UNIVERSIDAD DE GRANADA

Trabajo Fin de Grado

Grado en Ingeniería Informática

Creation of a voice-driven controller for home automation

Autor
David Vargas Carrillo

Director
Juan Antonio Holgado Terriza

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 29 de agosto de 2018

Creation of a voice-driven controller for home automation

Autor
David Vargas Carrillo

Director
Juan Antonio Holgado Terriza

Creación de un controlador domótico activado por voz

David Vargas Carrillo

Palabras clave: domótica, asistencia por voz, sistemas distribuidos, Raspberry Pi, software libre

Resumen

El objetivo principal de este proyecto es la creación de un controlador domótico activado por voz en un sistema embebido, como la *Raspberry Pi*, centrándose en el uso de software libre, obteniendo la máxima compatibilidad y el mínimo coste.

Para conseguirlo, se ha analizado la situación actual del sector, distinguiendo entre dispositivos domóticos, asistentes de voz y sistemas orientados a la automatización del hogar. A través de la Ingeniería del Software, se han estudiado las posibles necesidades de los usuarios, intentando suplir las carencias actuales del sector. Finalmente, se presenta una implementación de un sistema domótico en un entorno real, utilizable y extensible a cualquier situación cotidiana.

Por tanto, el proyecto trata de demostrar las infinitas oportunidades que habilita el reciente campo de la domótica, y la posibilidad de crear sistemas domóticos funcionales de bajo coste.

Creation of a voice-driven controller for home automation

David Vargas Carrillo

Keywords: home automation, voice assistance, distributed systems, Raspberry Pi, open source

Abstract

The main goal of this project is the creation of a low-cost, voice-driven home automation controller in a embedded system, such as the *Raspberry Pi*, using open source technologies and trying to obtain maximum compatibility with minimum cost.

To achieve this, I have analyzed the current state of the sector, distinguishing between domotic devices, voice assistants and home automation oriented systems. Through Software Engineering, I have studied the possible necessities of the users, trying to make up for the scarcities in this sector. Finally, I show an implementation of a home automation system in a real environment, usable and extensible to any daily situation.

Therefore, this project tries to demonstrate the infinite opportunities that the recent field of domotics enables, and the possibility of creating low-cost functional home automation systems.

Yo, **David Vargas Carrillo**, alumno de la titulación GRADO EN INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76592492P, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: David Vargas Carrillo

Granada, a 29 de agosto de 2018

D. **Juan Antonio Holgado Terriza**, Profesor del **Departamento de Lenguajes y Sistemas Informáticos** de la **Universidad de Granada**.

Informa:

Que el presente trabajo, titulado ***Creation of a voice-driven controller for home automation***, ha sido realizado bajo su supervisión por **David Vargas Carrillo**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada, a 29 de agosto de 2018.

El director:

Juan Antonio Holgado Terriza

Agradecimientos

A mis padres, cuyo esfuerzo y dedicación han hecho que hoy esté escribiendo estas líneas.

A todos los compañeros y amigos que han estado conmigo en este camino, por haberlo hecho mucho más agradable y ameno.

Y, por supuesto, a Juan Antonio, por haber aceptado mi idea y haber hecho posible este proyecto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"I am a HAL 9000 computer. I became operational at the H.A.L. plant in Urbana, Illinois on the 12th of January 1992. My instructor was Mr. Langley, and he taught me to sing a song."

These words were spoken by HAL 9000, the artificial general intelligence depicted in the movie *2001: A Space Odyssey* by Stanley Kubrick, published back in 1968. In this film, HAL 9000 is in charge of controlling the systems of the *Discovery One* spacecraft and interacting with the ship's astronaut crew. The abilities of this computer were impressive: it was capable of speech recognition, facial recognition, natural language processing, automated reasoning and many other features characteristic of the most complete artificial intelligence ever created. And on top of that, it was also capable of doing tasks that are now known as home automation.

Of course, in 1968 the field of Artificial Intelligence was only taking its first steps, and these features were only a dream in many people's minds. Nevertheless, *2001: A Space Odyssey* contributed greatly to the popularization of these technologies among the general public.

Today, home automation and voice assistance are experiencing one of their most popular moments, thanks to the lower cost of components and the incredible development of Artificial Intelligence and Internet of Things by leading companies. And most importantly, these long-awaited technologies are finally within everyone's reach.

1.1 Incentive

The interest from companies about home automation and voice assistance has been growing in this decade. Currently, we can find solutions from technology companies that combine a virtual assistant with a home automation

system, and these are exactly the most popular ones.

On the other hand, companies that have classically made home appliances and lightning systems, are now entering the smart home market. The range of *smart devices* is enormous at the moment, and many users may feel lost when looking for a solution for their homes. This is one of the problems that I identified, but not the only one.

Another big problem is that home automation products tend to work only with other devices from the same maker. For example, Philips lightning systems require a Philips bridge and a Philips mobile application in order to work. But if the user has lights from different makers, he will probably need to install more bridges and more applications in his mobile phone. However, all bridges usually do the same job: receiving commands via WiFi or cable and sending them to the domotic devices via Zigbee or Z-Wave, for example (both are popular communication protocols in domotics). Makers are not moving towards unification, but to differentiation.

Luckily, there are some systems that can unify a bit a home automation system composed by devices from different makers. For example, Apple HomeKit or Amazon Alexa. However, these products are usually expensive and their customization is very limited. They also fall short of availability, as these previous devices are not yet available in Spain, nor in a large number of countries.

1.2 Objectives

From the previous incentive, we can see the need for an affordable and customizable home automation system that can group devices from different manufacturers, and that offers as many facilities as the systems mentioned before. For example, an user-friendly interface and a virtual assistant.

1.2.1 General Objectives

1. Design a domotic system that groups effectively home automation devices from different makers.
2. Include extra facilities, such as automation or management from a mobile application.
3. Make the system modular, extensible, safe and fully customizable by the user.
4. Make the system accessible and adaptable, that is, having the ability to use it with an attached screen, an external screen or only using the voice.

5. Make the common processes (adding devices, configuring them, making automation rules) seamless and easy.

1.2.2 Specific Objectives

1. Integrate a home automation system in a embedded system, like a Raspberry Pi.
2. Integrate a voice assistant in the same embedded system as the home automation system.
3. Explore current home automation systems and voice assistants, focusing on open-source solutions.
4. Explore automation possibilities and implement an automation service in the domotic system.
5. Explore options for managing the system from a mobile application.
6. Explore options for providing global access to the system and implement one.
7. Explore safety and privacy concerns related to the home automation system.
8. Provide an adaptive and responsive user interface, usable on touch and non-touch screens.
9. Connect the virtual assistant to openHAB, so it can manage the devices present in the system.
10. Test domotic devices in the final system and present an usable solution.

1.3 Structure of the Work

This work is structured in seven chapters. The objective is to introduce first all the results of my research, that is, the general and specific concepts and the most important products related to this project to provide a knowledge base in order to better understand the development of the resultant project.

Chapter 2 introduces the home automation technology. In this chapter, I explore the different concepts of home automation, its main features and its history. I also provide data and statistics about the attitude of society towards this technology. Then, I focus on home automation system design, indicating the different possible architectures that a domotic system can have.

Chapter 3 is about voice assistance, another very important part in this project. The objective is similar to the previous chapter, I explain what are the virtual assistants and, more precisely, the voice assistance technology. I give examples of where we can find virtual assistants and, in the second section of this chapter, I indicate the capabilities and services that virtual assistants can provide.

In Chapter 4, I analyze many different products related to this project. It is divided in three sections: home automation systems, home automation devices and voice assistants. In the first section, I explore the most popular home automation systems on the market, as well as other open source software. In the second one, I explore home automation devices that are made for very different purposes. I classify them by type and indicate the pros and cons for each one, and regarding their integration with openHAB. In the third section, I do the same for voice assistants, exploring the main systems currently in the market. For the home automation systems and for the voice assistants, I end their sections with a comparative table of all the options I have presented.

Chapter 5 offers a deeper insight into openHAB, a home automation system previously presented in Chapter 4. OpenHAB is a huge system worth exploring in depth, and in this chapter I introduce it, as well as its history and structure. I then explore its main concepts from a *logical* point of view, and next I offer a developer's perspective, explaining the internal organization of the software, its installation and other technical concepts.

I explain the entire project development process in Chapter 6. First, I provide an analysis of the system from a software engineering perspective, indicating product specification, system analysis and system design. Then, I describe the implementation process from the installation of the system. This chapter is mainly technical, and in this section I provide snippets of code that I have used in the project. Also, the appendix A is related to this chapter, where i include the full script that composes the voice assistant.

This work ends with Chapter 7. Here I analyze the obtained results and give ideas for future developments based on this work. I also analyze the fulfillment of the specific objectives specified in this chapter.

Chapter 2

Home Automation

Home automation, also known as domotics, has been a recurrent topic in Computer Science that has become a reality in the last decades, thanks to the growth and decrease in the price of embedded systems and wireless technologies, that have permitted to create distributed systems, the heart of this technology.

In this chapter, I am going to analyze this technology and its current state, including its implementation in commercial products.

2.1 What is Home Automation?

Although science fiction has represented the idea of smart houses since the past century, including in them an intelligence able to respond to all the dweller's needs and desires, it has never felt as close to real world as today.

The basic idea of home automation is to employ sensors and control systems to monitor a dwelling, and accordingly adjust the various mechanisms that provide heat, ventilation, lighting, and other services. By more closely tuning the dwelling's mechanical systems to the dweller's needs, the automated *intelligent* home can provide a safer, more comfortable, and more economical dwelling.[64] For example, the automated system can determine the intensity and direction of the sunlight, and adequate the house according to its condition (which would include closing the blinds and adjusting the air conditioner).

Unlike many may think, we don't actually need a very modern house, since advanced systems can be perfectly integrated in older, traditional buildings. This fact makes domotics a real possibility in every situation. In fact, the number of home automation systems installed in Europe is expected to reach around 29 million by 2019.[54]

Figure 2.1: Example of a smart home with security-oriented devices

Therefore, a Smart Home is expected to meet the following applications.

- Temperature control, including heating, air conditioning and air ventilation.
- Lightning control.
- Occupancy detection.
- Power control.
- Security system, including theft, or smoke or fire detection.
- Baby and pet care.
- Basic health control, like water and air monitoring.

There is not an exact point where we can set the beginning of the domotics as a real concept, but during the last century there has been some remarkable efforts, and even before. In 1898, Nikola Tesla created a wireless control for a toy boat, the first of its kind [7]. That marks the beginning of wireless technologies, one of the fundamental parts of Home Automation.

Figure 2.2: The Clapper, a sound-activated switch

In 1975, after lots of appearances of the idea of home automation in films, the first general purpose home automation technology, called X10, was developed. X10 defines a protocol for communication between electrical devices, which uses power line wiring for signaling and control, where the signals involve brief radio frequency bursts representing digital information. Therefore, it also defines a wireless radio based protocol. Surprisingly, the X10 technology is still widely used and available, with millions of units in use worldwide.

However, it was not until 1984 that the word Smart Home appeared, invented by the *American Association of House Builders*. After that, different inventions rapidly followed one another, with devices such as
The Clapper (which was operated through sound, like a clap or a bark) and interest from the biggest technological companies, like Microsoft.

Home Automation has not stopped gaining ground on our homes and now it is experiencing one of the best moments in its lifetime, with the unstoppable growth of the Internet of Things (IoT) and the simultaneous development of Artificial Intelligence for the general public, with the biggest companies, like Google and Apple, investing millions of dollars on it. Devices like Amazon Echo and Google Home, or assistants like Siri, Cortana, Google Assistant and Amazon Alexa are a good representative of this trend. I will talk in depth about them in the following chapters.

We have always imagined that Smart Homes would bring us a whole world of benefits. And that is partly true, but they have ended up offering benefits that no one could imagine some decades before, when matters such as energy savings were not as important as today. These benefits are

responsible for their increasing popularity, and they can be summarized in the following points:

- **Control anywhere:** Smart Homes can be completely controlled anywhere in the world from smart phones or other devices with Internet connection, so we can know the status of our devices at any time. That would allow us, for example, to stop worrying when staging out of home thinking if we have left the air conditioning on.
- **Safety:** there are tons of security systems ready to work on Smart Houses. They are capable of monitoring the people going in and out of home and send alerts to the owners if necessary. Like many other devices, there are also smart locks for the door and cameras that we can control from our smart device.
- **Accessibility:** Smart Homes can increase a lot the quality of life of elderly or disabled people, as they can be managed via voice commands, making the interaction much easier to people which is not experienced with computers and improving their independence.
- **Energy efficiency:** one of the main goals of Home Automation is to work with the least amount of energy needed, and a big part of the research in this field is going in this direction. There are induction cooktop stoves that can be powered on only if there is anything placed over them (and even get the perfect cooking, powering off themselves)[13] or heating systems that power on and off depending on the weather and inner conditions of the home, or even a faucet technology that can maximize shower water usage by shaping the individual droplets of water, so the experience feels almost the same but with less water usage.
- **Money saving:** the last point leads to another benefit: saving money. Smart Homes can use less energy and water, making a big difference in how much we pay at the end of the month. Reports show that the savings on the energy bill for this reason range from 10% to 30%.[13]
- **Comfort:** Smart Houses can also help save time. Today, when everyone is trying to make the most of their free time, this technology is capable of doing housework, so that people can spend their time on things they enjoy most, or simply gain time to spend with their families.

This range of benefits has made possible to see home automation systems in many homes, but also in offices. Now, almost every new house that is built is prepared for domotics, including Internet access points in every room, a

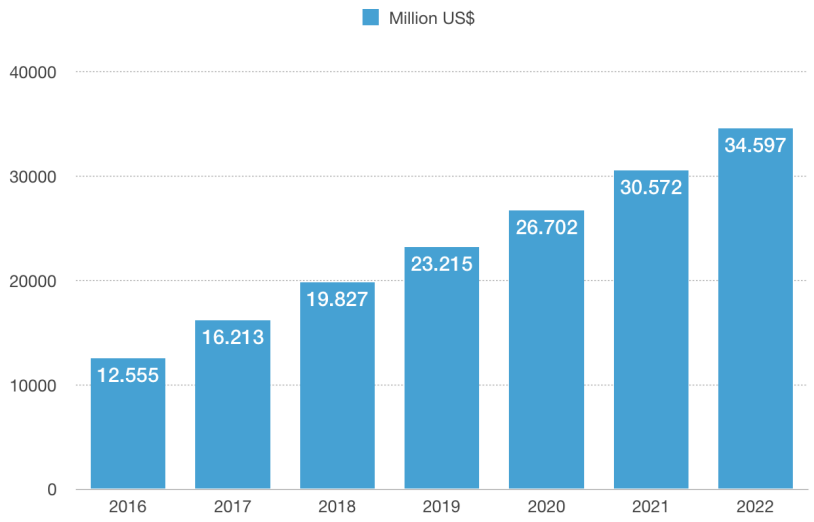

Figure 2.3: The Smart Home Market revenue from 2016 to 2022 in the US[55]

big amount of plugs, and a lot of space to extend its capabilities in a future. Indeed, the global home automation and security control market is expected to reach 12.81 billion dollars by 2020.[46] The following charts is a perfect example of how rapidly is growing the Smart Home sector and how powerful it is at this moment, showing the data for the most important Smart Home market at this moment: the United States.

Predictions are not bad either: they show that this trend will continue in the coming years, reaching 34.5 million of the US dollars, and this is just in the United States, although there will be similar situations in the rest of the world.

2.2 Home Automation System Design

After a look at the definition and history of Home Automation and its benefits, I am going to explain how these systems are usually organized. There is more than one valid way, and it will always depend on the requirements and conditions of the user, the home environment and of course the capabilities of its components.

First of all, from all the elements present in a Smart Home Environment, we can mainly distinguish the following ones:

- **Controller or controllers:** which are usually devices in charge of

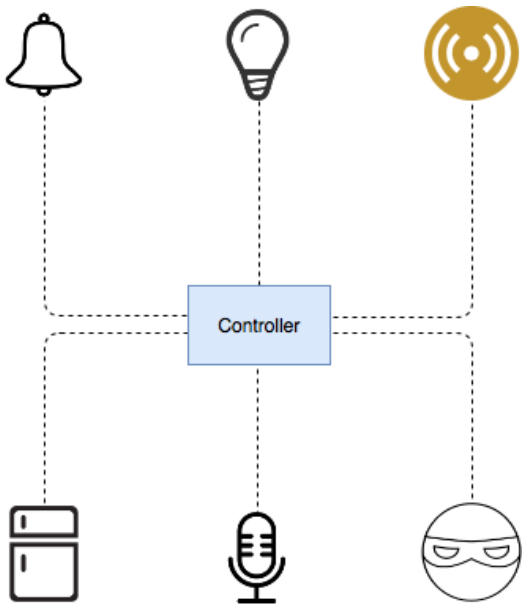

Figure 2.4: Centralized Smart Home structure

processing the data and take decisions, as well as communicate the devices between them and with other controllers, if any.

- **Sensors:** they are devices that are capable of perceiving changes in their environment by different means (audio, video, movement...). Examples of sensors are motion sensors or microphones.
- **Actuators:** these devices are the opposite of the sensors. They can inform of events, but they can also make changes in their environment. An actuator could be a light bulb or a speaker.
- **Communication mediums:** this is what devices and controllers use for communication. They can use the power grid, or by wireless protocols, like WiFi or Zigbee.

The figure 2.4 is a good example of a basic organization for Home Automation[15]. This organization has a name, indeed: centralized architecture.

2.2.1 Centralized Architecture

In a centralized Smart Home Environment architecture, the Control System, which is realized by means of a computer system, is in charge of acquiring data from sensors, providing a user interface, and executing the control

algorithms and sending instructions to actuators.[63] In the example in the figure 2.4, the sensor, in the upper right corner, could represent a smoke detector that can trigger the alarm (the actuator) to alert the householders.

The controller is often called Home Gateway, and in this case it is the central computer. It is also responsible of making accessible the system via Internet, as well as providing services to the home residents. An option to increase its performance while maintaining the same architecture is to limit the functionalities of the Home Gateway to data acquisition, software interfacing with domotic devices and basic processing, and to delegate to more powerful servers outside home the most part of the processing.

If they are placed in a powerful system, it is probably beneficial to use this architecture and get the maximum performance, which is ideal in big, complex systems.[44] However, there is only a controller and the system fully depends on it. If it failed, the whole system could be affected, which is a major issue in a Smart Home Environment.

This is the most popular architecture in home automation, partly because product manufacturers tend to centralize communications between their intelligent devices in a hub or gateway of the same brand, which users need to install to operate the rest of the devices.

2.2.2 Decentralized Architecture

In this case, there is more than one controller in the system. They are interconnected with a bus, so each one of them can interact and communicate with the rest. An example system that could use this architecture is a system with smart devices connected to hubs from different makers, and all of them interconnected thanks to a system that can work with all of them.

In the figure 2.5, we can spot three different controllers. Although the example shows a simple configuration, with this architecture it would be possible to interconnect big centralized systems, making one even bigger. Each controller is an independent system, and in this case they represent a lightning system, a voice assistant and a smart system for the garage.

2.2.3 Distributed Architecture

In a distributed Smart Home Environment architecture, the Control System software is conceptualized and implemented as a distributed computing system, that is, a series of intercommunicating devices working together to achieve an end, which in this case is running a Smart Home system. The integration and interoperability of heterogeneous domotic devices is achieved by an intermediary software layer called *middleware*.[63]

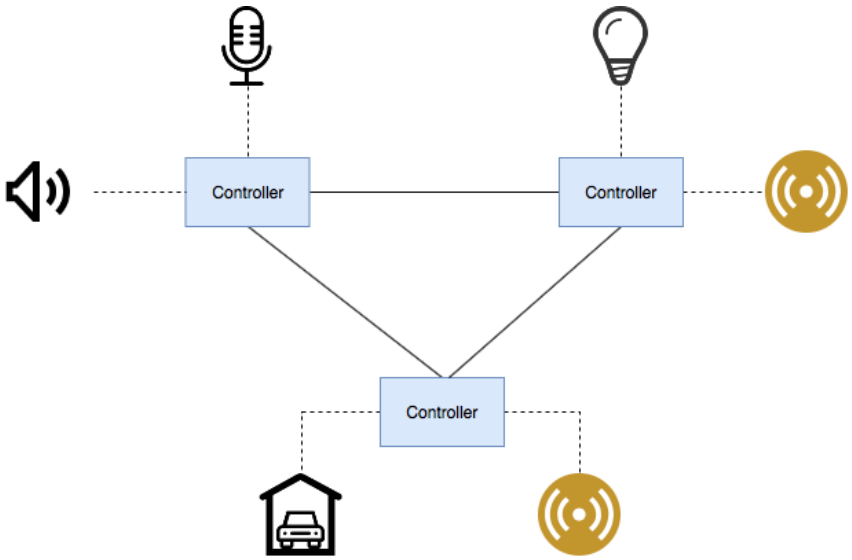

Figure 2.5: Decentralized Smart Home architecture

The distributed architecture benefits from the computational resources of smart devices to integrate software components into the nodes of the Home Automation network, which produces a big increase in autonomy and modularity[44]. However, the cost of this architecture is significantly higher compared to the centralized architecture, and therefore it is hard to achieve a fully distributed architecture. For this reason, this architecture is often applied conceptually, while still physically centralized into the Home Gateway.

In the figure 2.6 we can see an example of this architecture applied over a similar example as the one shown in 2.4. In this case, we can distinguish 4 independent but intercommunicated devices. For instance, the one at the upper right corner could contain a motion and a sound sensor, that could activate the alarm system located in the bottom right corner. The most important part about the distributed architecture is that each device acts as a controller as well, so there are not independent controllers anymore.

