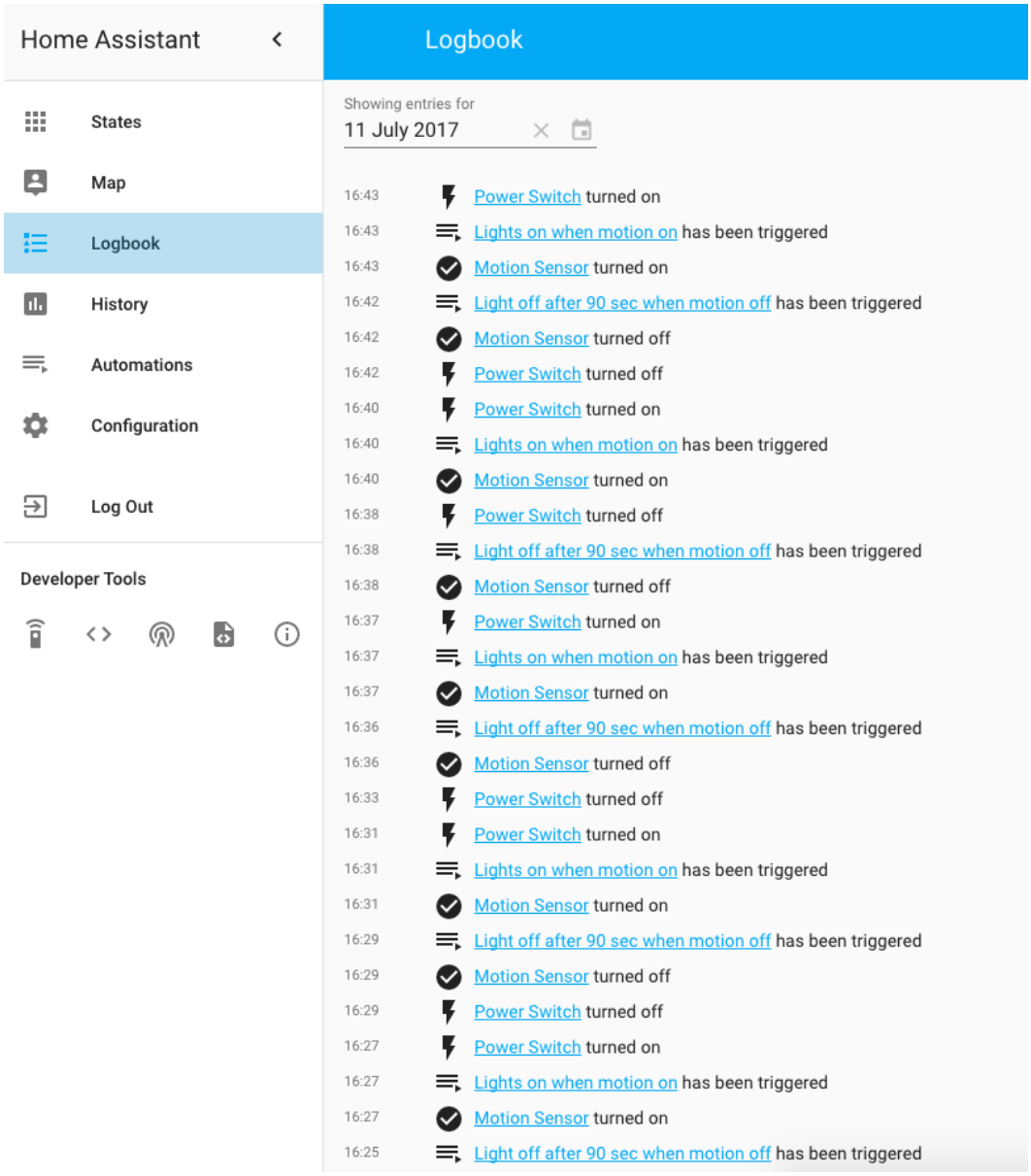

Figure 3: Home Assistant devices log

Appendix B: Client Management Software Configuration

```yaml
components:
  - register
  - heartbeat
  - ssh_tunnel

server: &server
  http_prefix: https://
  host: smart-gateway.lan
  port: 443
  gw_prefix: /gw
  url_check: /api/v1/device/check-registered
  timeout: 5

device: &device
  device_file: "/resources/device.json"
  hardware_file: "/resources/hardware.json"
  software_file: "/resources/software.json"
  private_key: "/certs/private-key.pem"
  certificate: "/certs/certificate.pem"
  ca_certificate: "/etc/ssl/certs/smart-gateway-ca.pem"

register:
  <<: *server
  <<: *device
  local_port: 60000

heartbeat:
  <<: *server
  <<: *device
  delay: 5
  url_heartbeat: /api/v1/device/heartbeat

ssh_tunnel:
  <<: *server
  <<: *device
  delay: 5
  url_get_port: /api/v1/forwarder/get-port
  url_free_port: /api/v1/forwarder/free-port
  local_host: localhost
  local_port: 8123
  ssh_user: ssh-tunnels
```

Snippet 8: Example of the configuration file for the client management system

Appendix C: Home Gateway (HGW) GUI

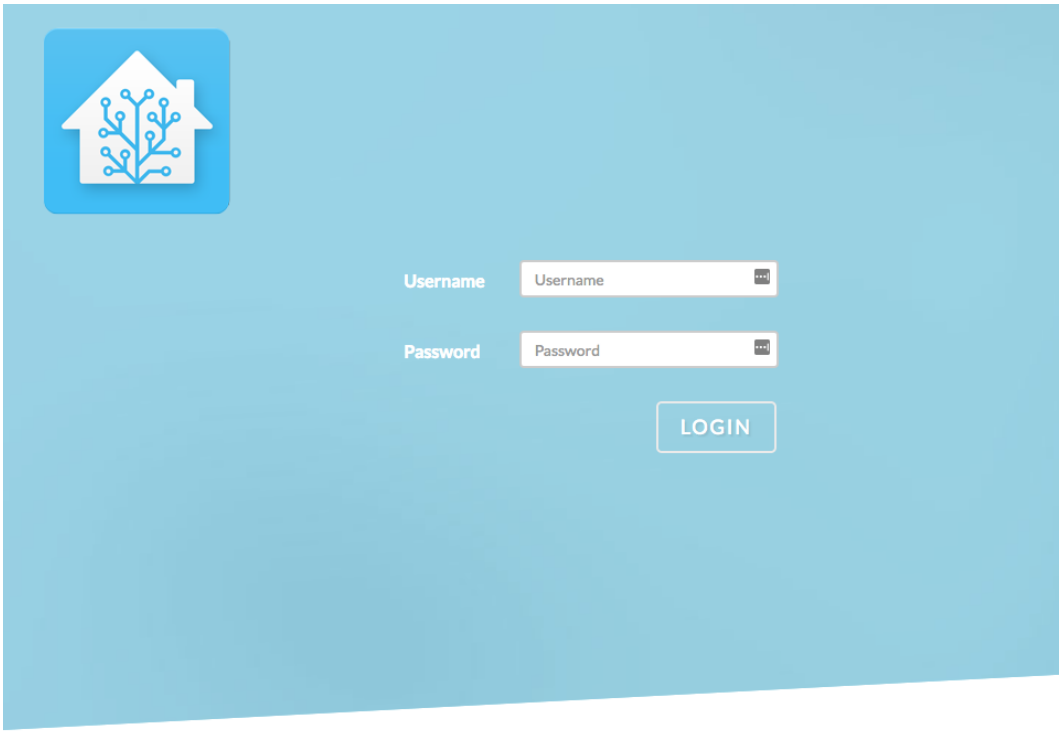

Figure 4: HGW GUI Login page

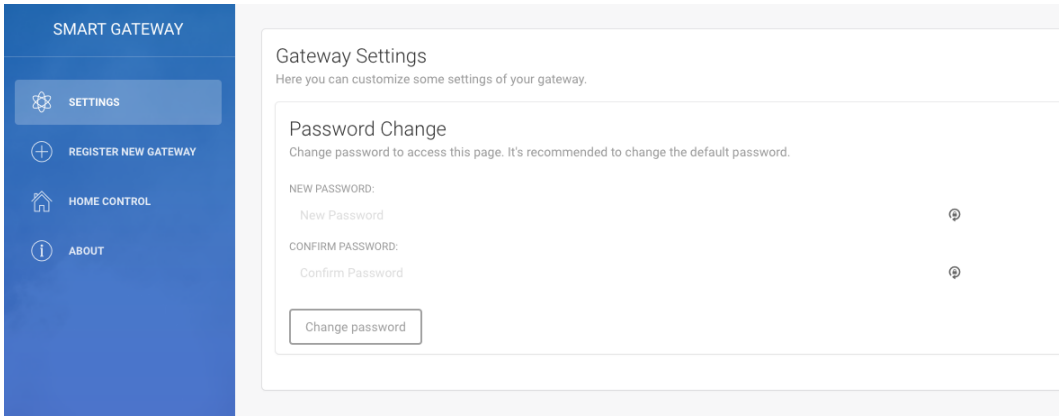

Figure 5: HGW GUI settings page

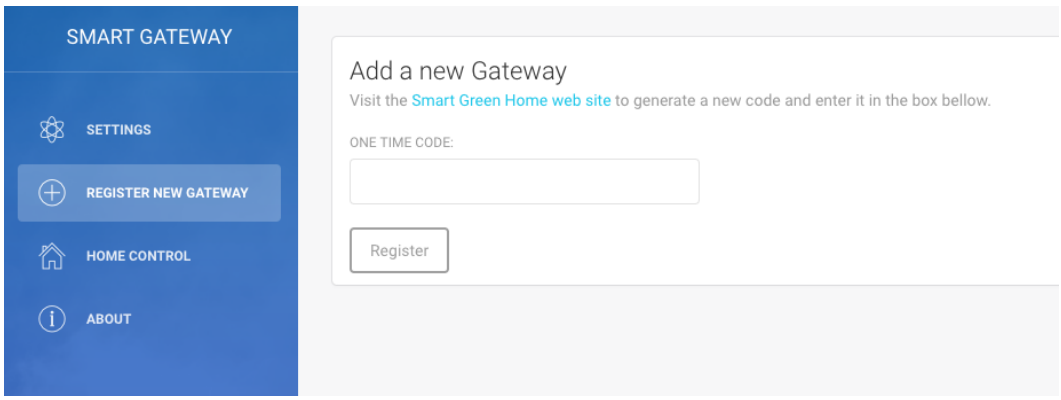