2.2.4 Hybrid Architecture

This architecture is a hybrid of the architectures mentioned above. In a system that follows this architecture, we may find a central controller (such as a centralized system), or a set of them (as the decentralized system), but also the end devices are controllers themselves, as it happens with distributed systems.

The main benefit of this architecture is that the devices are able to

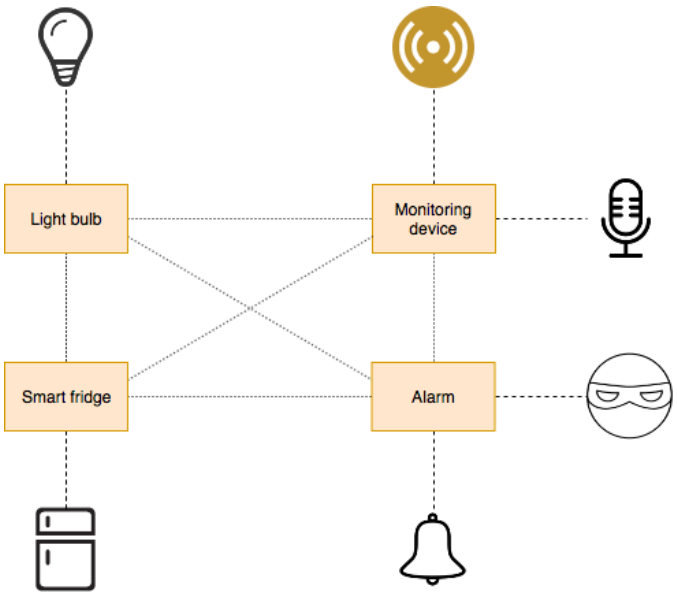

Figure 2.6: Distributed Smart Home architecture

retrieve, process and communicate the information they get directly between them, so it does not have to pass through the controller. On the downside, this architecture could create an unnecessary mess in our system.

In further chapters, I will explore more in depth Home Automation applications and technologies, showing different devices and their usages. Finally, I will use this knowledge to create a home automation controller, which will be functional in a real environment.

Chapter 3

Voice Assistance

We spend so much time using devices that have integrated voice assistants that we usually forget how incredibly fast they have evolved. Nowadays, they can recognize thousands of words and expressions really fast, and they are even capable to imitate emotions. What is more, they fit in a pocket. But the reality was totally different just a couple of decades ago. From the IBM Shoebox to Siri, in this chapter I will explore the fundamentals of voice assistance.

3.1 What is Voice Assistance?

Voice assistance is the result of another form of interaction between humans and computers.[8] The Voice User Interface (VUI), which has the voice assistants as a result, allows a user to interact with computer or mobile or other electronic devices through speech or voice commands. Thus, VUI is an interface of any speech recognition applications.

Therefore, a voice assistance, also known as virtual assistant, is an application program that understands natural language voice commands and can perform tasks or services for an individual. Its expansion has been truly remarkable in the last few years, to the point that we can see devices that exclusively work as virtual assistants, with integration with many other services. Its real usefulness in society, though, remains to be seen, as this field is commonly viewed with skepticism and mistrust, and the fact of talking to a machine as if it were another human being remains an obstacle to overcome.

As I mentioned, voice assistants are now present in plenty of platforms:

- **Smart speakers:** Google Home (Fig. 3.1), Apple HomePod, Amazon Echo, Movistar Home.

Figure 3.1: Google Home, a smart speaker integrated with the Google Assistant

- **Mobile operating systems:** Siri on iOS, Google Assistant on Android, Bixby on Samsung phones.
- **Desktop operating systems:** Siri on macOS and Cortana on Windows 10.
- **Smartwatches**: Apple Watch, Google Wear OS.
- **Cars:** Apple CarPlay, Android Auto.
- **Televisions:** Siri on Apple TV (Fig. 3.2) and the voice assistant in Samsung Smart TVs.
- **Inside mobile apps:** EVO Assistant in the mobile application of the Spanish bank EVO.

The history of voice assistance goes back to 1961, when IBM introduced the IBM Shoebox.[58] This was a very innovative product at that moment. Although it was not suitable for commercial use, it did mark the beginning of a revolution, the fruits of which we can now see.

The Shoebox was capable of recognizing 16 spoken words, including ten digits from 0 through 9. When a number and command words such as *plus*, *minus* and *total* were spoken, Shoebox instructed an adding machine to calculate and print answers to simple arithmetic problems. It classified the electrical impulses generated from a microphone according to various types of sounds and activated the attached adding machine through a relay system.[20]

Figure 3.2: Apple TV and the Siri Remote, which has a microphone to interact with the assistant

Later on, there were more attempts from the research field, as the HARPY Speech Recognition System from the Carnegie Mellon University, in 1976.[65] Is could recognize about 1000 words.

Nevertheless, it was not until 1990 that the first speech recognition for consumers appeared: the Dragon Dictate. Seven years later, the same company presented the Dragon NaturallySpeaking, which introduced continuous speech recognition as a novelty. They led the way with competent voice recognition and transcription. This field attracted the attention from big companies of that time, and Microsoft began working on their own assistant: Clippy, in the Microsoft Office suite. In spite of the fact that this was not a voice assistant exactly, it showed how natural language could be interpreted and used in order to allow the human-computer interaction. It was quite unpopular and Microsoft decided to end it on 2001, but its impact was huge for the assistants that followed it. A bit before Windows XP, Microsoft introduced the speech recognition feature in their Office XP suite.

With the launch of Siri in 2014, Apple marked the modern era of voice assistants. For the first time, people could fit a full functional voice assistant in their pocket. And most importantly, Siri reached a wide audience and began to popularize this technology. Siri was able to make searches on Internet and reproduce the results, to set reminders and events in the calendar or to call any contact by its name, between many others. In addition, it included a layer of *natural interaction* with the user, being able to respond to any other phrase as any other human would (even to sentences that were

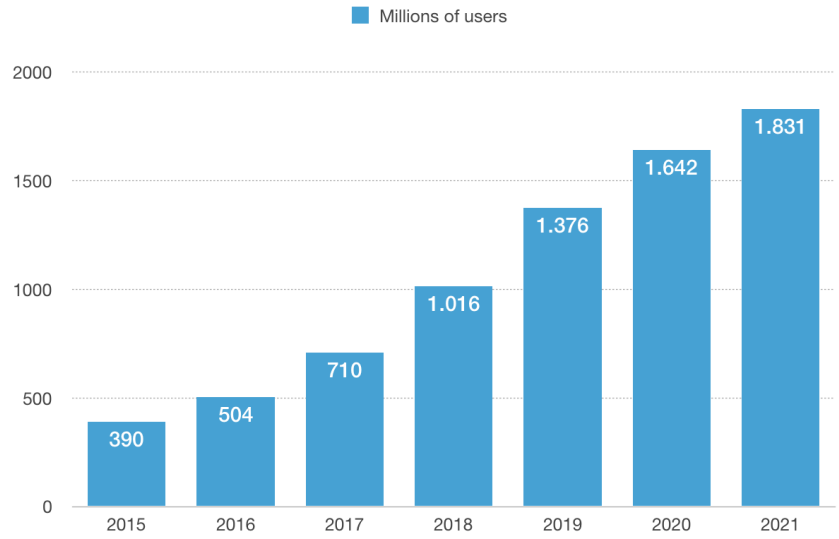

Figure 3.3: Estimated number of users of virtual assistants worldwide [53]

not commands, like *How do you feel today?* or *Tell me a joke*).

Google, with Google Now, and then Microsoft and Amazon, with Cortana and Alexa respectively, followed this trend, even improving on what Siri failed, and in the case of Microsoft, making a voice assistant available on PCs as well. Then, in 2014, Amazon introduced Echo, the first smart speaker of all time. It was just the beginning of what we now call the *Smart Speaker Revolution.*[58]

After the Echo, Apple and Google followed with the HomePod and Home, respectively. In fact, Google Home has been recently launched in Spain, and the HomePod is not available yet, as an example of how recent this technology is. Its number of users is expected to continue to grow, and even faster than it has already done.[53]

3.2 Services Voice Assistants Provide

The range of services provided by voice assistants is constantly becoming bigger, as they are a booming technology that is constantly receiving new updates. The following are shared by most of the virtual assistants currently:

- Provide information, such as weather forecast, routes to any point in a map or general knowledge.
- Manage components in a Home Automation environment.

- Interact with media content, such as music and video (which is commonly integrated with streaming services, like Netfix or Apple Music).
- Make phone calls and send instant messages.
- Manage the personal agenda.
- Provide accessibility indications.
- In call centers, they complement or replace the customer service by humans.

We are nowadays in the first stages of this new technology, that combines artificial intelligence, machine learning, voice recognition and human-computer interaction. Google is the company that apparently has done the biggest advancements and, in fact, they have recently introduced Google Duplex, a technology capable of almost perfectly simulating a human speech, which can be used for a wide range of purposes, such as ordering food or making an appointment with a hairdresser. This would be a new service to include in the previous list.

They are also providing very useful tools to developers and makers, like the Cloud Speech-To-Text API. I will come back to this technology in the following chapters, as it will be an essential part to achieve my objective, the creation of a voice-driven home automation controller, a service that can also be seen in the previous list.

Chapter 4

Product Analysis

The aim of this chapter is to provide a detailed analysis of the devices more closely related to this project, now that we have a clearer idea about its main pillars. I will go into many of the available commercial devices and software in the fields of home automation, voice assistance and smart devices.

4.1 Home Automation Systems

This section covers all the hardware and software systems related to home automation. As we will see, there are lots of solutions with very different purposes: while there is Amazon Alexa, a full hardware and software system that integrates other home automation systems, we can also find pure online solutions, like the automation platform IFTTT. Sometimes, home automation systems are built underneath a virtual assistant, as it happens with Amazon Alexa, so some devices are going to appear in this section and in the next one. However, they will analyzed from two different perspectives, as having a good virtual assistant does not mean having a good home automation system.

4.1.1 Philips Hue

Philips Hue is a personal wireless lighting system aimed at the smart home. In combines LED light bulbs, LED strips and other lighting devices, and sensors that can be configured in their mobile app, so they can modify the home lighting based on a set of rules. There is a wide range of products, including color and only white lights, so users can build a pretty customizable lighting experience.[40]

The system requires a bridge connected to the Internet (called Philips Hue Smart Hub) in order to work. This is because the Hue devices do not

Figure 4.1: A Philips Hue dimmer switch, three color light bulbs and the Hue Smart Hub

use WiFi in order to communicate with the bridge, but the system needs to have WiFi to be controllable from a mobile phone. Thus, it follows a centralized architecture. Moreover, Philips does not provide any type of assistant or external interface to manage the system apart from the mobile application by default, although Hue works with the most popular home automation systems, like Alexa or Apple HomeKit, that provide much more flexible home automation management.

4.1.2 LG SmartThinQ

LG SmartThinQ groups the range of Wi-Fi enabled home appliances made by the company LG, including refrigerators, dishwashers, vacuum cleaners or air purifiers, between others. As of September 2017, they were the most extensive range of devices of their kind.[27]

Unlike Philips Hue, SmartThinQ devices do not require a bridge to work. They can be controlled from the mobile phone and, in some cases, like in the refrigerators, they include a touchscreen to interact with the device. However, LG does not provide any extra device or virtual assistant to interact with them, though they are manageable through Amazon Alexa and Google Assistant. A standard setup with this system will follow a hybrid architecture, as some devices are also their controllers, but there can also be external controllers.

4.1.3 Samsung SmartThings

Samsung SmartThings is a home automation system composed by a series of applications for the Samsung mobile phones, Samsung TVs and Samung refrigerators. It is even possible to do small management tasks from Samsung smartwatches, called Galaxy Gear. It uses the cloud to synchronize all the applications, in order to have the most recent information in all of them. This makes it necessary for the user to have a Samsung account.[48]

Unlike the previous systems, SmartThings is not a specific system for a range of devices from the same maker, but it is more aimed to provide an effective interconnection between devices from different makers, as long as they are compatible with their system. The SmartThings Smart Home Hub is necessary in order to use Samsung SmartThings. It is a bridge that supports common home automation protocols, like Zigbee or Z-wave, essential to manage some devices that only use these protocols. It also provides comprehensive automation options.[49] The usage of the Hub makes the architecture of this system centralized.

Furthermore, SmartThings is not yet compatible with many commercial devices, and the restrictions imposed by Samsung forces the user to stick to their environment. In addition, the system is not open source, so making any modification apart from the ones that Samsung allows is impossible. Also, users are forced to purchase the Smart Home Hub, which makes it necessary to have an additional device, unlike other similar systems. The system is compatible with Bixby, the virtual assistant from Samsung.

4.1.4 Google Home

Introduced at Google I/O 2016, the annual Google developer conference, *Google Home* is the name of Google's smart speaker, which is Google's biggest insight into home automation technology. Its aim is to work with all the possible smart home devices, so it follows the same idea as the Samsung SmartThings system, being a *maker-independent* system, as long as, of course, devices are compatible with it. Also, Google Home brings all the functions of Google Assistant to the smart speaker.

The main difference with SmartThings is that this system is mainly voice-driven. The home automation layer is pushed down to just one more function of the virtual assistant, and Google does not even provide a graphical interface to manage the smart devices. Anyway, normally the makers of each device provide a mobile application from where users can manage their devices in a more user-friendly interface, but having a centralized view is a desirable feature. On the bright side, all Google devices that support Google Assistant can automatically control smart home devices.

The number of compatible devices with the Google Assistant, unlike SmartThings, is very high, and almost any new smart home device is tagged as compatible with it.

4.1.5 Apple HomeKit

HomeKit is the result of Apple's efforts to create a home automation environment adapted to its devices. It has been also made to work with a wide

range of devices, but in this case, Apple included some notable security policies, with the goal of achieving the highest security and privacy. In fact, all HomeKit devices need to be approved by Apple first.

On iPhone and Mac computers (starting with macOS Mojave), Apple includes an application called Home, which displays all of the smart home devices in a convenient way and lets people organize and manage them. In addition, it is also possible to establish automation rules, based on the user's location, time of day, actions or even occupancy of the house. Furthermore, Apple also provides integration with their personal assistant, Siri. As it happens with the Google Assistant, all Apple devices configured with the same Apple ID will have the same information automatically synchronized.[5] Usually, systems made under Apple HomeKit will follow a decentralized architecture.

Although this is a very comprehensive home automation system, the number of compatible devices is not as high as in other options. Apple has been lately working on promoting their home automation system and their assistant by introducing the HomePod, their smart speaker with Siri.

4.1.6 Somfy

Somfy is a French company founded in 1960, which since the 1980s has been devoted to the construction of home automation systems. They have implemented their solutions in important places, like the United Nations Headquarters or the Vancouver Convention Center and have created their own home automation technologies, such as *Radio Technology Somfy (RTS)* and *Somfy Digital Network (SDN)*.[50]

Their range of products goes from control devices (as hand-held remotes, mobile applications and wireless switches) and sensors (sunlight, temperature, wind) to blinds, lighting systems and other smart home utilities. Unfortunately, the control devices only work with Somfy devices. The system follows a hybrid architecture, where each device is a controller, but there can also be extra controllers, like the hand-held remotes.

In addition to the previous domotic systems, which are proprietary, there are also open source and more customizable solutions that, although they may require more time in their configuration, are much more adaptable to the needs of the user. I will explore three of the most popular: openHAB, Home-Assistant.io and Jeedom.

4.1.7 OpenHAB

OpenHAB, the acronym for Open Home Automation Bus is an open source, technology agnostic home automation platform which runs as the center of the smart home. [35] This means that its aim is to integrate different home automation systems into a single one. It allows the user to configure almost every aspect of the system, providing a common interface and a uniform approach to automation rules.

In its most recent version, openHAB 2, it has implemented new user interfaces that automate many processes, so it is almost unnecessary to write a single line of code, making the system more attractive to all types of users. OpenHAB needs to be installed in a computer that will act as a server in the local network, making the system accessible via HTTP. It also offers more connectivity options that I will explore in the following chapters. The architecture, in this case, is decentralized, as there may be multiple controllers, including openHAB itself.

OpenHAB's compatibility is somewhat limited when it comes to home automation devices, but it supports Apple HomeKit and common protocols, such as ZigBee and Z-Wave, to get rid of specific gateways from other systems.[33]

4.1.8 Home-Assistant.io

Home-Assistant.io, or simply Home Assistant, is a open source home automation platform running on Python 3.[19] Based on a distribution called *Hass.io*, it creates a secured local server in the computer where it is installed. It is accessible via HTTP and also includes a web user interface that will automate the process of discovering and configuring devices. In terms of functionality, it is very similar to openHAB, although it might be a little simpler for some users, as it uses the YAML syntax for configuration, while openHAB has its own.

Its functionality is organized in *Components*, the name that Home Assistant gives to any add-on, which will add a compatibility layer with a device, system or service. They are fully backed by the Home Assistant community and they are similar in number and type to what openHAB provides, but with very interesting additions, including Wink and Arduino.

4.1.9 Jeedom

Jeedom is a open source, multi-protocol, autonomous and customizable home automation software.[24] It is aimed for individuals and professionals, and provides custom support for both. They also sell what they call

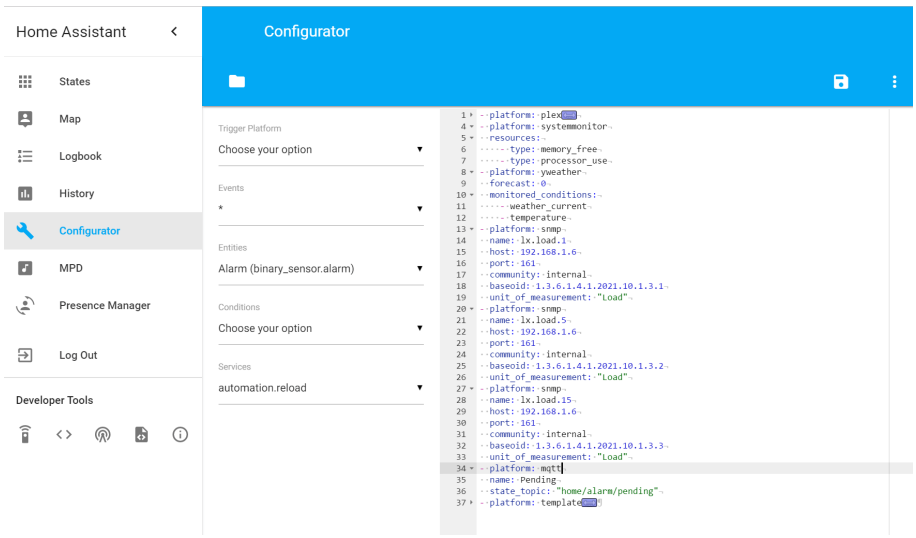

Figure 4.2: Home-Assistant.io web user interface

Boxes, which are small computers with Jeedom pre-installed, although their software can be installed on any Linux system. Jeedom also provides mobile phone apps for Android and iOS, which connect to the Jeedom system by scanning a QR code. As in the previous platforms, they also provide the *Jeedom Market*, from where users can add new features to their system.

Also, there are different Jeedom versions, which are called *Service packs*. There is the free and open source version, which includes the lowest number of functionalities. Then, the other versions can be purchased, although some of them come with their Boxes, and they include dynamic DNS and HTTPS, the mobile application for free or more plugins offered, among other things.

Although the free version is fully open source, the limitations that it has and the obligation to pay in order to have the *full experience*, could annoy some users and make them lean towards other platforms.

It seems difficult to directly compare these home automation systems, but all of them share characteristics that we can contrast. The table 4.1 represents a comparison of the features that I consider most important.

System name	Developer	Free	Open source	Architecture	Requirements	Extra software	Extra hardware	Compatibility
Hue	Philips	No	No	Centralized	Hue Smart Hub	Mobile app	Hue devices	Only for Hue devices Integrable in Alexa, Google Assistant, HomeKit
SmartThinQ	LG	No	No	Hybrid	None	Mobile app	SmartThinQ devices	Only for SmartThinQ devices Integrable in Alexa and Google Assistant
SmartThings	Samsung	No	No	Centralized	Smart Home Hub	Mobile app	Samsung smart home devices	Compatible with devices from different makers Integrated in Bixby
Assistant	Google	No	No	Centralized	A device with Google Assistant	Google Home app	Google Home smart speaker	Huge compatibility with devices from different makers
HomeKit	Apple	No	No	Decentralized	An Apple device	Home app	Apple HomePod	Compatible with devices from different makers Integrated in Siri
Somfy	Somfy	No	No	Hybrid	None	Mobile app	Somfy smart home devices and controls	Only for Somfy devices
openHAB	The openHAB Community and the openHAB Foundation e.V.	Yes	Yes	Decentralized	A computer with Internet connection	myopenHAB openHABian Mobile app	None	Compatible with devices and services from different makers, fully customizable
Home Assistant	The Home Assistant community	Yes	Yes	Decentralized	A computer with Internet connection	iOS app Hass.io	None	Compatible with devices and services from different makers, fully customizable
Jeedom	Jeedom SAS	Partly	Partly	Decentralized	A computer with Internet connection or a Jeedom Box	Mobile app	Jeedom Boxes	Compatible with devices and services from different makers

Table 4.1: Comparison between different home automation systems

4.2 Home Automation Devices and Related Services

In this section I will explore different devices, services and protocols that may be suitable for the outcome of this project. The section presents a brief description of each device and a comparison between the same type of devices. These devices are usually handled in the same way in open source home automation systems. For example, in openHAB they are called bindings[35], and each of them is installable over the base system. As the main idea is to use open source technologies in this project, I will focus on compatible devices, and analyze them based on their integration with openHAB. Therefore, I will talk about bindings, which is the abstraction layer that openHAB employs for devices, services and protocols.

4.2.1 Alarms

DSC PowerSeries Alarm System

The DSC PowerSeries Alarm System is a popular do-it-yourself home security system, which can be monitored and controlled remotely through a standard web-browser or mobile device.

Pros:

- Supporting a DIY Alarm System is acceptable due to the range of users we expect to cover
- Communication via API (OpenHAB binding)

Cons:

- Availability of this product outside USA

4.2.2 Amazon Dash Button

The Amazon Dash Button is a cheap and small Wi-Fi connected device to order products from Amazon with the simple press of a button. This Binding allows to integrate Dash Buttons into the controller.

What to consider:

- Privacy concern: The Dash Button will try to contact the Amazon servers every time the button is pressed. Details about this can be read in this section of the documentation.

- The Binding uses Pcap4J in order to capture ARP and BOOTP requests send by the Amazon Dash Button. Buttons will hence only be usable within the same network as your openHAB instance.

Pros:

- Usability
- Communication via API (OpenHAB binding)
- Easy to configure

Cons:

- Not open source
- Manual device addition

4.2.3 AV Receivers

Denon

The openHAB Denon Binding allows interaction with Denon AV receivers.

Pros:

- Plenty of available settings
- Communication via API (OpenHAB binding)

Cons:

- Not a common device
- Manual device addition

Marantz

Denon binding also seem to work with Marantz devices.

Onkyo

This binding integrates the Onkyo AV receivers. Binding should be compatible with Onkyo AV receivers which support ISCP (Integra Serial Control Protocol) over Ethernet (eISCP).

Pros:

- Communication via API (OpenHAB binding)
- Usually cheaper than Denon devices

Cons:

- Quite limited product support

4.2.4 Digital Media Players

Google Chromecast

The binding integrates Google Chromecast streaming devices. It not only acts as a typical binding, but also registers each Chromecast device as an audio sink that can be used for playback.

The binding currently supports the "classic" Chromecast that is an HDMI dongle as well as the Chromecast Audio, which only does audio streaming and offers a headphone jack.

Chromecast devices are discovered on the network using UPnP. No authentication is required for accessing the devices on the network. They can also be manually added.

Pros:

- Common device
- Automatic discovery
- Communication via API (OpenHAB binding)

Cons:

- Not many actions can be performed
- Current library might be problematic because of how it works

Kodi

Kodi is a free and open source software media center for playing videos, music, pictures, games, and more. Kodi runs on Linux, OS X, BSD, Windows, iOS, and Android. It allows users to play and view most videos, music, podcasts, and other digital media files from local and network storage media and the internet. The Kodi Binding integrated Kodi media center support with openHAB, allowing both controlling the player as well as retrieving

player status data like the currently played movie title.

What to consider:

- Needs some initial configuration in Kodi.

Pros:

- Auto-discovery feature
- Fully open source
- Extremely cheap and useful solution, capable of running in almost every device
- Communication via API (OpenHAB binding)

Plex

This binding supports multiple clients connected to a Plex Media Server. With this binding, it's possible to dim the lights when a video starts playing, for example. Most changes are pushed to the binding using web sockets. Polling (and the corresponding refresh interval) is only applicable to the online/offline status of clients.

What to consider:

- It is necessary to configure the username and password (or to use the Plex token) in order to make it work.

Pros:

- Plex is a free and easy to use media centre, available for many platforms
- Wide control of the Plex Media Server from OpenHAB
- Communication via API (OpenHAB binding)

Cons:

- Plex is not fully open source

4.2.5 Garage Door Control

Chamberlain MyQ

Chamberlain MyQ system allows you to connect the garage door to the internet to be controlled from anywhere using a smartphone. Using this API, The Chamberlain MyQ Binding can get the status of the garage door opener and send commands to open or close it.

Pros:

- Easy to control, only needs the user and password to start working
- Communication via API (OpenHAB binding)
- Availability in Europe

Cons:

- High price (Starting at EUR 200, Amazon Spain price)

4.2.6 Garden Care

Gardena

This binding allows to integrate, view and control Gardena Smart Home devices in the openHAB environment.

Pros:

- There are not many smart solutions for garden care, but Gardena can fulfil any needs
- Communication via API (OpenHAB binding)

Cons:

- Needs an account to discover the devices, though the discovery is fully automatic once the account is set

4.2.7 Lighting

Philips Hue Lighting System

This binding integrates the Philips Hue Lighting system. The integration happens through the Hue bridge, which acts as an IP gateway to the ZigBee

devices.

What to consider:

- The Hue Smart Bridge is required.
- Almost all available Hue devices are supported by this binding. This includes not only the "friends of Hue", but also products like the LivingWhites adapter.
- Devices need to be registered with the Hue bridge before it is possible for this binding to use them.
- The Hue bridge is discovered through UPnP in the local network. Once it is added as a Thing, its authentication button (in the middle) needs to be pressed in order to authorize the binding to access it. Once the binding is authorized, it automatically reads all devices that are set up on the Hue bridge and puts them in the Inbox

Pros:

- Philps Hue is a beautiful and easy way to take advantage of home automation and smart devices. Its devices are very common and useful, being the final user its main target
- Compatibility with many other systems (Alexa, Apple HomeKit, etc.)
- Communication via API (OpenHAB binding)

Cons:

- Need an extra device (the Hue Smart Bridge) to communicate with the rest of them

LIFX LED Lightning

This binding integrates the LIFX LED Lights. All LIFX lights are directly connected to the WLAN and the binding communicates with them over a UDP protocol.

What to consider:

- The binding is able to auto-discover all lights in a network over the LIFX UDP protocol. Therefore, all lights must be turned on

Pros:

- LIFX is one of the most popular alternatives to Philips Hue
- No need of any extra device
- Communication via API (OpenHAB binding)

Cons:

- More expensive that Philips Hue (EUR 65 vs. EUR 45, Amazon Spain prices)
- Less compatibility than Philips Hue

MiLight, EasyBulb, Limitless LED and iBox

This binding is for using Milight, Easybulb or LimitlessLed bulbs and the iBox.

What to consider:

- The binding supports Milight/Easybulb bridges from 2014+, iBox from 2016 and iBox2 from 2017 and their respective bulbs. The Dual White bulbs from 2015 and the new generation of Dual White bulbs is supported. RGB/White from 2014 and the new generation RGB/White from 2016 as well as RGB/Cold, warm white and iBox bulbs work.
- All supported bridges can be discovered by triggering a search in openHAB's Inbox. Found bridges will show up and can easily be added as things. Unfortunately, Milight like bulbs have no back channel and cannot report their presence, therefore all possible bulbs are listed as new things after a bridge has been added.

Pros:

- Extremely affordable products, these bulbs are one of the best options for the user due to their attractive price and good performance (EUR 12,50 for RGB + Warm White MiLight, GearBest Spain prices)
- Easy discovery and API support for recent devices
- Seems that they can work directly with openHAB with no extra devices
- Communication via API (OpenHAB binding)

Cons:

Figure 4.3: The D-Link DCH-S150 motion sensor

- Worse performance than Philips Hue or LIFX
- Messier discovery than the other options

WiFi LED

This binding is used to control LED stripes connected by WiFi. These devices are sold with different names, i.e. Magic Home LED, UFO LED, LED NET controller, etc.

Pros:

- WiFi LED stripes are used to improve the lightning of some areas, creating an artistic effect. For example, on the back of a TV. They are cheap and very easy to configure.
- Communication via API (OpenHAB binding)

4.2.8 Sensors

D-Link Smart Home Devices

OpenHAB supports the D-Link DCH-S150 (figure 4.3), a WiFi motion sensor.

Pros:

- Easy to install and very useful for a smart home, no special configuration needed

- Communication via API (OpenHAB binding)

EnOcean Sensor Solutions

EnOcean provides reliable and self-powered wireless sensor solutions for the Internet of Things. This binding allows openHAB to monitor and control EnOcean devices through the EnOcean USB 300 gateway. EnOncean sensors include rocker switches, environment sensors and contact sensors.

What to consider:

- We need a USB300 stick to control EnOcean devices

Pros:

- Variety of sensors available
- Can work together with many other devices
- Communication via API (OpenHAB binding)

Cons:

- We must have extra hardware to make them work with our system (USB 300 gateway)

X10

X10 is a company that makes gadgets like cameras and sensors for a Smart Home. This binding makes it possible to control X10 devices via a server running the Mochad X10 daemon. Mochad is a Linux TCP gateway daemon for the X10 CM15A RF (radio frequency) and PL (power line) controller and the CM19A RF controller. With the current version of the binding items of type Switch, Dimmer, and Rollershutter can be controlled. The binding only uses one-way communication so no status reading

Pros:

- Relatively low price
- Communication via API (OpenHAB binding)

Cons:

- Low availability in Europe, mainly via specialised shops

- Needs extra software
- Offers less control than other devices

4.2.9 Smart TV

LG TV

This binding supports LG TV models with Netcast 3.0 and Netcast 4.0 (Model years 2012 and 2013), and with LG TVs which support the UDAP 2.0 protocol over Ethernet.

Pros:

- LG Smart TVs are a very common device that should be supported by our system
- Communication via API (OpenHAB binding)

Cons:

- OpenHAB documentation is not very specific about this binding
- It does not appear to support more modern LG televisions

Panasonic TV

This binding supports Panasonic TVs. It should be compatible with most up-to-date Panasonic Smart-TVs.

Pros:

- It is possible to control the TV completely from the system
- Panasonic TVs are a very common device
- Easy configuration
- Communication via API (OpenHAB binding)

Samsung TV

This binding integrates the Samsung TV's.

What to consider:

Figure 4.4: The Netatmo Personal Weather Station

- Samsung TV C (2010), D (2011), E (2012) and F (2013) models should be supported. Because Samsung does not publish any documentation about the TV's UPnP interface, there could be differences between different TV models, which could lead to mismatch problems.

Pros:

- Support for Samsung TVs is truly necessary, as they are one of the biggest Smart TV resellers

Cons:

- Very limited control of the TV
- It has not been tested much, so we really do not have many information about this binding and if it will work with other models

4.2.10 Temperature Control

Devices using eBUS protocol

The eBUS binding allows controlling the heating system. The eBUS protocol is used by heating system vendors like Wolf, Vaillant, Kromschröder etc. It is possible to read temperatures, pump performance, gas consumption etc.

Pros:

- One of the main purposes of this Smart Home Controller is covering heating devices, thanks to this binding it is possible.
- Communication via API (OpenHAB binding)

Cons:

- Quite complex to implement

EcoTouch Binding

The openHAB EcoTouch binding allows interaction with Waterkotte EcoTouch heat pumps.

Pros:

- Communication via API (OpenHAB binding)

MAX! Thermostats

This is the binding for the eQ-3 MAX! Home Solution. This binding allows you to integrate, view and control the MAX! Thermostats in the openHAB environment.

What to consider:

- Discovery: when the Cube is found, it will become available in the discovery Inbox. Periodically the network is queried again for a Cube. Once the Cube is available in openHAB, all the devices connected to it are discovered and added to the discovery inbox. No scan is needed to trigger this.

Pros:

- Communication via API (OpenHAB binding)
- Useful and multipurpose. For example, it is possible to track the temperature of the heating devices anytime

Cons:

- Extra device needed: MAX! Cube. It is also possible to communicate with these devices using a CUL USB Dongle rather than the MAX! Cube.

Netatmo

The Netatmo binding integrates the following Netatmo products:

- Personal Weather Station (figure 4.4): reports temperature, humidity, air pressure, carbon dioxide concentration in the air, as well as the ambient noise level.
- Thermostat: reports ambient temperature, allow to check target temperature, consult and change furnace heating status....

What to consider:

- Discovery: Netatmo Binding is able to discover automatically all depending modules and devices from Netatmo website. It is also possible to add manually devices by creating things in in the *.things file.

Pros:

- Good-looking and easy to install solutions for temperature control and temperature information
- Communication via API (OpenHAB binding)

Cons:

- Too expensive for our price range (EUR 179 for the thermostat and EUR 169 for the weather station)

4.2.11 WiFi Sockets

Orvibo S20

This binding integrates Orvibo devices that communicate using UDP. Only supports Orvibo S20 WiFi sockets.

Pros:

- Smart Plugs enable controlling and automating non-smart devices from the controller
- Orvibo offers unexpensive and easy devices for this matter
- Communication via API (OpenHAB binding)

4.2.12 Xiaomi Mi Smart Home

This binding allows openHAB to communicate with the Xiaomi Smart Home Suite. This includes the following devices:

- Xiaomi Smart Gateway v2 (with radio support)
- Xiaomi Smart Temperature and Humidity Sensor (round one)
- Xiaomi Smart Door/Window Sensor (round one)
- Xiaomi Wireless Switch (round one)
- Xiaomi Motion Sensor / IR Human Body sensor
- Xiaomi Smart Plug
- Xiaomi Smart Magic Cube
- Xiaomi Aqara ZigBee Wired Wall Switch (1 and 2 buttons)
- Xiaomi Aqara ZigBee Wireless Wall Switch (1 and 2 buttons)
- Xiaomi Aqara Smart Curtain
- Xiaomi Aqara Water Leak Sensor
- Xiaomi Aqara Wireless Switch (square one)
- Xiaomi Aqara Temperature, Humidity and Pressure Sensor (square one)
- Xiaomi Aqara Door/Window Sensor (square one)
- Xiaomi Aqara Motion Sensor (with light intensity support)
- Xiaomi Mijia Honeywell Gas Alarm Detector
- Xiaomi Mijia Honeywell Fire Alarm Detector

What to consider:

- The MiHome app is necessary in order to connect the Gateway

Pros:

- Xiaomi is an extremely affordable brand that offers many devices for Home Automation.
- Easy to install and configure

- Communication via API (OpenHAB binding)

Cons:

- Needs extra hardware (Smart Gateway) and software (MiHome app)

4.2.13 Other devices and services

Logitech Harmony Hub

The Harmony Hub binding is used to enable communication between openHAB2 and multiple Logitech Harmony Hub devices. Logitech Smart Hub devices can control other devices like Apple TV, Amazon Alexa, or Sonos devices. The Binding works as a bridge between the Harmony Hub and the devices connected to it.

Pros:

- Might be beneficial to consider it given the fact that there could be devices (with proprietary communication protocols) that we wouldn't be able to support, like Apple TV
- Communication via API (OpenHAB binding)

Cons:

- Limited API

Epson Projectors

This binding is compatible with Epson projectors which support ESC/VP21 protocol over serial port.

Pros:

- Communication via API (OpenHAB binding)

Cons:

- Seems that only business-oriented projectors are compatible with this binding

MQTT

This binding allows openHAB to act as an MQTT (MQ Telemetry Transport) client, so that openHAB items can send and receive MQTT messages from or to an MQTT broker.

What to consider:

- Implementing MQTT makes us able to use OwnTracks, a location service that uses MQTT that focuses on privacy.

NTP

The NTP binding is used for displaying the local date and time based update from an NTP server. Discovery is used to place one default tem in the inbox as a convenient way to add a Thing for the local time.

Weather Binding

The Weather binding collects current and forecast weather data from different providers with a free weather API. You can also display weather data with highly customizable HTML layouts and icons. It is also possible to install the WeatherUnderground and YahooWeather bindings.

4.3 Voice Assistants

The objective of this section is to explore the main voice assistants available commercially. As I explained previously, the services provided by virtual assistants, and in particular voice assistants, are very varied. One of them is smart home control, and many of the home-oriented voice assistants provide it. The focus on this section will be on them, the most closely related to my project.

4.3.1 Samsung Bixby

Bixby is the virtual assistant that Samsung includes in their phones, introduced in 2017, along with the introduction of the Samsung Galaxy S8 phone. It is the evolution of their previous voice assistant, *S Voice*.

Bixby is divided in three parts: *Bixby Voice*, which is the voice assistant, *Bixby Vision*, an assistant that works through image recognition, and *Bixby*

Home, a dashboard that provides different information depending on the current conditions.[47]

Bixby has been widely criticized for his intrusiveness and its lack of utility in many situations. In addition, it is only available in English and Chinese at this moment. As for home automation, it is seamlessly integrated with Samsung SmartThings, making it possible to send basic commands to devices via voice.

4.3.2 Google Assistant

Google Assistant is the name of the virtual assistant developed by Google. It was introduced in 2016, and at this moment it is available for mobile phones (with Android and iOS operating systems), laptops, TVs, cars and smart watches. It is also integrated in Google Home, their smart speaker.[18]

The most remarkable feature of this assistant is its ability to engage in two-way conversations, thanks to a powerful artificial intelligence developed by Google, and Google Duplex, a new technology that Google is developing, which will allow it to have natural conversations, mimicking the human voice.

Google Assistant is able to manage a wides range of smart home devices and in a very flexible way, thanks to its outstanding voice recognition.

4.3.3 Apple Siri

Siri is the voice assistant developed by Apple, and the one who began the revolution of voice assistance in mobile phones. Introduced back in 2011 and included in the iPhone 4S, it has been present in all the iOS devices (iPhones, iPads and iPods) since then, and lately in the macOS devices (Macintosh computers) and in the Apple Watch as well.[6] Its range of functionalities is also similar to its competitors, but with a slightly narrower range of possible interactions, which can make Siri feel a little less natural.

In later iOS versions, Apple has implemented Siri suggestions, which use the artificial intelligence that Siri provides to suggest applications to the user based in the current circumstances. Siri is able to adapt over time to the user's personal preferences by customizing search results and other responses. In the latest version of iOS, iOS 11, Siri has a much more natural voice, that can also simulate different moods.

4.3.4 Amazon Alexa

Alexa is the virtual assistant by Amazon, and it is included by default in all their Echo devices (Amazon's smart speakers), Fire TVs (digital media players) and Fire tablets. It is also available for iOS and Android devices as a standalone application. Its capabilities are very similar to those of its competitors: music playback, making to-do lists, setting alarms, playing podcasts and audiobooks, providing information, such as weather forecast and general knowledge, and of course voice interaction and home automation management. This means that, as in the previous cases, there is also a home automation system underlying the assistant in Alexa.

Alexa differentiates from the rest on its skill system. A *skill* is a functionality developed by a third-party vendor that the user can install in the assistant in order to extend is capabilities. They can be, for instance, news services or little games. Amazon is constantly encouraging the creation of new skills for Alexa between the developer community.[3]

4.3.5 Mycroft

Mycroft is the name of a suite of software and hardware tools that use natural language processing and machine learning to provide an open source voice assistant.[30] Unlike the other virtual assistant we have seen previously, Mycroft is fully open source and free. It has been undergoing heavy development since late 2017, but now it claims to be usable effectively by developers and enthusiasts, making it the world's first fully open source AI voice assistant. But unfortunately, it is not yet usable by the general user, as it still requires technical skills.

Mycroft is available for Linux-based operating systems and Android, but it is not ready yet for macOS and Windows, making it harder to spread as quickly as other virtual assistants have done. However, to make up for this, they are selling their own smart speaker, called Mark, currently in its second version.

Mycroft is modular, which makes the system easily customizable, and uses a skill system similar to Amazon Alexa. It comes with a number of default skills, such as setting an alarm, providing the weather, or telling the time. The other skills are also installable via voice commands, based on a list of community-contributed skills. The number of additional skills is not as high as in Alexa, but it is constantly growing, and Mycroft encourages its users to contribute to the project.

The table 4.2 shows a comparison between the most important aspects of the previous voice assistants. In this table, note that *Smart Home* means

having Home Automation capabilities, and always on means that the user is able to trigger the assistant via the voice. For example, Siri is able to react to the sentence *Hey Siri!*, even if the device is with the screen off.

Name	**Developer**	**Free**	**Open source**	**Home Automation**	**Mobile app**	**Extra devices**	**Always on**
Bixby	Samsung	No	No	Yes - SmartThings	Yes Samsung phones	No	Yes
Assistant	Google	No	No	Yes	Yes Android	Google Home	Yes
Siri	Apple	No	No	Yes - HomeKit	Yes iOS	HomePod	Yes
Alexa	Amazon	No	No	Yes	Yes iOS, Android	Echo Echo Dot Echo Dot Kids Echo Plus	Yes
Mycroft	Mycroft and the Mycroft Community	Yes	Yes	Yes	Yes Android	Mark II Mark I	Yes

Table 4.2: Comparison between different voice assistants

Chapter 5

OpenHAB

After analyzing the main products and services in the field of home automation and voice assistance, I must choose the solution that can best fit this project. One of its requirements is to use as many open source and free technologies as possible, and that the final product is easily usable by the final user and sufficiently flexible to adapt it to our needs.

OpenHAB is fully open source and completely free, and has reached a level of maturity where it is highly stable and intuitive. In its most recent version, it provides an user interface that automatize many tasks that a standard user might not know how to do. And, of course, it can integrate many devices from different vendors, as In have mentioned in the previous chapters, which is also one of the most important matters for reaching our objective.

In this chapter, I will explore in depth this home automation platform and all its possibilities, in order to have a better general idea about it when building the final system.

5.1 Introduction

As I mentioned in previous chapters, openHAB (open Home Automation Bus) is a completely free, technology agnostic and open source platform for home automation.

OpenHAB software is capable of integrating different domotic systems, devices and technologies into a single solution. It also provides uniform user interfaces, and a common approach to automation rules across the entire system, regardless of the number of manufacturers and sub-systems involved.[35]

The platform runs on many popular platforms including Linux, Windows

and macOS. It is also popular to install it in systems like the Raspberry Pi, and openHAB even provides a special distribution for this computer, called *openHABian*, a simplified way of getting up and running openHAB, but offering the complete experience.

OpenHAB defines also a community of users, contributors and maintainers, working together on the improvement of the system. Everything related to the community is in the openHAB community forum. The community is very active and helpful, and thanks to them I have always found a way to solve my issues.