Figure 6: HGW GUI register page

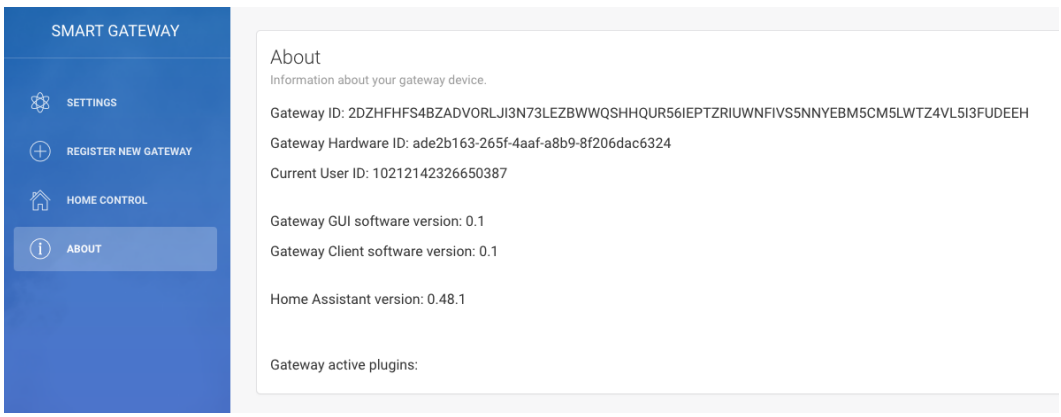

Figure 7: HGW GUI about page

Appendix D: Smart Green Home (SGH) GUI

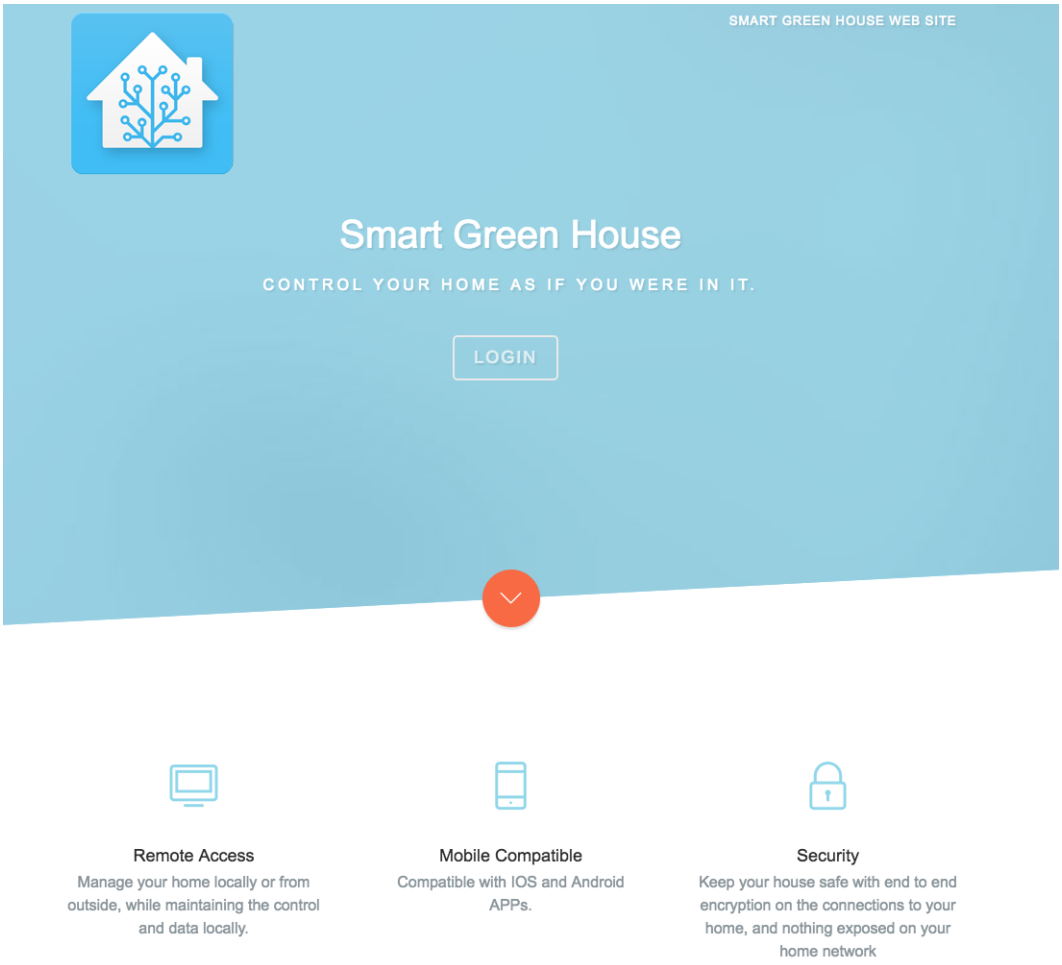

Figure 8: SGH GUI home page

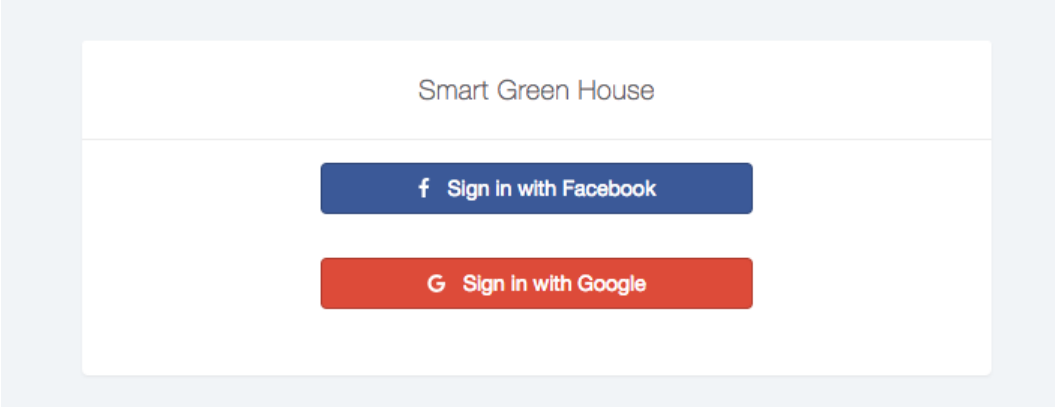

Figure 9: SGH GUI login page

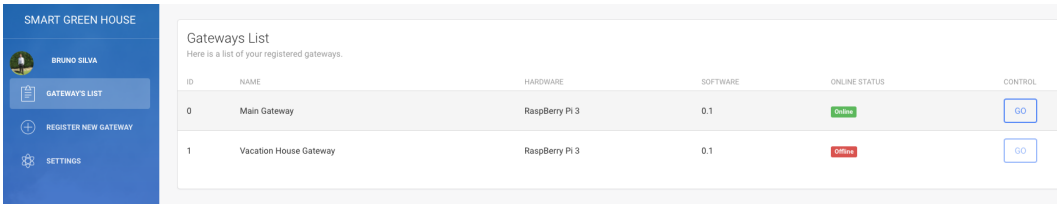