5.2 History of openHAB

The history of openHAB begins in 2010, when Kai Kreuzer, a smart home enthusiast from Germany, developed in Java and using the OSGi technology (Open Services Gateway Initiative) as the basis, which is a set of specifications that define a dynamic component system for Java. The use of this technology makes it easier to update the services independently and their implementation. It favor the expandability of the system.

In 2013, openHAB becomes an official Eclipse project under the name of Eclipse SmartHome, but they decide to keep both projects active and to develop them at the same time. In Eclipse SmartHome would maintain the architecture and the functionalities from the previous openHAB, and in openHAB they would study how to integrate the different devices and technologies that it supports via add-ons.

The newest version, OpenHAB 2, has been the biggest change that OpenHAB has suffered since its initial launch. It includes more add-ons and some changes that simplify much more the process for developers, as well as implementing Apache Karaf underneath, which greatly extends its possibilities. In addition, the UIs have been improved, improving greatly the user experience. OpenHAB 2 is much easier to install, and it automates many repetitive processes that might result hard for some users.

5.3 Structure

OpenHAB works thanks to add-ons, which can extend its capabilities to fit each user's needs, from User Interfaces, to the ability to interact with a large and growing number of physical *Things*. Add-ons may come from the OpenHAB 2 distribution, the Eclipse SmartHome project Extensions, or from the OpenHAB 1 distribution.

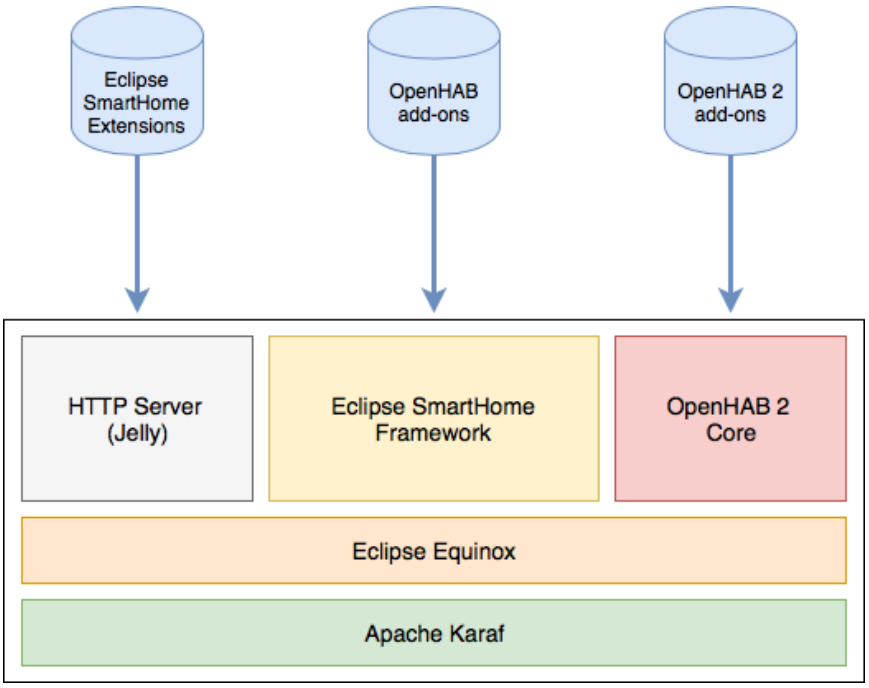

Figure 5.1: OpenHAB architecture

The figure 5.1 shows the overall architecture of openHAB 2. In the lowest layer, we can find Apache Karaf. Apache Karaf is basically a modular and open source OSGi runtime environment that can host any kind of applications.[4]

Next, there is Eclipse Equinox, which is also an implementation of the OSGi core framework specification. But the goal of the Equinox project is to be a first class OSGi community and foster the vision of Eclipse as a landscape of bundles too. Equinox is responsible for developing and delivering the OSGi framework implementation used for all of Eclipse.[56]

The next and last level is divided in three parts. The first one is the Jetty HTTP server, also part of Eclipse, which provides a Web server and javax.servlet container, plus support for HTTP/2, WebSocket, OSGi, JMX, JNDI and JAAS, among others.[25] Secondly, we can find the Eclipse SmartHome Framework , the framework to build end user solutions on top like openHAB, that I mentioned before.[14] The last part is the core of openHAB 2, which provides the full solution.

As the diagram in the figure 5.1 indicates, we can add to this system extensions from Eclipse SmartHome and add-ons from the first and second version of openHAB.

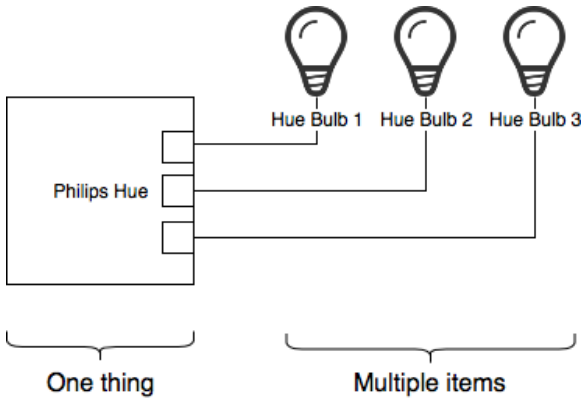

Figure 5.2: A simplification of the concepts of Thing and Item

5.4 Concepts

As Eclipse SmartHome is the logic part of OpenHAB 2, all the elements I am listing are part of it. Eclipse SmartHome strictly differentiates between the physical view and the functional view of the system. The physical view is more familiar to us, and focuses on the devices on the system, the connections between them (e.g. wires, Netatmo devices, WiFi hardware) and other physical aspects of the system. The functional view focuses on how information about the devices, connections, and so on, is represented in user interfaces, focusing on how rules effect representations of physical devices in software. The functional view focuses on how an action in a user interface affects the software associated with the physical device it represents.[35]

That said, I will explore the different elements that Eclipse SmartHome considers in this section. The greatest difference that we can find related to devices, is between *Things* and *Items*. Generally speaking, Things represent physical systems that can be added to openHAB and Items represent functionalities that can be used by the applications. The figure 5.2 shows a graphical explanation of this concept, though I will explain in depth these concepts below.

5.4.1 Things

Things are the entities that can physically be added to a system and which can potentially provide many functionalities in one. They do not need to be devices, but they can also represent a web service, or any other manageable source of information and functionality. Things are important in the setup and configuration process, when the user has to add his devices to the system, but they are not for the operation, when everything is up and

running.

Things can have configuration properties, which can be optional or mandatory. Such properties can be basic information like an IP address, an access token for a web service or a device specific configuration that alters its behavior.

Channels

Channels are part of the Things, and they represent the different functions they provide. Where the Thing is the physical entity or source of information, the Channel is a concrete function from this Thing. For example, some Philips Hue light bulbs have a color temperature Channel and a color Channel, both providing functionality of the one light bulb Thing to the system. For sources of information the Thing might be the local weather with information from a web service with different Channels like temperature, pressure and humidity.

Channels are linked to Items, where such links are the glue between the virtual and the physical layer. Once such a link is established, a Thing reacts on events sent for an item that is linked to one of its Channels. Likewise, it actively sends out events for Items linked to its Channels

Bridges

Bridges are special types of Things. They are *Things* that need to be added to the system in order to gain access to other Things. For example, an IP gateway for some non-IP based home automation system or a web service configuration with authentication information which every Thing from this web service might need.

Some Bindings come with a Bridge, like the *PHC Binding*, which allows to integrate modules of PHC in openHAB.[38]

Thing Status

Every Thing has a status, which helps to identify possible problems with the device or service and gives useful information to the user in any moment. The statuses are limited to seven types, as the table 5.1 shows.

The statuses UNINITIALIZED, INITIALIZING and REMOVING are set by the framework, where as the statuses UNKNOWN, ONLINE and OFFLINE are assigned from a binding. Additionally, the REMOVED state is set by the binding to indicate that the removal process has been completed, that it, the Thing must have been in REMOVING state before.

Status	Description
UNINITIALIZED	This is the initial status of a Thing, when it is added or the framework is being started. This status is also assigned, if the initializing process failed or the binding is not available. Commands, which are sent to Channels will not be processed.
INITIALIZING	This state is assigned while the binding initializes the Thing. It depends on the binding how long the initializing process takes. Commands, which are sent to Channels will not be processed.
UNKNOWN	The handler is fully initialized but due to the nature of the represented device/service it cannot really tell yet whether the Thing is ONLINE or OFFLINE. Therefore the Thing potentially might be working correctly already and may or may not process commands. But the framework is allowed to send commands, because some radio-based devices may go ONLINE if a command is sent to them. The handler should take care to switch the Thing to ONLINE or OFFLINE as soon as possible.
ONLINE	The device/service represented by a Thing is assumed to be working correctly and can process commands.
OFFLINE	The device/service represented by a Thing is assumed to be not working correctly and may not process commands. But the framework is allowed to send commands, because some radio-based devices may go back to ONLINE, if a command is sent to them.
REMOVING	The device/service represented by a Thing should be removed, but the binding did not confirm the deletion yet. Some bindings need to communicate with the device to unpair it from the system. Thing is probably not working and commands can not be processed.
REMOVED	This status indicates that the device/service represented by a Thing was removed from the external system after the REMOVING was initiated by the framework. Usually this status is an intermediate status because the Thing gets removed from the database after this status was assigned.

Table 5.1: Statuses of Things in openHAB 2

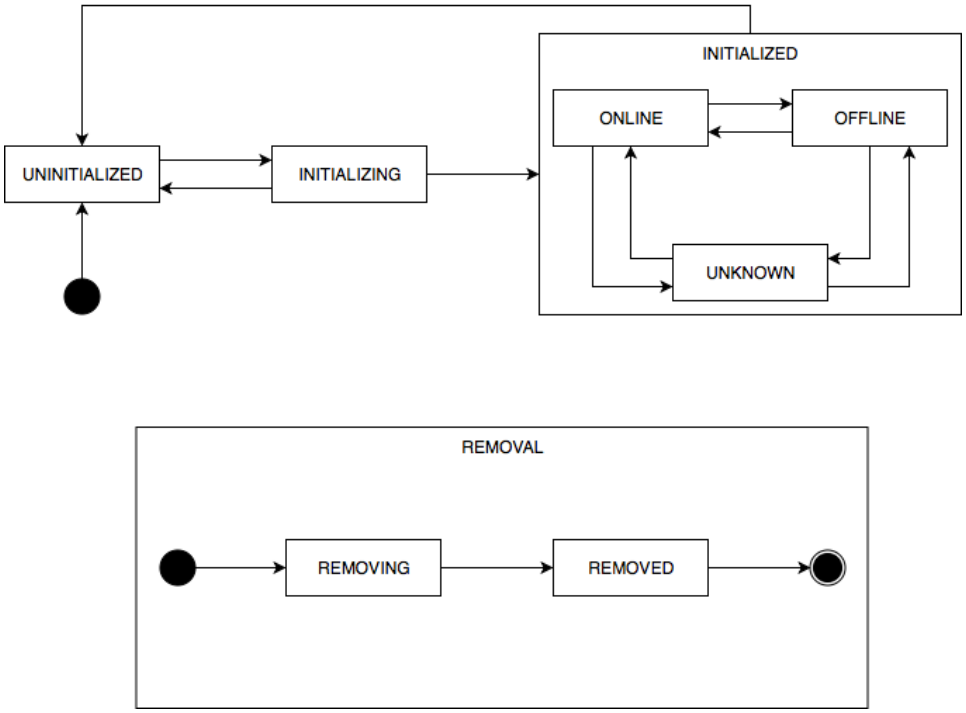

Figure 5.3: Thing status transitions

Status Transitions

The figure 5.3 shows the possible status transitions in openHAB.

The initial state of a Thing is UNINITIALIZED. From UNINITIALIZED the Thing goes into INITIALIZING. If the initialization fails, the Thing goes back to UNINITIALIZED. If the initialization succeeds, the binding sets the status of the Thing to UNKNOWN, ONLINE or OFFLINE, which all mean that the Thing handler is fully initialized. From one of this states the Thing can go back into UNINITIALIZED, REMOVING or REMOVED. The statuses REMOVING and REMOVED can also be reached from any of the other states.

5.4.2 Items

Eclipse SmartHome has a strict separation between the physical world (Things) and the application, which is built around the concept of *Items* (also known as the *virtual layer*).

As mentioned at the beginning of this section, Items represent functionalities that can be used by the applications, mainly user interfaces or

Type	Description	Command Types
Color	Color information (RGB)	OnOff, IncreaseDecrease, Percent, HSB
Contact	Item storing status of e.g. door/window contacts	OpenClose
DateTime	Stores date and time	
Dimmer	Item carrying a percentage value for dimmers	OnOff, IncreaseDecrease, Percent
Group	Item to nest other Items / collect them in Groups	
Image	Holds the binary data of an image	
Location	Stores GPS coordinates	Point
Number	Stores values in number format, takes an optional dimension suffix	Decimal
Number <dimension>	Like Number, but with additional dimension information for unit support	Quantity
Player	Allows to control players (e.g. audio players)	PlayPause, NextPrevious, RewindFastforward
Rollershutter	Typically used for blinds	UpDown, StopMove, Percent
String	Stores texts	String
Switch	Typically used for lights	OnOff

Table 5.2: Types of Items in openHAB 2

automation logic. Items also have a state and are used through events.

Each openHAB Item must be between the list of types that the table 5.2 specifies.

Group Items

Group Items are a special kind of items that collect other Items into Groups. Group Items can themselves be members of other Group Items. Depending on the user interface, it might display Group Items as single entries and provide navigation to its members.

With Group Items, it is also possible to derive their state from their member items. To derive a state the Group Item must be constructed using a base Item and a Group function. Between the available Group functions we can find common operators such as EQUALITY, AND, OR, NAND, NOR, SUM, AVG, MIN and MAX, among others.

Links

Links are the glue between Things and Items. They are associations between exactly one Thing Channel and one Item. If a Channel is linked to an Item,

it is enabled, which means that the functionality that the Item represents is handled through the given Channel. Channels can be linked to multiple Items and Items can be linked to multiple Channels.

5.4.3 Thing Discovery

Thing Discovery is the process that the system makes in order to show the devices connected in your network. Many technologies, devices and systems can be discovered automatically or browsed through an API.

In Eclipse SmartHome bindings implement *Discovery Services* for Things, which provide *Discovery Results*. All Discovery Results are regarded as suggestions to the user and are put into the *Inbox*.

Inbox

The Inbox holds a list of all discovered Things from all active discovery services. A discovery result represents a discovered Thing of a specific Thing type, that could be instantiated as a Thing. The result usually contains properties that identify the discovered Things further like IP address or a serial number. Each discovery result also has a timestamp when it was added to or updated in the Inbox and it may also contain a time to live, indicating the time after which it is be automatically removed from the Inbox.

Discovery results can either be ignored or approved, where in the latter case a Thing is created for them and they become available in the application. If an entry is ignored, it will be hidden in the Inbox without creating a Thing for it.

Eclipse SmartHome offers a service that is capable of automatically ignore discovery results on the Inbox, whenever a Thing is created manually, that represents the same Thing, as the respective discovery result would create. This Thing would either have the same Thing UID or the value of its representation property is equal to the representation property's value in the discovery result. The service is enabled by default.

5.4.4 Audio and Video

Audio and voice features are an important aspect of any smart home solution as it is a very natural way to interact with the user.

Eclipse SmartHome comes with a very modular architecture that makes it possible in plenty of situations. At its core, there is the notion of an *audio stream*. Audio streams are provided by *audio sources* and consumed by *audio sinks*.

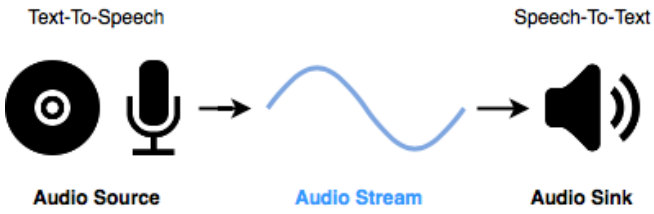

Figure 5.4: Eclipse SmartHome audio stream scheme

- **Audio Streams** are essentially a byte stream with a given audio format.
- **Audio Formats** define the container (e.g. WAV), encoding, bit rate, sample frequency and depth and the bit order (little endian or big endian).
- **Audio Sources** are services capable of producing audio streams, which are able to support different formats and provide a stream in a requested format upon request. Typical audio sources are microphones, and a continuous stream is expected from them.
- **Audio Sinks** are services that accept audio streams of certain formats. Typically, these are expected to play the audio stream, for example, a speaker.
- **Text-to-Speech (TTS)** services are similar to audio sources with respect to the ability to create audio streams. The different is that they take a string as an input and will synthesize it to a spoken text using a given voice.
- **Speech-to-Text (STT)** services are similar to audio sinks, but they do not simply play back the stream, but convert it to a plain string.

TTS and STT can provide information about the voices that they support, formats and locales. In TTS, each voice supports exactly one locale.

However, the STT service itself does not seem to be very useful. In order to process the generated string, there is the concept of a *human language interpreter*.

Human Language Interpreter

A Human Language Interpreter takes a string as an input. It then derives actions from it, like sending commands to devices, or replies with a string, which opens the possibility to realize conversations. The figure 5.5 shows a simple schema of how it works.

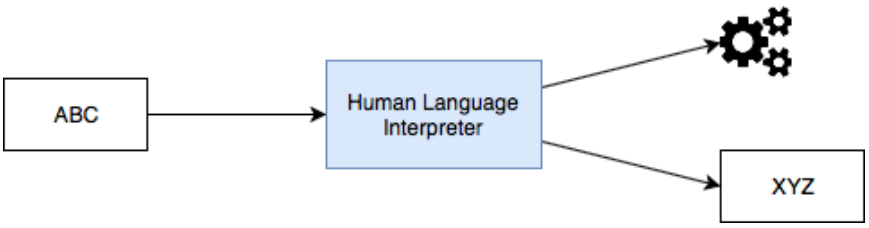

Figure 5.5: A Human Language Interpreter transforms strings into other strings or commands

In fact, an interpreter is not directly related to audio streams, but operates only with strings, so it is suitable either for voice assistants or chatbots for console, Twitter or other messaging services.

Applications can dynamically choose which services to use, so that different sinks can be used for different use cases. Defaults can be set as configuration for all those services in case an application does not ask for any specific service.

5.5 A Developer Perspective on openHAB

OpenHAB 2 is a great open source, technology agnostic home automation platform that is able to manage lots of smart services and devices.

However, from a developer point of view, OpenHAB is not a complete platform itself, but rather an aggregation of features from different repositories, and mainly from Eclipse SmartHome:

- **Eclipse SmartHome Framework**: the major parts of the core functionality are held by this repository. It contains bindings, services and items, amongst others, as openHAB does.
- **OpenHAB 2 Core and openHAB add-ons**: add-ons of openHAB that use the Eclipse SmartHome API.
- **Eclipse SmartHome Extensions**: openHAB is compatible with all extensions that are available for the Eclipse SmartHome Framework and maintained within their repositories.

5.5.1 Development environment set up

Installing the development environment is a different process than installing OpenHAB itself, as a developer would require a local copy of the source code of all elements, including the bindings, in an IDE.

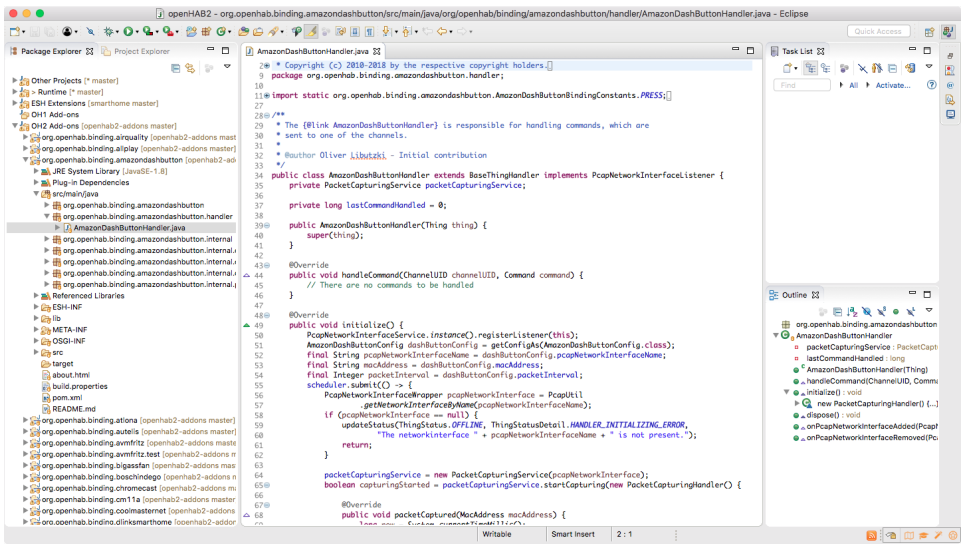

Figure 5.6: Eclipse SmartHome IDE with the OpenHAB repositories

The process for installing OpenHAB 2 as a developer requires first to install Eclipse IDE with the OpenHAB-related repositories, which are the ones listed previously. It is usually required to have Maven 3 installed, and it is mandatory to have Oracle JDK 8 beforehand, because OpenHAB is fully coded in Java. [37]

The Eclipse installer downloads a copy of the repositories that the developer selects and installs and integrates them with Eclipse IDE automatically. Then, user can compile, run and debug the project from the IDE.

5.5.2 Platform structure

Installing the platform as mentioned above provides us with a clear knowledge of the platform's structure and makes it easy to perform any modification or addition to it. The OpenHAB 2 code is highly modular and presents a very well-defined organization, as can be seen in the figure 5.7. Below there is a detailed explanation about the structure.

We can divide OpenHAB 2 in five parts, which are composed by one or more subsections that host differentiated code parts, each one performing services, connecting to devices or managing the internal system:

- **Runtime**: these are the basic libraries that OpenHAB need in order to execute properly, which come mostly from Eclipse SmartHome. It is a big collection of different services:
  - **Automation files**: they are in charge of the automated services

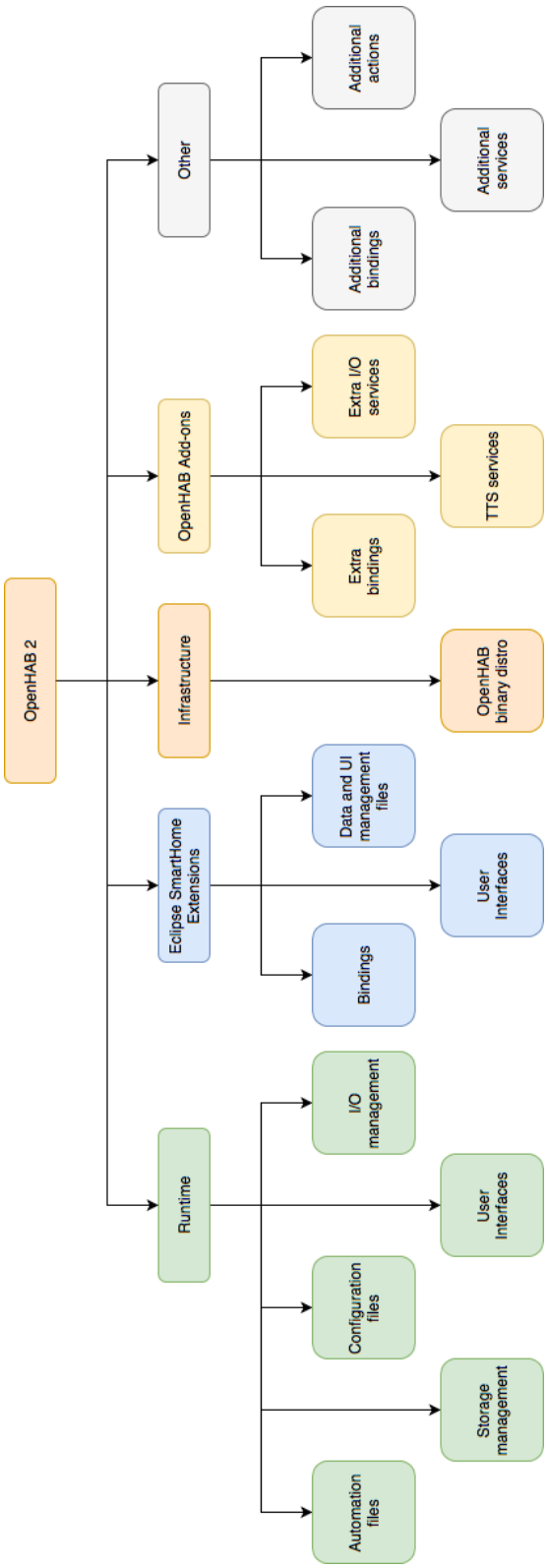

Figure 5.7: OpenHAB 2 structure

in the platform, so they manage triggers and events.

  - **Configuration files**: hosts the configuration files of the platform, such as the user folders or the listeners for the item discovery process.
  - **I/O management**: these libraries manage the inputs and outputs of the system, such as the MQTT communication or the REST APIs.
  - **Storage management**: a pair of libraries that are in charge of storing JSON files and managing MapDB databases.
  - **User interfaces**: the user interfaces OpenHAB provides by default are the BasicUI, the ClassicUI, the HomeBuilder and the PaperUI (being this the most common in OpenHAB 2, as it requires zero manual configuration). These UIs are all part of this repository and they come from OpenHAB, not from Eclipse SmartHome. However, Eclipse also provides packages for internal aspects of the UI and for icons.

- **Eclipse SmartHome Extensions**: functionality of Eclipse SmartHome can be extended through different additions, such as bindings or other UIs. In this case, we can find:

  - **Bindings**: they are the most important part of our system. Bindings integrate external systems, like services, protocols or single devices to the platform. Therefore, the main purpose of a binding is to translate events from the Eclipse SmartHome event bus to the external system and vice versa. This repository contains bindings for communicating via Bluetooth, or to Philips Hue or DMX devices, amongst others. Many other bindings that OpenHAB support are located in the OpenHAB Add-ons repository.
  - **User interfaces**: includes a bunch of UIs from Eclipse SmartHome, from which the OpenHAB UIs were created.
  - **Data and UI management files**: this category contains additional configuration files and for managing UI's elements.

- **Infrastructure**: includes the binary files of OpenHAB. The end-user version of OpenHAB consists in only this repository.

- **OpenHAB add-ons**: these are libraries that OpenHAB introduced to Eclipse SmartHome. They extend its functionality to make it a fully usable Home Automation environment.

  - **Extra bindings**: OpenHAB created a huge number of new bindings for Eclipse SmartHome, from the binding for Amazon Dash

Button to the one for Samsung TVs. They cover an enormous range of smart devices, and the list is constantly growing.

  - **Extra I/O services**: some bindings, like the Apple HomeKit binding, require I/O services that Eclipse SmartHome does not originally include. In addition, new services like the OpenHAB cloud are also contained here.
  - **TTS services**: OpenHAB 2 has added Text-To-Speech and Speech-To-Text functionality to Eclipse SmartHome, which is also part of the OpenHAB add-ons repository.

- **Other projects**: this repository holds libraries that are not part of any of the others, and it is a mix of additional bindings (EnOcean, FritzBox...), additional actions and more services. Although they are located in this repository, they are officially supported by OpenHAB.

5.5.3 OSGi

OpenHab is based on OSGi. The OSGi technology is a set of specifications that define a dynamic component system for Java. These specifications enable a development model where applications are dynamically composed of many different and reusable components.

The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components.[35] The main features of OSGi are modularity, runtime dynamics and service orientation.

OSGi Containers

Different containers might implement different parts of the OSGi specifications and might provide slightly different API. The OpenHAB project uses Equinox, which is the reference implementation of the OSGi R4.x Core Specification and one of the mostly used as well.

Other popular open source OSGi containers are Apache Felix and Concierge. The container ProSyst OSGi Framework is widely used as well, but it is not free.

Definitions

- **Bundle**: the OSGi components made by the developers. A bundle is comprised of Java classes and other resources, which together can provide functions to end users.

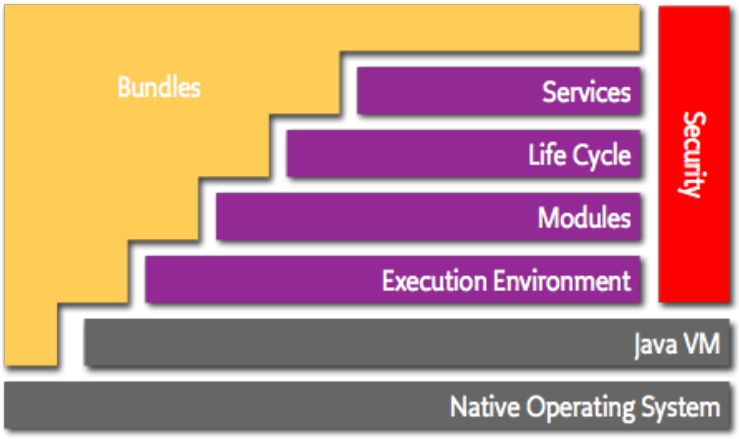

Figure 5.8: OSGi layer structure

- **Service**: any object that is registered in the OSGi Service Registry and can be looked up using its interface name(s).
- **Manifest**: descriptive information about the bundle, contained in its JAR file.
- **Service** Registry: enables a bundle to publish objects to a shared registry, advertised via a given set of Java interfaces.

Layer Structure

As the figure 5.8 shows, the OSGi framework consists of layers build on top of each other:

- **Module layer**: responsible for managing dependencies between bundles and for class loading.
- **Life Cycle Layer**: controls the life cycle of the bundles.
- **Service Layer**: defines a dynamic model of communication between different modules.
- **Actual Services**: this is the application layer, using all other layers.
- **Security Layer**: optional layer that manages permissions for different modules.

Bundles

Bundles, also known as modules, are the smallest unit of modularization. Technically, they are a JAR file with additional meta information, which are stored in a file called *manifest file*. The manifest file is part of the standard Java specification, but OSGi adds additional metadata to it in form of specific headers. The *Bundle-SymbolicName* and the *Bundle-Version* headers uniquely identify a bundle. In OSGi is allowed to have bundles with same name, but different version running at the same time.

The manifest contains information like the bundle dependencies. A bundle can depend on another bundle or on a package. Preferred way to define dependencies in a bundle is with *Import-Package* and *Export-Package* headers and not with *Require-Bundle* header. This gives you an access only to the packages that you need and allows you to exchange the packages at a later point in time

The OSGi runtime uses the information about the dependencies to *wire* the bundles and hides everything in this JAR unless it is explicitly exported. The dependencies to the Java standard libraries are managed by the *Bundle-RequiredExecutionEnvironment* header, so it is not needed to import the Java core packages

Bundles are used often to register and consume services.

Lifecycle

OSGi is a dynamic platform. That means that bundles may be installed, uninstalled, started, stopped or updated at runtime, as the table 5.3 indicates. The OSGi specification defines a mechanism how to manage the dependencies between the bundles and the functionality that they provide. This is achieved with the help of the lifecycle concept.

The framework introduces a different states, transitions between these states and rules how this states are affecting the packages exported by the bundle and the services, that it provides. The table 5.3 shows the possible states of an OSGi bundle with a short explanation

The possible status transitions are shown in the state diagram in the figure 5.9.

The Service Model

The service model is another main concept that allows the bundles to communicate between each other.

In OSGi, a bundle can register a service in a central service registry under

Status	Description
INSTALLED	The bundle has been installed into the OSGi container, but some of it's dependencies are still not resolved. The bundle requires packages that have not been exported by any other bundle.
RESOLVED	The bundle is installed and the all the dependencies at a class level are resolved and wired. The bundle can export the packages, that it provides.
STARTING	A temporary state that the bundle goes through while the bundle is starting, after all dependencies have been resolved. The bundle is permitted to register services.
ACTIVE	The bundle is running
STOPPING	A temporary state that the bundle goes through while the bundle is stopping
UNINSTALLED	The bundle has been removed from the OSGi container

Table 5.3: Bundle states description

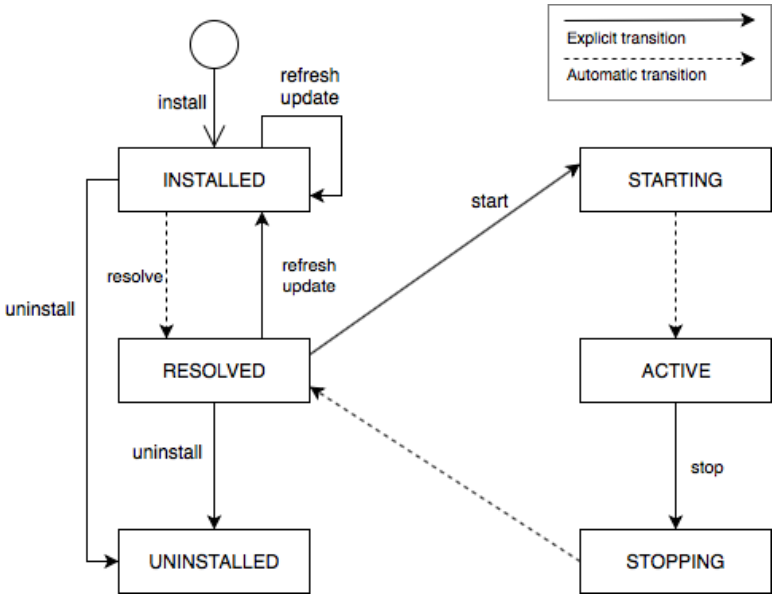

Figure 5.9: Bundle state diagram

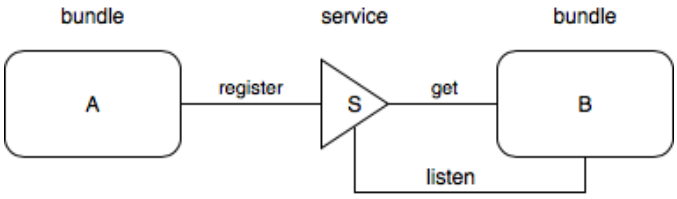

Figure 5.10: OSGi services [39]

one or more service interface. Published services also have service properties associated with them in the registry. It is an important feature of OSGi, because it provides a central place to register and get services. A bundle is permitted to register service objects at any time during the STARTING, ACTIVE or STOPPING states. Other bundles can go to the registry and list all objects, that are registered under a specific interface or class.

A bundle can therefore register a service, it can get a service and it can track for appearing and disappearing of service. Any number of bundles can register the same service type and any number of bundles can get the same service. The figure 5.10 represents a basic diagram of the service usage and tracking.

Declarative Services

In order to simplify the usage of services the OSGi Alliance has developed a model of managing services dynamically called Declarative Services. It is based on three main concepts:

- **Declarative Services Container (DS)**: a module that is managing the lifecycle of a service component dynamically. It activates and deactivated different components, basing its decisions on the information contained in the component description.
- **Service Component**: an object whose lifecycle is managed, usually by a component framework such as DS.
- **Component Description**: the declaration of a component, contained within an XML document in a bundle.

DS Container In order to use the Declarative Services, a bundle has to be started with an implementation of the DS container. In Equinox this bundle is called *org.eclipse.equinox.ds*.

When a bundle that contains a component is added to the framework, DS reads its component description and if the conditions described in this

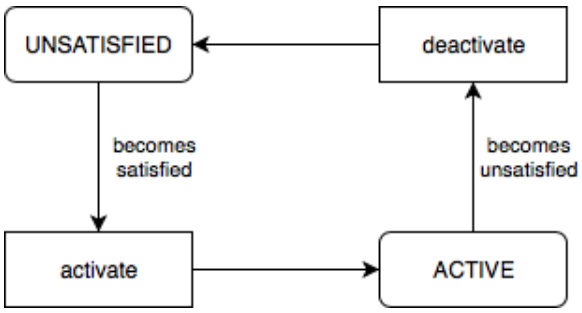

Figure 5.11: Immediate component lifecycle

file are fulfilled, the DS activates the component.

Components A component is a normal Java class, that can reference services and provide services. What makes it specific is that it is declared in a XML file and is managed completely by the DS, so the DS instantiates the component, calls methods on it and manages its lifecycle.

A component in a bundle requires an XML description of the component, a *Service-Component* manifest header, which locates the XML description, and an implementation class. There are three types of components:

- **immediate**: with *immediate* attribute set to true
- **delayed**: with *immediate* attribute set to false
- **factory**

A component goes through several states in his lifecycle:

- **UNSATISFIED**: initial state of the Service Component, after the bundle is started.
- **REGISTERED**: temporary state, only *delayed* components go through this state.
- **ACTIVE**: the component is active and component instance is created.

The component lifecycle depends on the lifecycle of the bundle, that includes the component. Component must be enabled before it can be used. A component is enabled, when the component's bundle is started and disabled, when the bundle is stopped.

After the Component is enabled, it is moved to the UNSATISFIED state. The next step is to satisfy the component configuration.

The component configuration is satisfied when:

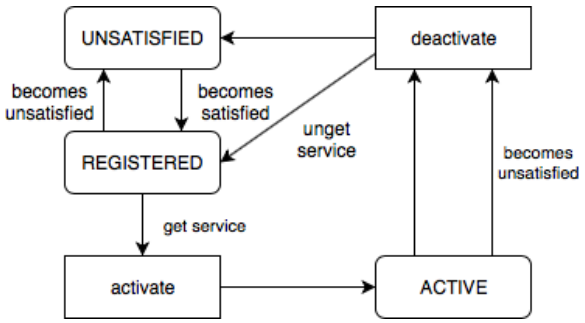

Figure 5.12: Delayed component lifecycle

- Component is enabled.
- All the component references are satisfied. A reference is satisfied when the reference specifies optional cardinality or there is at least one target service for the reference. If the component has lazy initialization (the component is delayed), it is moved to the REGISTERED state and it is waiting to be activated, when the service is requested (see figure 5.12). Otherwise (the component is immediate) as soon as its dependencies are satisfied, the component is activated (see figure 5.11).

OpenHAB is much more than what I have explained in this chapter. But, as I will develop the project partially over openHAB, I will explore more concepts from a developer perspective in the following chapter, like the installation of the system and the configuration of its different parts.

Chapter 6

Project Development

In each of the previous chapters I have explored different aspects of the fields of home automation and voice assistance from a theoretical point of view, from their definition to smaller details, including additional explanations about a specific home automation system, openHAB.

At this time, I think I have provided enough background to begin my case study: the building of a home automation controller. As I mentioned in the openHAB chapter, I will base my project on this system and build on it, as it accomplishes very well the main requirements that I determined for this project.

In this chapter, I will detail the process that I have followed in order to develop this project, from its specification to the final result.

6.1 Product Specification

The first task to do is to define the product. What should a home automation system do? What do users expect it to do? How? All the answers to these questions can be clarified by following some processes that, although they do not completely answer them (we can see many failed projects from time to time), they provide a very clear and detailed specification from the beginning.

6.1.1 Personas

Creating personas is a common process applied to the product design and development process in order to help in making a user-centered design, and it is applicable in this project in order to have a better idea of what would users expect from the final product.

A persona is a representation of a user, typically based off user research and incorporating user goals, needs, and interests.

In this project, I am going to use proto-personas, which are based on secondary research and the guess of who they should be designed for, as currently we do not have means and time for making true research-based personas (and it is not the main objective in this project).

After thinking about the main uses of this system and the people that would be interested on it, I have extracted these three personas, representing its main uses, although not the only ones. I have built them with the online platform Xtensio, and the figures 6.1, 6.2 and 6.3 represent them. I tried to extract a varied range of backgrounds, current situations, desires and worries.

Oswald Douglas (figure 6.1) is a freelance technology blogger from Dallas, USA, that is very interested in the areas of home automation and Internet of Things. He is looking to automate his own home and write about his experience in his blog. He already has experience with technology, and this next step will not be too difficult for him. His interests are clear: to try cutting-edge technology in his own home and make the most of home automation.

Anna Lahtinen (figure 6.2) is a 16 year-old high school student from Lappeenranta, Finland. She is up to date on technology but she is not passionate about it. However, she heard about home automation and thinks that she could enjoy a better media experience with it. In addition, she thinks that adding smart color light bulbs to her bedroom would make it look more beautiful. However, she feels that there is a lack of general information about devices and the set up and configuration of a home automation system. She thinks that the price of it is too high as well.

Rosario Vera (figure 6.3) is an administrative from Vitoria-Gasteiz, Spain. She is 37 years old, is married and has two young children. She is not very familiar with technology, but she has heard about home automation in the news and thinks that it could fit her needs. Rosario and her husband work outside home, and sometimes their children need to be alone at home. Home automation would provide more security to the home and would allow them to have more spare time. Voice assistance would be helpful for their children when they are alone, as it is a very easy and natural way to interact with technology. However, she is concerned about their privacy regarding these systems and she thinks that companies should give more accessible explanations about it. In addition, she finds these systems difficult to use.