Figure 10: SGH GUI user dashboard

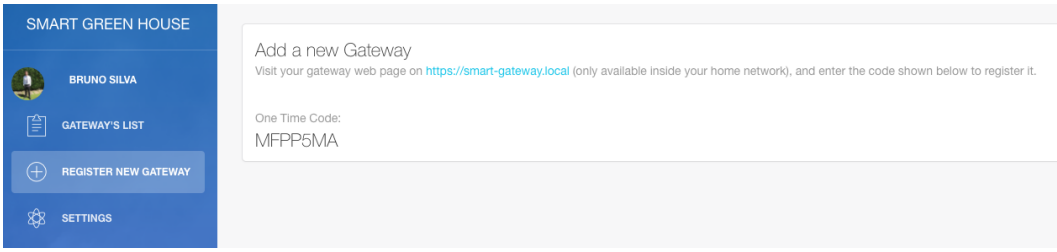

Figure 11: SGH GUI register new gateway

Appendix E: Mender Web UI

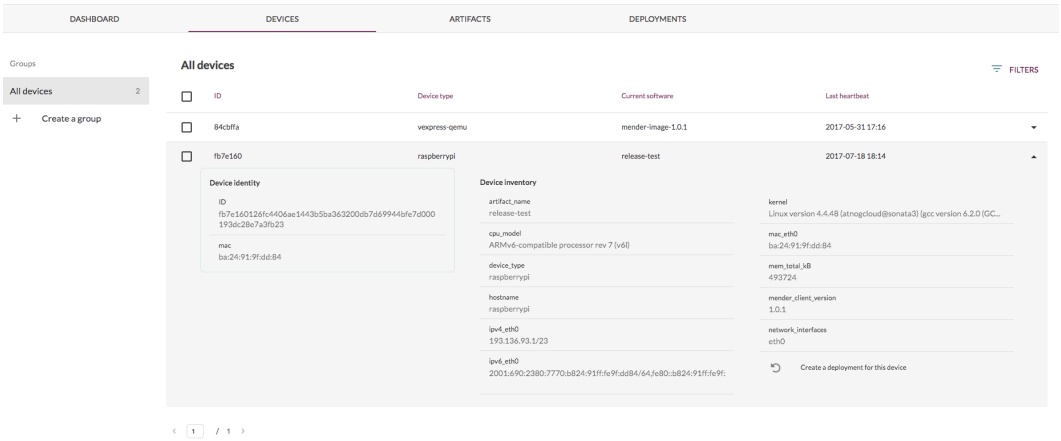

Figure 12: Example of Mender devices list with details

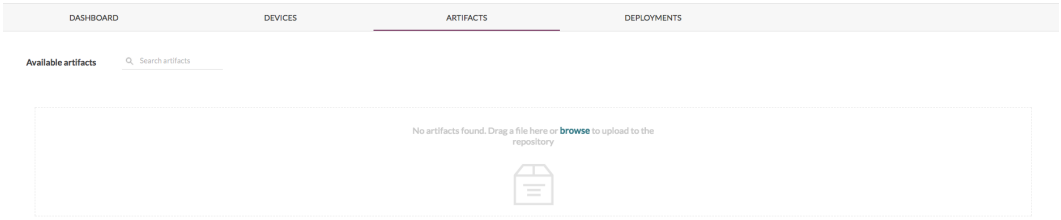