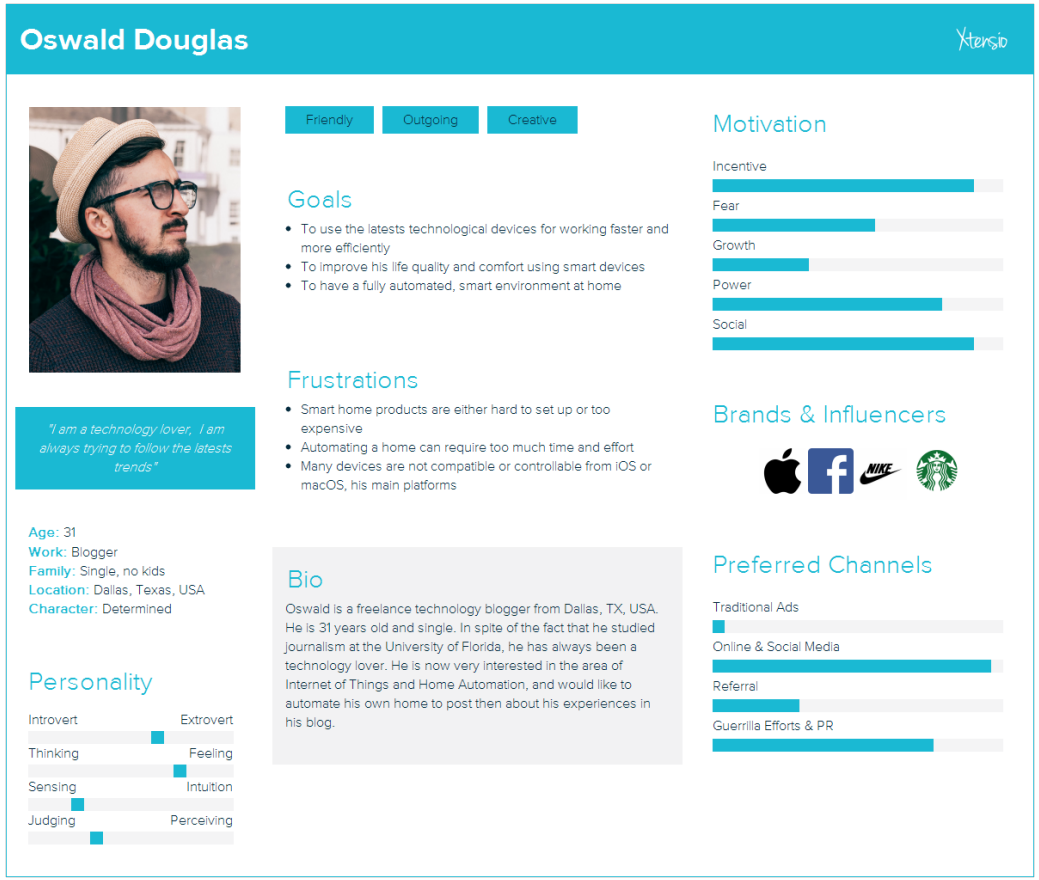

Figure 6.1: Persona: Oswald Douglas

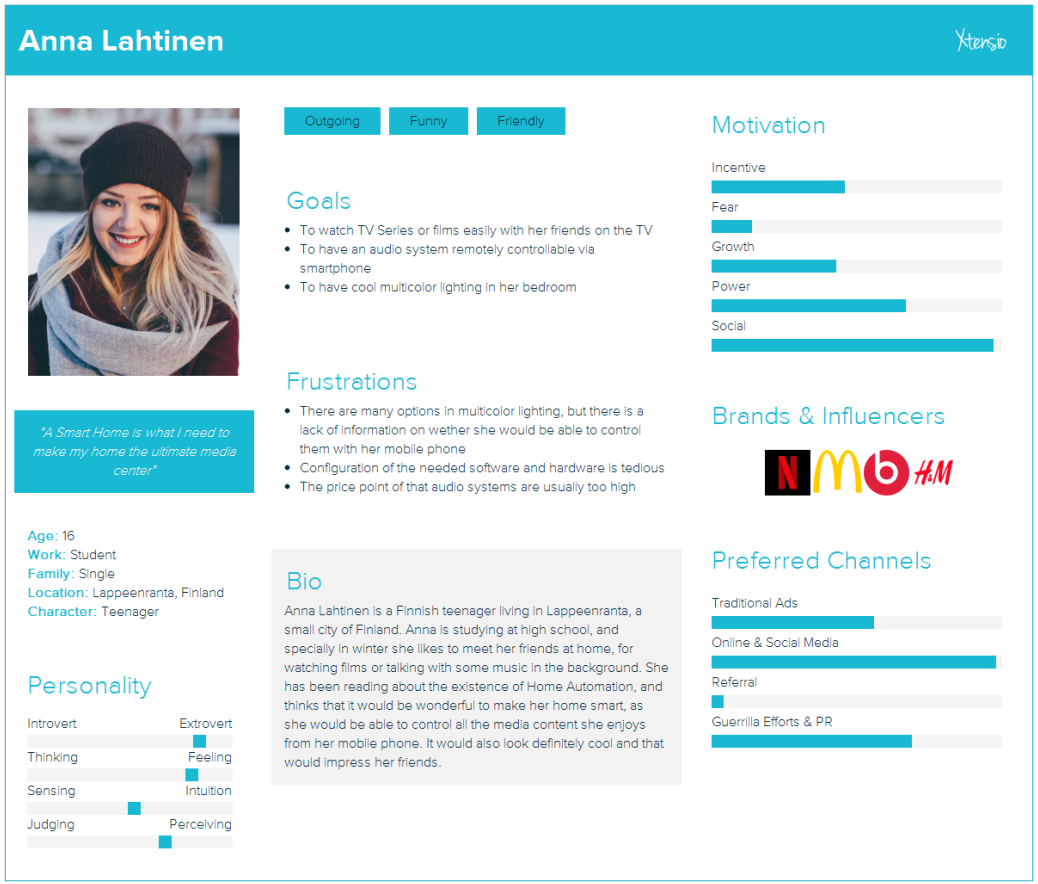

Figure 6.2: Persona: Anna Lahtinen

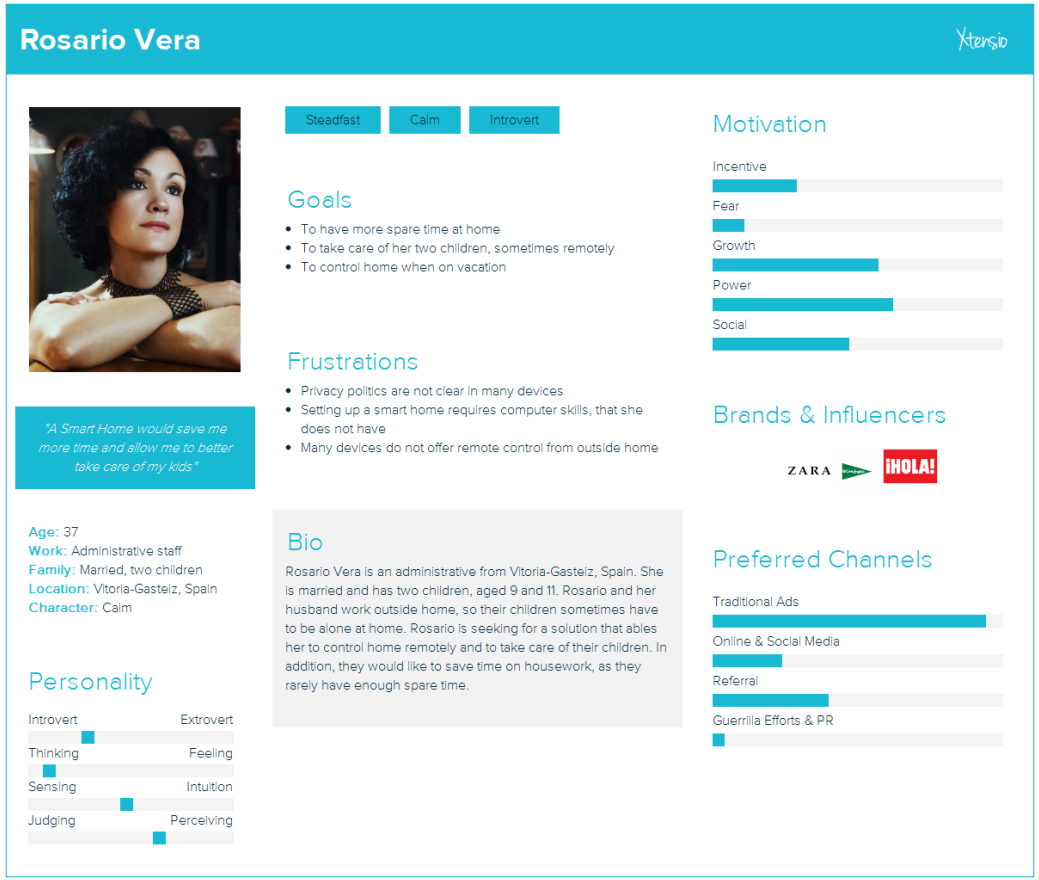

Figure 6.3: Persona: Rosario Vera

6.1.2 Software Requirements Specification

With the Software Requirements Specification (SRS), I try to describe the project to develop from a functional point of view, that is, to determine the capabilities that the software system will have.

The domotic controller will be able to control all modern devices in our home, regardless of their maker and the technology they use. It will provide an easy to use user interface and the ability to easily install, modify or remove the devices. It will also include natural human-computer interaction through the voice. A more detailed specification of the requirements can be found in the subsections below.

Functional Requirements

Functional requirements are a description of the facility or feature required. They deal with what the system should do or provide for users.[51]

- **FR1**: The system will be able to retrieve automatically the status of the different properties of the elements.
- **FR2**: The system will be able to retrieve automatically data from its different data providers.
- **FR3**: The system will not need any operation to launch openHAB after it is powered on.
- **FR4**: The system will be able to turn itself off safely.
- **FR5**: The system will automatically detect new smart devices connected to the local network.
- **FR6**: The system will be able to detect the possible operations with each connected smart device.
- **FR7**: The system will be able to operate with the connected devices according to the detected possible operations.
- **FR8**: The system will be able to tell the user in an understandable manner the current connection status for each device.
- **FR9**: The system will provide different user interfaces, so users can choose one between them, according to their needs.
- **FR10**: The user will be able to change the configuration of the system from a graphical user interface.

- **FR11**: The system will be manually configurable and modifiable using configuration files.
- **FR12**: The system will automatically include and configure new devices found in the network, after the user decides to add them.
- **FR13**: The user will be able to configure the name, display icon, IP and all possible aspects of the item directly from the user interface.
- **FR14**: The system will allow the user to make groups of items.
- **FR15**: The system will allow the user to add groups of items to other groups of items.
- **FR16**: The system will be modular and include a package system. Each package supports a set of devices.
- **FR17**: The package system will be accessible from the graphical user interface.
- **FR18**: The installation of packages will be done automatically after the user clicks on the install button.
- **FR19**: The removal of packages will be done automatically after the user clicks on the remove button.
- **FR20**: The system will maintain an updated and classified list of packages according to an external repository.
- **FR21**: The system will be manageable from external sources.
- **FR22**: The system will be accessible from external devices and from the Internet.
- **FR23**: The system will allow to have automation rules defined by the user.
- **FR24**: The voice assistant will allow to perform the main operations of each device in the system.
- **FR25**: The voice assistant will be able to answer in different languages.
- **FR26**: The voice assistant will be able to do power management tasks in the system, such as powering the system off or restarting the system.
- **FR27**: The voice assistant will be operable through mechanical methods or by voice.

- **FR28**: The users will be able to edit and adapt the functionalities of the voice assistant according to their needs.
- **FR29**: The voice assistant will provide spoken feedback if there has been an error performing the given command.
- **FR30**: The voice assistant will be able to print debug information in the command line if it is required.

Non-Functional Requirements

Non-functional requirements detail constraints, targets or control mechanisms for the new system. They describe how, how well or to what standard a function should be provided.[51]

- **NFR1**: The system, along with the voice assistant, will be installable in the Raspberry Pi.
- **NFR2**: The system and the voice assistant will provide a satisfactory response rate.
- **NFR3**: The user interfaces of the home automation system will be adaptable to the size and resolution of the user's screen.
- **NFR4**: The user interfaces will be made according to modern standards, like Material Design.
- **NFR5**: The home automation system will run in a local server, created in the machine which executes it.
- **NFR6**: In order to allow external connections, the home automation system will provide a REST API.
- **NFR7**: The system will provide secure access options.
- **NFR8**: The system must be available the 99% of the time.
- **NFR9**: The system must be scalable.
- **NFR10**: The system must be recoverable in less than 45 minutes.
- **NFR11**: The cost of the system must me minimal.
- **NFR12**: The system must be interoperable with multiple devices.
- **NFR13**: The voice assistant will recognize English.
- **NFR14**: The voice assistant will be fast in its response. Therefore, its procedure for deciding a response will be optimal.

- **NFR15**: The voice assistant will operate with the home automation system via REST, so it can be installed in a different computer than the one with the home assistant.

6.1.3 Use Cases

The next step after the specification of requirements is to specify the use cases of the system.

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).[57]

Use cases are represented in the figures 6.4, 6.5, 6.6 and 6.7.

6.1.4 Functional Subsystems

From the previously specified use cases, we can differentiate four functional subsystems. They are represented in the figure 6.8.

6.1.5 Implementation Possibilities

After exploring what users would expect from a system like this, it is worth thinking about how it should be implemented. Together with my director, I proposed different options to meet the previous requirements.

Web Platform

A well designed web platform is a great idea nowadays. If this platform is hosted in a local server, it means that it can be accessed from any device connected to the network. With the correct configuration, it could be even reachable from anywhere in the world. It is easy to implement and maintain, and users with basic web development knowledge would be able to easily modify it. In addition, the same web platform can be adaptable to any device and resolution, thanks to responsive designs. It would also work if we attach a tactile screen to the Raspberry Pi.

OpenHAB mounts a web platform in a local server by default, and provides several adaptable user interfaces, so this solution would not require much effort, and it offers many possibilities.

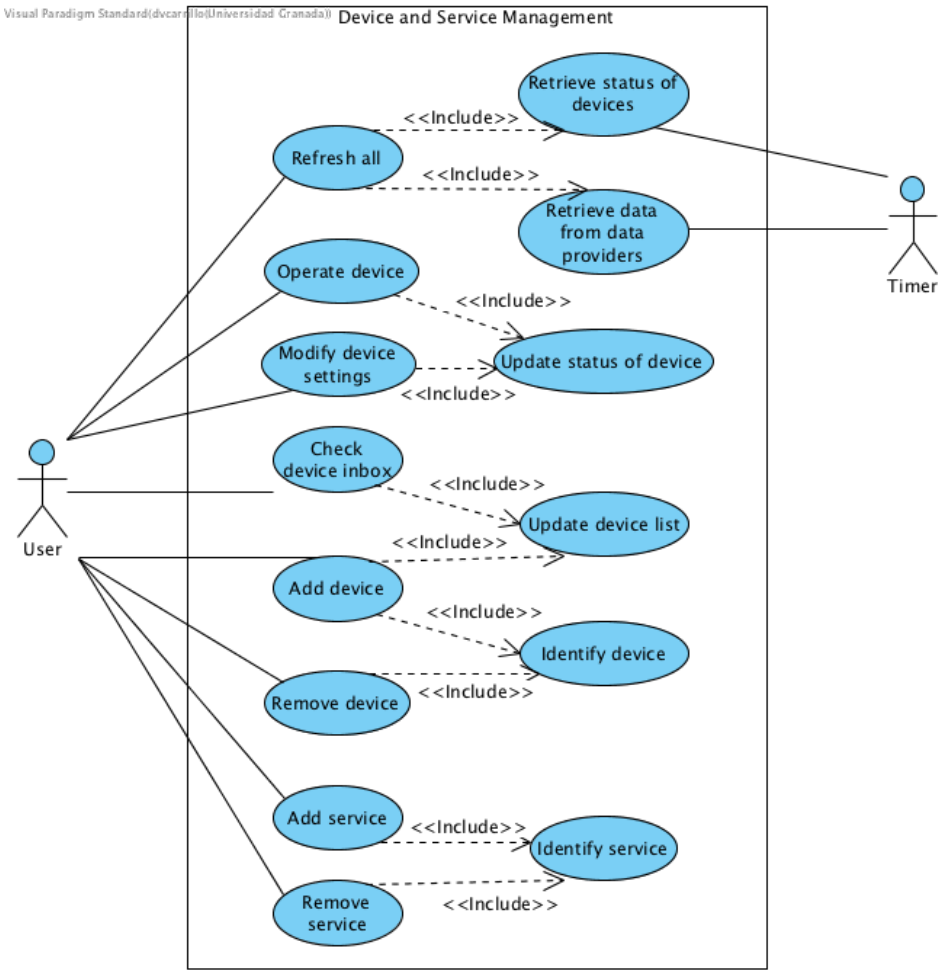

Figure 6.4: Use cases diagram for the Device and Service Management subsystem

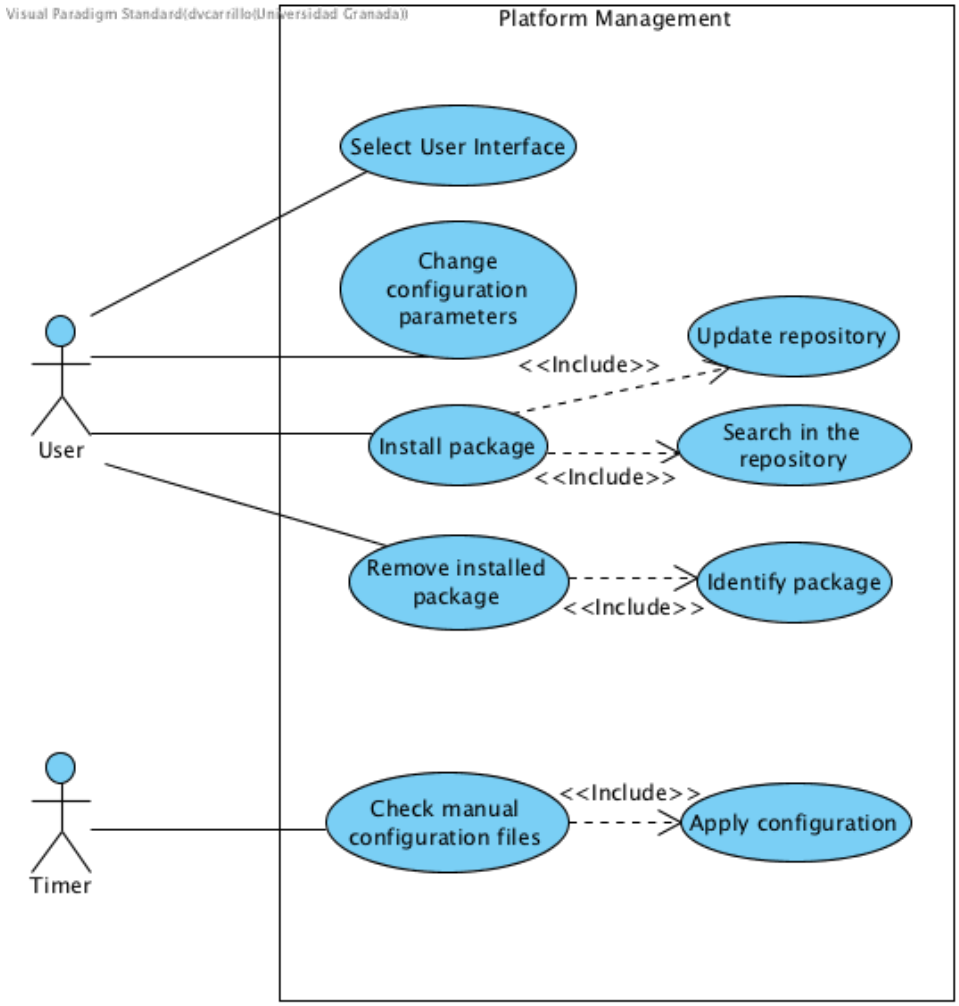

Figure 6.5: Use cases diagram for the Platform Management subsystem

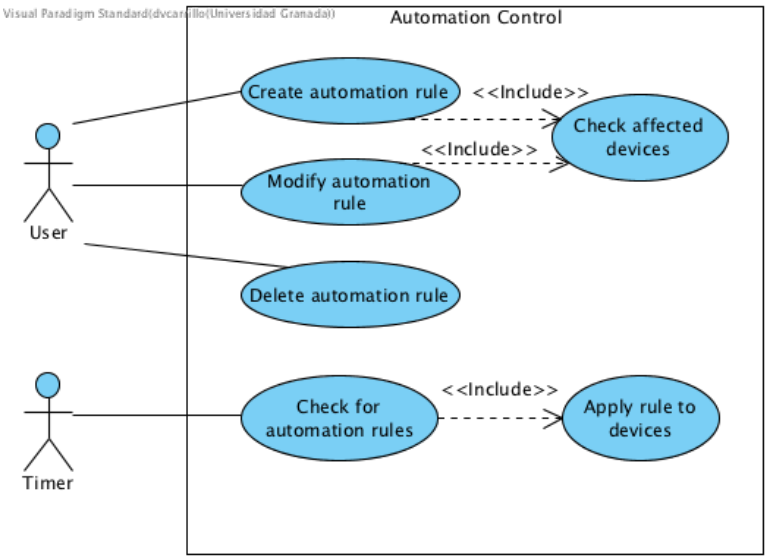

Figure 6.6: Use cases diagram for the Automation Control subsystem

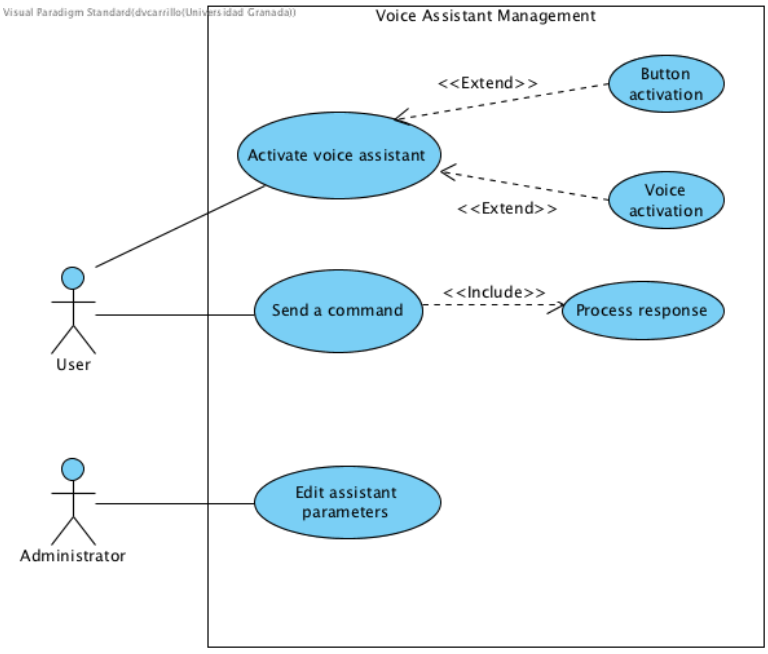

Figure 6.7: Use cases diagram for the Voice Assistant Management subsystem

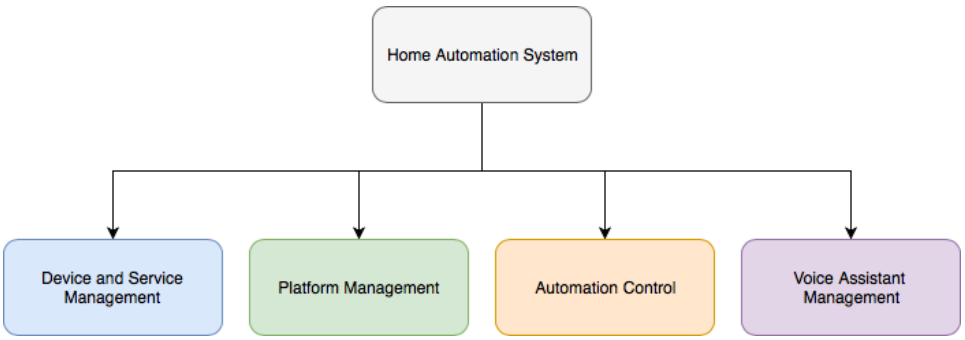

Figure 6.8: Functional subsystems of the home automation controller

Desktop Application

Another solution, focused on the Raspberry Pi desktop (where we mainly want to implement the system), is developing a desktop application with an adapted user interface for tactile screens. The advantage of a desktop application is that it usually requires less resources and it is faster, but it is a specific solution for the Raspberry Pi, so it would only work there. We would need a different application if we wanted it to run on mobile phones or tablets.

We would need to make it from scratch, as openHAB does not provide any desktop application.

Mobile Application

Mobile phones and *tablets* are common devices that are widely used to manage smart homes. Some setups have a tablet located somewhere in the home, which is only used for managing the domotic devices. An application is faster and more accessible than a web platform, and users are more used to them, so it would be convenient in some cases.

However, making the system only accessible from a mobile or tablet application would reduce the utility of the Raspberry Pi, and would at least require two devices (one for the local server and another one for accessing it). Although it is a good complement to the system, it is not convenient to base it only on a mobile application. However, it is applicable in a hybrid solution.

Hybrid Solution

A hybrid solution consists of applying two or more solutions together of those specified above. For example, providing a web platform and an optional

Feature	Web platform	Desktop application	Mobile application	Hybrid Solution
Support for multiple devices	Yes	Only PCs	Only mobile phones/tablets	Yes
Allows access from Internet	Yes	No	No	Yes
Control from Raspberry Pi	Yes	Yes	No	Yes
Easy implementation	Yes	No	No	Yes
Maximum performance	No	Yes	Yes	Partially

Table 6.1: Comparison between possible implementations

mobile application to easily manage the system from a mobile device.

OpenHAB provides a web platform, as explained above, and a mobile application for iOS and Android than connects to the web platform.

Comparison

The table 6.1 compares all the possibilities for the implementation of the system.

As we can see, the hybrid solution, which is a web platform and mobile application, meets all the features in the best way, and offers easy and fast implementation. At this point, I have not considered the implementation of the voice assistant, which I will talk about in following sections.

Figure 6.9: Raspberry Pi 3 Model B

6.2 System Analysis

6.2.1 Conceptual Model

6.3 System Design

6.3.1 Architecture

6.3.2 Conceptual Class Diagram

6.4 Implementation

In this section, I will explain the process that I have followed to implement the home automation controller, along with the voice assistant.

As I have mentioned before, the main objective is to install the system in a Raspberry Pi, which is an affordable and small computer, with enough power and connectivity options to install a home automation system on it, and even a voice assistant. The possibilities of this device are endless: it can work with or without display, and has a special connector to which the user can connect screens, microphones, speakers and many other accessories. It also has USB ports on its high-end models, and accepts all the USB accessories that a normal computer would accept.[42]

The first approach to this system was to install Raspbian, a Linux distribution provided by openHAB, over a Raspberry Pi 3 Model B (figure 6.9). Raspbian has openHAB preinstalled and configured, so it is ready to use. The first idea was to have a computer with openHAB and then a voice assistant in other machine. I installed the distribution on this mini PC, but after reviewing the possible voice assistants, I found that it was not the best way to implement it.

I found the Google AIY Voice Kit, which is a kit provided by Google to makers that want to play around with voice recognition. They provide a package with some accessories for the Raspberry Pi and a cardboard box that is meant to contain the board and the accessories. Then, I reconsidered the architecture. It would be also possible to install openHAB in the same machine as the virtual assistant. This way, we would need just one machine for all. It would be a voice assistant, but with a local server, accessible from all the devices connected to the local network. It would be possible to attach a screen to it as well, so the user can manage the smart home graphically from the same device.

I thought this would be the best solution, and it would still meet all the requirements specified previously. So, I began working on it.

6.4.1 Introduction to Google AIY Voice Kit

The AIY Voice Kit is a do-it-yourself project created by Google that demonstrates how easy and inexpensive can be to create a natural voice recognizer that works with Google Assistant, at a price of only EUR 30 in Europe. The project, aimed for makers, also lets the user add their own questions, which is the most powerful part for our purpose.

The idea is to adapt the device in order to fetch the voice commands that the microphone captures and manage them in openHAB, making the system act following user's instructions.

Capabilities

The main advantage of the Google AIY Voice Kit, and the reason why we have chosen it for this project, is that we can access the Google Cloud Voice API and create our own voice interfaces. Everything is coded in Python, and we can create voice commands to control it.

This, along with the more than reasonable quality of its stereo microphone and speaker, make it a very useful device that could perfectly fit in a smart home, despite its looks.

The device is capable of recognizing the "OK Google" command, as

well triggering the assistant when the big red button is pressed, and, by default, it is capable of doing almost everything that the Google Assistant on smartphones can do, including integration with the apps in the user's smartphone and reading the news, among other things.

The Main Challenge

The challenge is to explore how Google AIY Voice Kit and openHAB 2, both installed on the same system, can work together and, if it is possible, connect them so a standard user can control its smart home using voice commands and universal, open source solutions.

6.4.2 Building the Google AIY Voice Kit

For making the voice recognizer, I primarily used:

- Cardboard for the body of the device.
- A speaker.
- A Voice HAT microphone board and an accessory board.
- A big button with a light inside to trigger the voice recognition.
- A Raspberry Pi Model 3 B.
- An 8GB microSD card.
- Cables to connect everything.

First of all, I had to put all the items together and assemble the cardboard body. The Voice HAT accessory board is connected to the Raspberry Pi through its GPIO connector, which is adapted to connect the rest of devices: the speaker, the microphone board and the button. Once everything was correctly connected, I had to assemble the cardboard body, which was composed by two pieces.[1]

The next step was to write the Voice Kit SD image[2] to the microSD card. I used the *Etcher.io* macOS application for this purpose.

The system includes everything that the Raspberry Pi needs to use the connected devices, so I only needed to check that they worked fine and were correctly connected. The system also includes two demos with Google Assistant: one that triggers the assistant by saying "OK Google!", and another that needs the button to be pressed to trigger it.

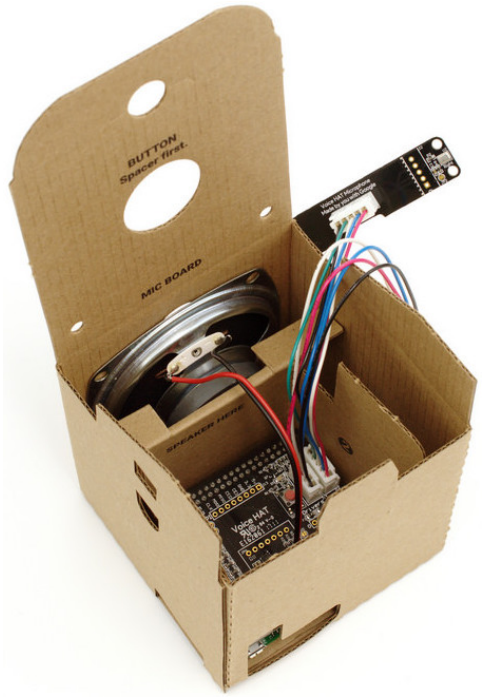

Figure 6.10: The AIY Voice Kit

To make any demo or a new app that uses Google Assistant work, it is necessary to register a new project on Google Cloud Platform (GCP) first. A project of this kind has to include the Google Assistant API, which can be enabled via GCP, as well as an OAuth 2.0 client. Finally, at the moment of running the application, it seeks a JSON file that contains all this information. By default, it must be placed in the home folder and it must be called assistant.json.

Google Assistant gathers all the user's information from their Google account, so they must have enabled the web and app activity, the device information and the voice and audio activity from their Activity Controls panel in their Google account settings.

6.4.3 Setting up openHAB 2

In this case, the goal is to install openHAB in the Linux distribution that the AIY Kit provides, which is a fork of Raspbian (that is a fork of Debian made for the Raspberry Pi), with the suitable drivers in order to support all the accessories from the kit. Luckily, openHAB provide in their documentation pages[35] enough instructions to make this process easier.

In this process, I used a keyboard and a mouse for interacting with the

operating system, but it can also be done remotely via SSH connection.

As a prerequisite, Java 8 must be installed on the system. Users may prefer Zulu, the main Java *alternative*, a fully certified build of OpenJDK.[60]

OpenHAB 2 can be installed though a package repository or manually from file, but it is recommended to install through a package repository, using *apt*, *apt-get*, *yum* or *dnf*. As this operating system is based on Debian, I used *apt-get*.

First of all, I have to add the openHAB 2 Bintray repository key to the package manager and allow *apt* to use the HTTPS Protocol.

```
wget -qO - 'https://bintray.com/user/downloadSubjectPublicKey?username
    =openhab' | sudo apt-key add -
sudo apt-get install apt-transport-https
```

OpenHAB offers Stable (Official), Beta and Snapshot builds to choose from. The stable builds contain the latest official release with tested features, so it is the *safest* to use. I will install this one. Then, I need to add the openHAB 2 Stable Repository to your systems apt sources list

```
echo 'deb https://dl.bintray.com/openhab/apt-repo2 stable main' | sudo
     tee /etc/apt/sources.list.d/openhab2.list
```

Next, I have to resynchronize the package index:

```
sudo apt-get update
```

And now I can install openHAB:

```
sudo apt-get install openhab2
```

Optionally, it is possible to install the add-ons package (*openhab2-addons*), but it is meant to be installed if the machine is going to be disconnected from the Internet, as openHAB downloads them on request by default. In this case, I assume that the system is going to be connected to Internet always.

At this point, I can start openHAB and register it to be automatically executed at system startup. To this end, the documentation provides the following commands for systems based on *systemd*, such as Raspbian.

```
sudo systemctl start openhab2.service
sudo systemctl status openhab2.service

sudo systemctl daemon-reload
sudo systemctl enable openhab2.service
```

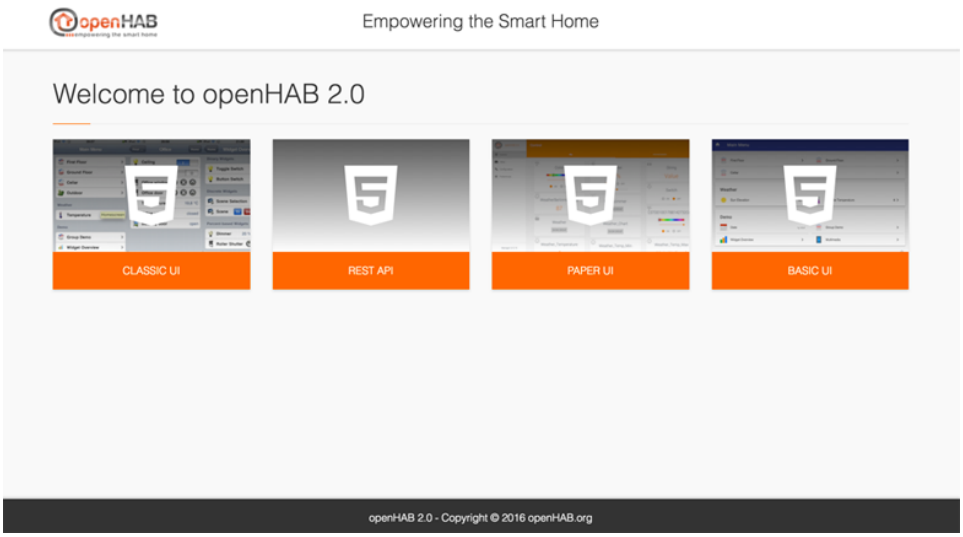

Figure 6.11: OpenHAB 2 startup screen

After some minutes, openHAB is ready on the local server, in the port 8080 (accessible in *http://localhost:8080*). Then, the following commands are used to control the service.

```
# Current service status
sudo /etc/init.d/openhab2 status

# (Re-)Start openHAB (background service)
sudo /etc/init.d/openhab2 restart

# Stop the openHAB background service
sudo /etc/init.d/openhab2 stop

# Make openHAB automatically start after booting the Linux host
sudo update-rc.d openhab2 defaults
```

OpenHAB installs also a utility in the command line interface, named *openhab-cli*, that provides access to the openHAB-specific commands.

```
Usage:  openhab-cli command [options]

Possible commands:
  start [--debug]     -- Starts openHAB in the terminal.
  stop                -- Stops any running instance of openHAB.
  status              -- Checks to see if openHAB is running.
  console             -- Opens the openHAB console.
  backup [filename]   -- Stores the current configuration of openHAB.
  restore filename    -- Restores the openHAB configuration from a
      backup.
  showlogs            -- Displays the log messages of openHAB.
  info                -- Displays distribution information.
```

Lastly, to stay up to date to new releases, they recommend to execute the following commands periodically. Note that this also applies to the rest of the packages installed in the Linux system.

```
sudo apt-get update
sudo apt-get upgrade
```

6.4.4 Adding Philips Hue Devices

Now that we have openHAB 2 up and running in the Raspberry Pi integrated in the AIY Kit, it is desirable to add a device to the home automation system. We have a Philips Hue lightning system available, so in this section I will explain the process that we have followed to add it and configure it in our openHAB 2 instance.

The Hue System

Our Philips Hue system is composed by the *white and color ambiance starter kit*[40] (which consists in three Hue white and color lights and a Hue hub) as well as a Hue motion sensor:

- **The Hue Hub** acts as a bridge between the user interface (the smartphone application or the OpenHab instance) and the connected devices. It is configured and controlled via the Hue app and supports up to 50 lights at the same time. It communicates via Zigbee with the Hue devices.
- **The Hue Motion Sensor** can be configured to turn on and off the lights when it detects movement and under some defined conditions. By default, it is configured from the Hue app.
- **The Hue White and Color Lights** are customizable RGB lights. By default, they are configurable from the Hue app, which offers many ways of changing the light emission.

Installation and Configuration Process on PaperUI

Philips, as many other companies, is restrictive regarding their home automation devices, and every communication between the Hue devices and the controller (openHAB in this case) needs to pass through the Hue Hub. The Hub communicates via WiFi with the controller and via Zigbee with the Hue devices. The software it uses is privative, but luckily openHAB 2 implements protocols that make possible the communication between the

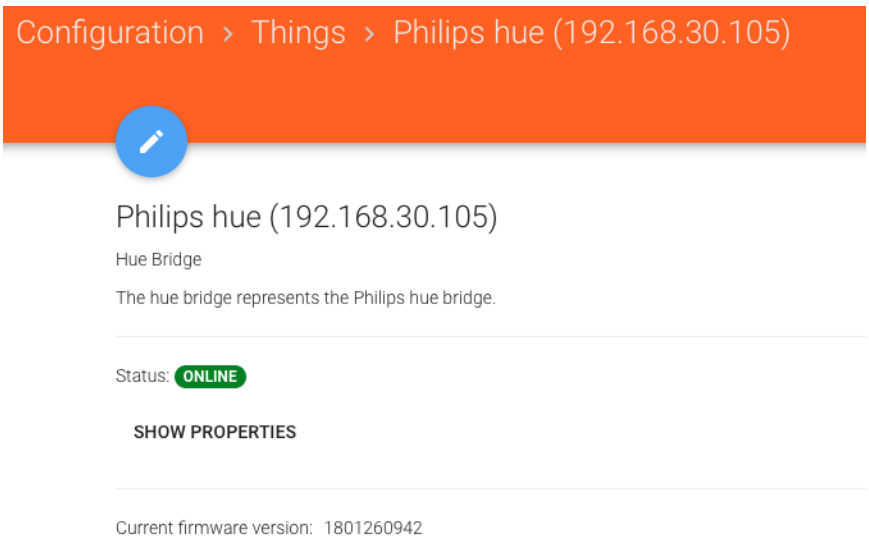

Figure 6.12: Philips Hue Hub Thing

controller and the Hub. This makes the Hub, though, a frustrating addition to our home automation system.

The Hue system needs to be configured from the Philips Hue mobile application in the first place. This process will link the Hub with the desired Hue devices, so the next time that the user connects to the Hub, it will not be necessary to configure them again, even if the connection is done from openHAB. The application allows to manage rooms and sort the devices, as well as setting automated commands (for instance, turning the light on when the motion sensor detects any movement). OpenHAB is unable to perform such complex operations by default, providing only a few channels for the light bulb.

Once the Hub is on, connected and linked with the Hue devices, we can proceed to configure it on openHAB. In this case, I explain the process to follow in order to add the light bulb controls, but it might work for other Hue devices in a future. If openHAB is used on a system connected to the same network as the Hub, the Hub will automatically appear in the *Inbox* section. We have to make sure before that its related binding. Adding the Hub in the Inbox section means that it will be listed as a Thing. OpenHAB takes care automatically of the Thing (figure 6.12) and Items configuration. After adding them, the host device and the Hub will be able to communicate between them. However, the user needs to press the pairing button on the Hub in order to activate the communication.

After that, the Hub will act as a bridge and openHAB will be able to

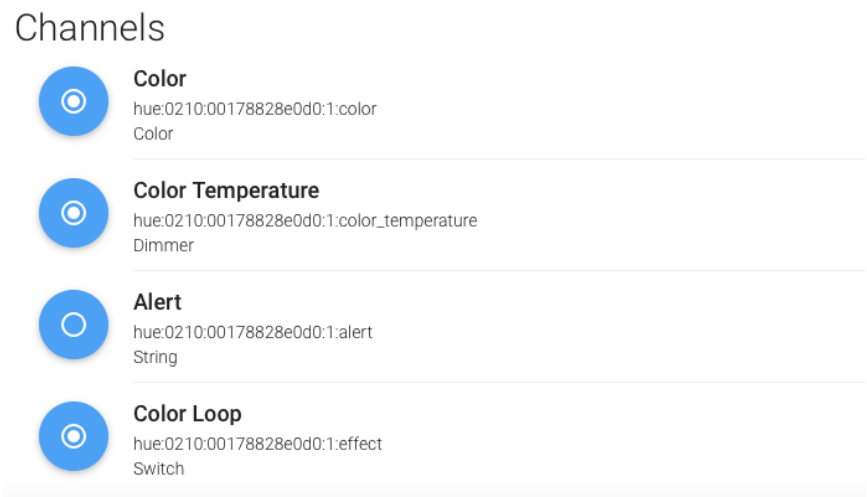

Figure 6.13: Available channels in the Hue color bulb

display all the compatible devices connected to the Hub. In this case, the Hue color lamp appears in the Inbox. The process for adding it is the same as with the Hub. Nevertheless, this time the light controls will be added automatically to the Control panel according to the linked channels on the thing. OpenHAB supports two channels that are directly related to the bulb functionality: *color* (which is divided in tone, brightness and saturation) and *white temperature*. The first one will set the light in a color with the desired properties, and the second will set a white color from a range of whites (from cold to warm white). Additionally, OpenHAB provides two more channels: *alert*, to use the bulb as an alert light under certain circumstances, and *color loop*, which will make the light color iterate in the whole color spectrum. The figure 6.13 shows the PaperUI interface showing these channels.

Internal Functionality

OpenHAB is able to communicate with the Hue Hub and Hue devices thanks to the Philips Hue binding (represented as a set of packages in Java, as shown in the figure 6.15), integrated in the Eclipse SmartHome Extensions repository.

As we can see, there are packages for handling the lights and Hub (which is named *bridge*), and for managing the internal functionality of the binding and openHAB. That is, discovering the devices and showing them in the Inbox and managing exceptions. Examples of exceptions can be having the device off or sending the device a wrong command.

Looking more closely at the device handlers, we can see that the bridge

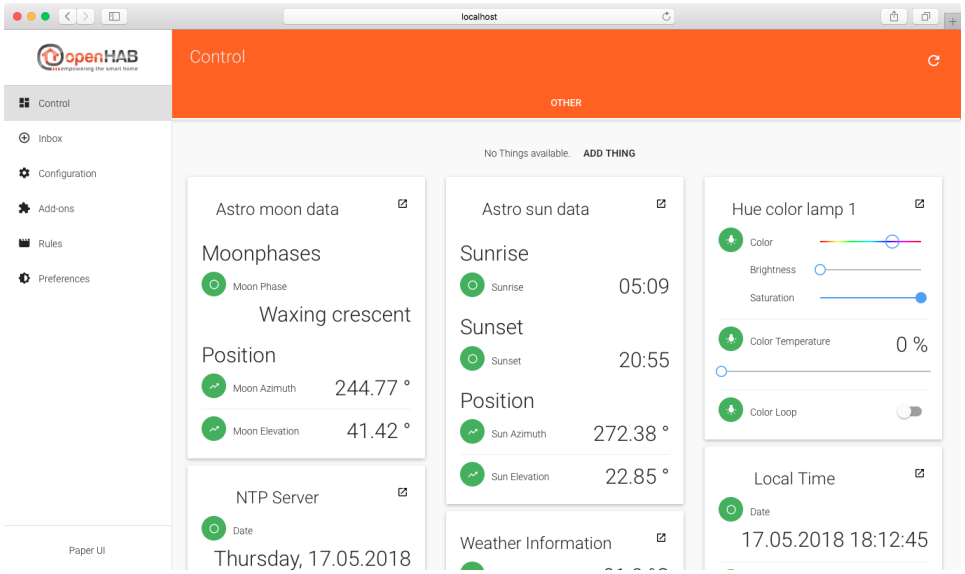

Figure 6.14: OpenHAB 2 Control Panel with the Hue color bulb controls

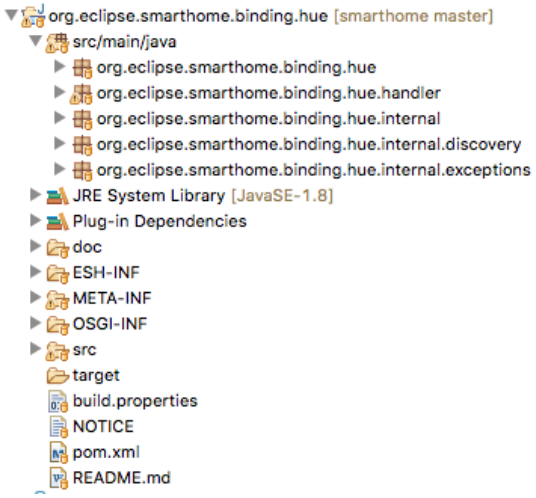

Figure 6.15: Philips Hue binding internal structure

handler implements a *Runnable* object with a *run* method, which gets the Hub configuration. The function gets the IDs of all the lights connected to the Hub and iterates over them. If a light is not listed in a list of light states caught from openHAB, it is added as a new light. If it is listed, then it checks for changes in the status of the light, comparing the actual state with the last one saved. If the states are different, notifies the Light listeners. There is a *LightStatusListener* interface that is called on each change, it will modify the Light and Hue objects.