Figure 13: Example of Mender upload of update images

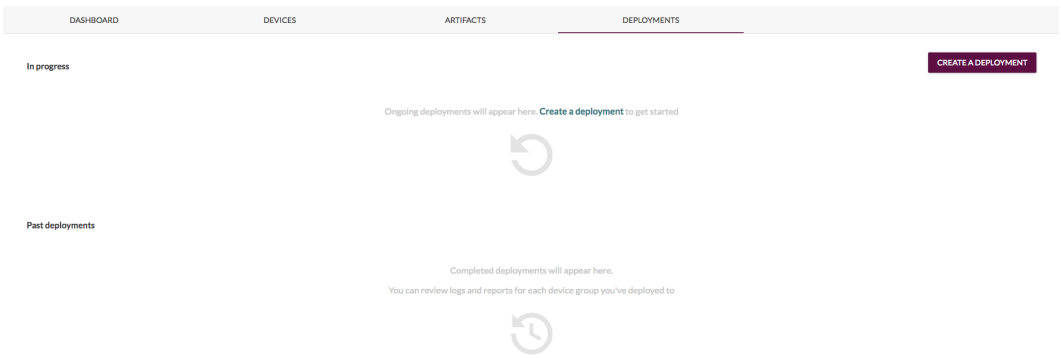

Figure 14: Example of Mender ongoing deployments

Appendix F: Home Assistant RF plug-in

```
switch:
  platform: rf_arduino
  device: "/dev/cu.wchusbserial1a12130"
  switches:
    home_easy:
      myplug1:
        name: "myplug1"
        address: "19266510"
        device: "1"
      myplug2:
        name: "myplug2"
        address: "19266510"
        device: "2"
```

Snippet 9: Example configuration for the developed Home Assistant plug-in.

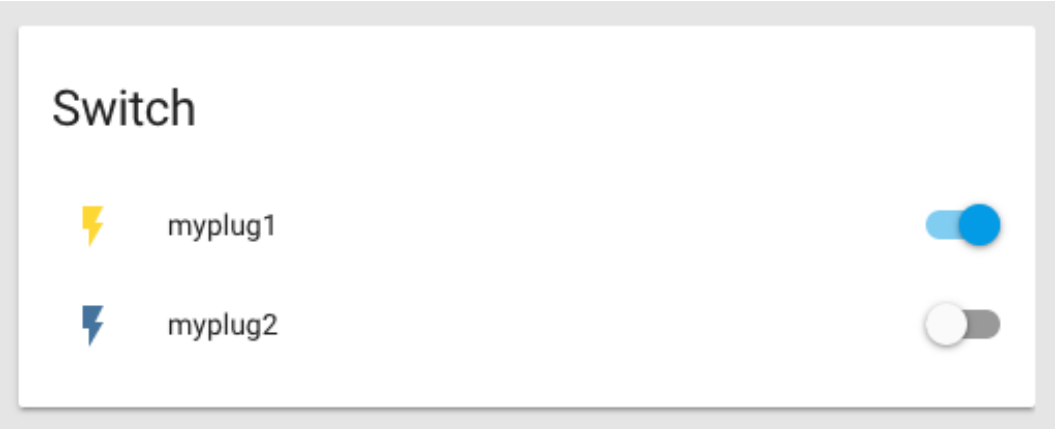

Figure 15: Developed RF Arduino plug-in in Home Assistant