6.4.5 Specifying Items Manually

In previous chapters, I introduced the concept of Item in openHAB 2. Items represent functionalities that can be used by the applications, mainly user interfaces or automation logic.

OpenHAB is able to configure many Things and Items automatically in the PaperUI. But in many cases we may need to specify Items manually. For example, for creating custom views(sitemaps), as we will see below. Therefore, it is important to know how to specify Items manually.

First of all, I need to create an Items file. Both the items and the sitemap files are located in the $OPENHAB_CONF directory, which is different on different operating systems. In my case, they are located in:

```
/usr/share/openhab2/conf/items       <-- *.items files
/usr/share/openhab2/conf/sitemaps    <-- *.sitemap files
```

After a fresh installation these directories are empty (except for the *readme* files), so I have to create a file there. I will call it *default.items*.

OpenHAB has its own syntax for defining Items and Sitemaps, but it is very easy to use. The basic syntax for defining an item is:

```
ItemType ItemName "ItemDescription" <ItemIcon> {ItemToThingChannelLink
    }
```

The code I have used for defining the item linked to the color channel and the item linked to the white tone channel of a Philips Hue color bulb is the following:

```
Color HueColor1 "Hue Light Bulb 1 Color" <lightbulb> {channel="hue
    :0210:00178828e0d0:1:color"}

Dimmer HueDimmer1 "Hue Light Bulb 1 Temperature" <lightbulb> {channel
    ="hue:0210:00178828e0d0:1:color_temperature"}
```

Now, these Items are registered in the system, but before we can see

them, I have to define a sitemap and include them there.

6.4.6 Creating a Sitemap

PaperUI, the newest addition in openHAB 2, is meant to make easier the device management, configuration and discovery. Many of these processes are carried out seamlessly, without user intervention. But this user interface lacks some functionalities, such as custom ordering of Things. *Sitemaps* are custom views that can be displayed in another user interface, the Basic UI, which is also automatically installed at the beginning.[35] Sitemaps take defined Items and display them as the user specifies in the UI.

The user is able to define as many sitemaps as desired, and they can be selected from openHAB 2 home screen, in the Basic UI. Sitemaps are text files with the .sitemap extension, and they are defined in the folder thet I specified in the previous section, inside the openHAB installation directory.

The basic syntax that a sitemap follows is:

```
sitemap <sitemapname> label="<title of the main screen>"
{
    [all sitemap elements]
}
```

Sitemaps are composed by arranging various user interface elements. A set of different element types supports a user-friendly and clear presentation. One line of Sitemap element definition produces one corresponding UI element. As shown in the example on the figure 6.16, each element generates a descriptive text next to an icon on the left side and a status and/or interaction elements on the right.

A certain set of parameters can be configured to customize the presentation of an element. In the shown example *item* and *icon* are parameters. Almost all parameters are optional, some are however needed to result in a meaningful user interface.

By encapsulating elements with curly brackets, multiple elements can be nested inside or behind others. The Frame element type is often used in combination with element blocks. Frames are used to visually distinguish multiple elements of the same topic on one interface page. When using code blocks behind other element types such as *Text*, *Group* or *Switch*, these UI elements will, in addition to their normal function, be links to a new view, presenting the nested elements. In the example in 6.16, I created a single Frame that represented all the functionalities of the Hue Color Light.

The *sitemap* element is mandatory in a Sitemap definition. This element shall be the first line in the sitemap file, and the following code block

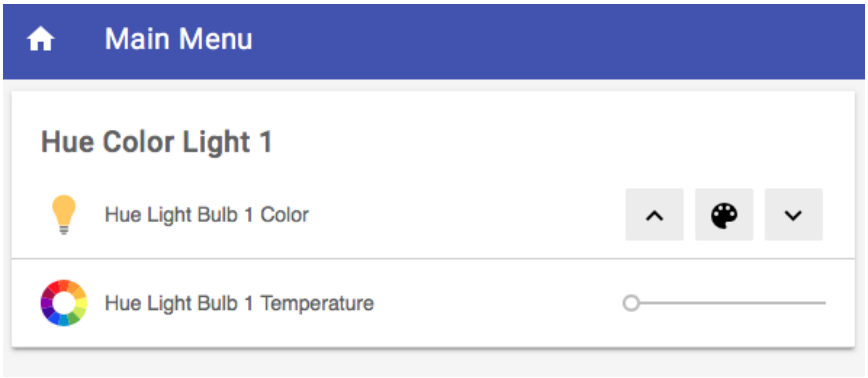

Figure 6.16: Basic UI displaying a Hue Color Light Item

comprises the entire Sitemap definition.[35]

From the previous example, I built a sitemap that could display the Hue color light Item that I configured previously in the Basic UI. The figure 6.16 shows the result of introducing the following code in a file that I called *default.sitemap*.

```
sitemap demo label="Main Menu"
{
    Frame label="Hue Color Light 1" {
        Colorpicker item=HueColor1 icon="slider"
        Slider item=HueDimmer1 icon="colorwheel"
    }
}
```

A frame is containing the main functionality of the Hue color light bulb, and that can be done for every new component installed at home.

OpenHAB 2 allows us to group our items so we can sort them by room or type. That can be made through the Group Item type.

Elements on a Sitemap

The element types specified in the table 6.2 may be used in a Sitemap definition file.

Data presented by Sitemap elements will almost always originate from a referenced Item. Each Item is of a certain Item type, for example *Switch*, *Number* or *String*.

While not all combinations are meaningful, Items of one *datatype* may be linked to different Sitemap element types. This provides the flexibility to present Items in the way desired in the home automation user interface.

Type	Description
Chart	Adds a time-series chart object for persisted data
Colorpicker	Allows the user to choose a color from a color wheel
Default	Renders an Item in the default UI representation specified by the type of the given Item
Frame	Establishes an area containing various other Sitemap elements
Group	Concentrates all elements of a given group in a nested block
Image	Renders an image given by an URL
Mapview	Displays an OSM map based on a given Location Item
Selection	Provides a dropdown or modal popup presenting values to choose from for an Item
Setpoint	Renders a value between an increase and a decrease buttons
Slider	Presents a value in a progress-bar-like slider
Switch	Renders an Item as an ON/OFF or multi-button switch
Text	Renders an Item as text
Video	Displays a video stream, given a direct URL
Webview	Displays the content of a webpage

Table 6.2: Types of elements for a Sitemap in openHAB 2

At this point, I have openHAB installed in the Raspberry Pi inside the AIY Voice Kit. OpenHAB is communicating with the Hue Hub and the Hue Color Light, and its state can be changed by accessing to the server that openHAB has created in the local network. This means that, if I type on the browser of my mobile phone the IP address of openHAB and its port (8080), I will be able to control the light from my phone as well. The state of the bulb can be changed from PaperUI (with the view that it has generated automatically) and from Basic UI, with the view and Items that I have specified in these previous steps.

Now the challenge is to build an assistant that can communicate with openHAB, and that is able to change the state of the configured Items.

6.4.7 Voice Assistants and Speech-To-Text Services

The Google AIY Voice Kit, the device I am building the system on, already provides some files to access the Google Assistant. This means that, by default, it can work just like any Google Home device (I explored these devices in previous chapters), so it can provide general information and knowledge, play songs and tell the weather forecast, among other things. The files that the AIY Voice Kit provide are accessible and modifiable, so I could write an assistant that would work on top of the Google Assistant, with special commands.

Introduction

We could divide a simple voice assistant, like the one I am building on top of the Google Assistant for managing openHAB and the Raspberry Pi, in three different parts:

1. Speech-To-Text (STT) conversion.
2. Command processing.
3. Text-To-Speech (TTS) conversion.

The flow after each command from the user follows the order above. On the next few lines, I am going to explain a little further each one of these points.

Speech-To-Text

The Speech-To-Text or STT comes from the Speech Recognition, a sub-field of computational linguistics and IA. It is a computer technique that enables

the recognition and transcription of spoken language into text by analyzing the captured audio.

It is the first step on the machine after capturing the voice, and it requires the execution of complex algorithms. This is why many companies, like Google, offer "cloud speech recognition", where the audio is entirely processed in the cloud, taking advantage of very powerful computers. Thanks to this, its precision is much higher than other local solutions.

Command Processing

Once the system has the transcription of the spoken command, it must relate it with an action. It can be the execution of a function, or just a spoken response, or both. The complexity of this part can vary significantly, depending on the range of actions we would like the assistant to perform, or on how specific we want to be when recognizing the language.

There are grammar specifications and even XML dialects for this purpose, like AIML. For the custom assistant I have built, I also propose a model for processing the speech, as I will mention in further sections.

Text-To-Speech

Text-To-Speech or TTS is the inverse process of STT, that is, the artificial production of human speech given a text through a number of processes that produce a pronounceable version of the text and a digital sound, thanks to signal treatment techniques.

It is often the final step in the interaction with an assistant, when the assistant gives a response to a command given by the user.

STT and TTS in the Custom Assistant

The part where I have researched the most, which is also the one that offers more possibilities to this kind of assistant, is the speech recognition. I have found plenty of services that offer this technology, locally and in the cloud. Many of them work on Raspberry Pi, the heart of the Google AIY Kit.

Google integrates in this kit the integration with two STT services: the Google Assistant Library and the Google Cloud Speech-To-Text API[17]. Both of them make the transcription in the cloud, freeing the Raspberry Pi from using much processing power in it. Both of them are truly accurate and rarely produce a wrong result, as it is expected from Google. Using this implementation of the Google Assistant Library is free, but it only works in English at this moment, so it is only possible to make the assistant in this

Name	Assistant Library	Cloud TTS[17]	Pocketsphinx[11]	Watson STT[22][21]
Maker	Google	Google	CMUSphinx	IBM
Location	Cloud	Cloud	Local	Cloud
Voice HAT support	Yes	Yes	No	No
Required space	Insignificant	Insignificant	40 MB + language models	Insignificant
Recognition quality	Excellent	Excellent	Good	Excellent
Languages	English	120 supported	13 supported[1]	13 supported
Prince	Free (comes with the AIY Kit)	$0.006 USD/15 seconds after 60 minutes/mo.	Free	$0.02 USD/min. after 1,000 minutes/mo.[2]
Open source	No	No	Yes	No

Table 6.3: Comparison of Speech-To-Text services

language with this service. On the other hand, the Google Cloud Speech API is capable of recognizing more than 120 languages with its variants, even with long audio files. It can also provide content filters and automatic punctuation. But this solution is not totally free, it comes with a price of $0.006 USD per 15 seconds of audio after 60 minutes, each month.[12][16]

Outside of Google, there are many other services that could also work with our assistant, from local solutions like CMUSphinx[11] to cloud services provided by other companies. Local solutions offer offline recognition, in exchange for taking up large storage space and being slower and a lot more inaccurate than the cloud ones. Furthermore, cloud services are constantly updated and offer state-of-the-art voice recognition. They are usually very fast and, as they work in the cloud, do not need space on the user's drive. Usually, the providers of these services are major technology companies such as Microsoft and IBM[22], which provide the API for a periodic fee.

To find out which service best fits in my custom assistant, I compare the most interesting ones I have found in the table 6.3.

I have included a selection of services that can sum up the benefits and problems of all the available services. The main difference is if they are in the cloud or installed locally. Generally, cloud services will provide great results, but they are not free in most cases. Local services can be free and open source, but their recognition is usually poorer, and they require some storage space that we might need for other things.

Then, in our specific case, Voice HAT support is a great matter. This is

[1]There are at the moment 13 language models available to download in their website. Although new languages can be added by creating new language models

[2]This price is maintained until 250,000 minutes are used. The complete set of prices is: $0.02 USD for minutes 1,001 - 250,000, $0.015 USD for minutes 250,001 - 500,000, $0.0125 USD for minutes 500,001 - 1,000,000, $0.01 USD for minutes 1,000,001 and up

a device made by Google and included in the AIY Voice Kit, which connects the microphone and other components of the assistant with the Raspberry Pi. Google services support it natively, but not the others, which recommend using a USB microphone, like the implementation of IBM Watson STT for Raspberry Pi. But I cannot get rid of the HAT, because I need it for managing the speaker and the button. In conclusion, these other options do not seem viable considering the effort and modifications they need in order to work properly.

Therefore, when comparing the solutions provided by Google, the only advantage of the Cloud Text-To-Speech service is that it offers support for many other languages. Although this option is very easy to implement (Google even includes an example of its usage with the AIY Kit), considering the main points in this project I decided to keep the assistant free of charge and working in English.

6.4.8 Bulding the Custom Voice Assistant

The best part of the Google AIY Voice Kit is that it is fully customizable and backed up by the Google Assistant API. This device is meant to be fully modifiable, so each user can play with it and use it for their needs.

In this case, I are going to take advantage of this fact to build our own assistant, able to process custom commands that will work with openHAB.

The AIY Voice Kit is able to process button taps and to recognize the "OK Google" command in order to trigger the assistant. In our case, I can use both triggers, so the voice assistant is launched the same way no matter what way the user choses each time.

The Base Assistant

For the programmer, it is really easy to create customized assistants. Google provides some examples that basically triggers the Google Assistant by pressing the button or saying the "OK Google" command. I am going to work over them to reach our objective.

Below is a Python program that activates the assistant by pressing the red button.

```
1 import logging
2
3 import aiy.assistant.grpc
4 import aiy.audio
5 import aiy.voicehat
6
7 logging.basicConfig(
8     level=logging.INFO,
```

```
9       format="[%(asctime)s] %(levelname)s:%(name)s:%(message)s"
10  )
11
12
13  def main():
14          status_ui = aiy.voicehat.get_status_ui()
15          status_ui.status('starting')
16          assistant = aiy.assistant.grpc.get_assistant()
17          button = aiy.voicehat.get_button()
18          with aiy.audio.get_recorder():
19          while True:
20              status_ui.status('ready')
21              print('Press the button and speak')
22              button.wait_for_press()
23              status_ui.status('listening')
24              print('Listening...')
25              text, audio = assistant.recognize()
26              if text is not None:
27                  if text == 'goodbye':
28                      status_ui.status('stopping')
29                      print('Bye!')
30                      break
31                  print('You said "', text, '"')
32              if audio is not None:
33                  aiy.audio.play_audio(audio)
34
35
36  if __name__ == '__main__':
37      main()
```

We can see that in the previous script there are some statuses for the assistant defined, although the script is pretty simple:

- Ready to listen
- Listening
- Stopping

We can also notice that this script only sends the commands to the assistant, an object that works with Google Assistant, catches its responses and plays them on the speaker. This means that no processing is made in our computer, Google Assistant takes care of the commands that the user sends and composes a response according to that.

Making the Custom Assistant

The previous script falls short for our purpose, as it does not include any integration with openHAB and it only triggers the assistant when the red button on the Google AIY box is pressed.

I worked over this example script, and some more that Google brings with this kit, in order to create a new one that integrates all the basic functionalities and works with openHAB. The main aspects to include are:

- Activation via red button and "OK Google" command at the same time.
- Add custom commands to the assistant:
  - Ability to manage openHAB from the assistant.
  - Ability to manage the Raspberry Pi.
- Possibility to integrate more languages.

Activating the Assistant via Voice Commands and by Pressing the Red Button I wanted users to be able to trigger the assistant in all the possible ways, so the first task is to make a script that can trigger the assistant by pressing the button and by saying "OK Google".

We can make the program listen for "OK Google" by importing the Google Assistant Library. It will be listening on the microphone all the time waiting for this command, as any Android phone with Google Assistant would do. This library has direct access to the audio API, so our Python program does not need to record any audio.

However, the Google Assistant Library event loop (which is the responsible of hearing the "OK Google" command) blocks the running thread, so we cannot place everything in the same thread because the function that is executed when the button is pressed would never be invoked. We need to import threading and run the event loop in a separate thread.

To simplify matters, I will create a class called MyAssistant, which will create the threads in the constructor and will have a variable that will indicate if it is ready and an object of the Google Assistant.

```
1  class MyAssistant(object):
2
3    def __init__(self):
4        self._task = threading.Thread(target=self._run_task)
5        self._can_start_conversation = False
6        self._assistant = None
7
8    def start(self):
9        self._task.start()
10
11   def _run_task(self):
12       credentials = aiy.assistant.auth_helpers.
             get_assistant_credentials()
13       model_id, device_id = aiy.assistant.device_helpers.get_ids(
             credentials)
14       with Assistant(credentials, model_id) as assistant:
15           self._assistant = assistant
16           for event in assistant.start():
17               self._process_event(event)
18
19 def _on_button_pressed(self):
```

```
20 if self._can_start_conversation:
21         self._assistant.start_conversation()
```

This code snippet specifies just what we want to create at this moment. It has a call to the function *_process_event()*, which will process our events. I will talk about it in the following sections.

Adding Custom Commands to the Assistant The Google Assistant responds to a huge variety of commands and can perform a considerable range of actions. Some devices even work with Google Home, but not many others that do work with openHAB 2. As we want to achieve an inexpensive, open-source home automation system, we need to integrate the assistant with openHAB 2, although this solution will also cover the devices that are compatible with Google Home, that are natively integrated with the assistant.

Unfortunately, we cannot customize Google Assistant commands, but we can build a basic assistant over this one that will cover our needs. Both will be working at the same time as well.

I will use the Text-To-Speech engine that Raspbian integrates: *pico2wave*. In the function *_process_event()* that I mentioned before, we need to add some conditions for checking some "keywords" that will lead the program to stop the assistant for that command and execute our own one.

Apart from commands for managing openHAB 2, it is also a good idea to be able to manage the system from the assistant, as it is meant to be headless (that is, able to work without a screen connected) and powering off the Raspberry Pi by disconnecting the power cord is not recommendable. For now, I am going to add functions for powering off and rebooting the system:

```
1 def power_off_pi():
2   aiy.audio.say('Powering off the system. Good bye!')
3   subprocess.call('sudo shutdown now', shell=True)
4
5 def reboot_pi():
6   aiy.audio.say('Rebooting the system. Hold on!')
7   subprocess.call('sudo reboot', shell=True)
```

In each one, we define what the assistant is going to say when the function is executed with *aiy.audio.say()* and what command to execute in the system with *subprocess.call()*.

Then, with only one function we will be able to send any command to openHAB via REST. I have defined the URL in the same machine, as the Raspberry Pi is supposed to host the assistant and openHAB. The function creates a POST request that includes the item(s) and the desired state.

When it gets the response, it responds with the success or failure of the command.

```
def openhab_command(item, state):
  url = 'http://localhost:8080/rest/items/' + item
  headers = {'Content-Type': 'text/plain',
             'Accept':       'application/json'}
  response = requests.post(url, headers=headers, data=state)
  if response.status_code == 200:
      aiy.audio.say('OK, command sent to OpenHAB')
  elif response.status_code == 400:
      aiy.audio.say('There has been an error: bad command')
  elif response.status_code == 404:
      aiy.audio.say('There has been an error: unknown item')
  else:
      aiy.audio.say('Command failed')
```

To execute these commands, and to trigger the assistant when necessary, we need the *_process_event()* function. It receives an event each time it is called, that includes a state. We will read this state to determine what to do in each step and the state of the assistant. We will consider the following ones in the *_process_event()* function:

- **ON_START_FINISHED**: the assistant is ready to receive commands and it is set as *ready*.
- **ON_CONVERSATION_TURN_STARTED**: the user has started speaking and the assistant is hearing. It is set to *listening* state.
- **ON_END_OF_UTTERANCE**: the user has finished speaking and now the assistant must process its command. The assistant is set to *thinking* state.
- **ON_RECOGNIZING_SPEECH_FINISHED**: the assistant has finished recognizing the command. Its state does not change in this case, but here is where it processes the custom commands and calls to their respective functions.
- **ON_CONVERSATION_TURN_FINISHED**: the conversation has ended, and the assistant is ready again. It is set to *ready* state.
- **ON_ASSISTANT_ERROR**: there has been an error and, if it is fatal, it stops the execution.

For the custom commands, we just need to check the text that is included when the event is *ON_RECOGNIZING_SPEECH_FINISHED*. The program will recognize the custom commands by looking for substrings inside the speech of the user. This part of the code takes care of it:

```
elif event.type == EventType.ON_RECOGNIZING_SPEECH_FINISHED and event.
    args:
  print('You said:', event.args['text'])
  text = event.args['text'].lower()
  if 'power off the system' in text:
    self._assistant.stop_conversation()
    power_off_pi()
  elif 'reboot the system' in text:
    self._assistant.stop_conversation()
    reboot_pi()
  elif 'make a test' in text:
    self._assistant.stop_conversation()
    test_speech()
  elif 'all the lights on' in text:
    self._assistant.stop_conversation()
    openhab_command("ALL_LIGHTS", "ON")
  elif 'all the lights off' in text:
    self._assistant.stop_conversation()
    openhab_command("ALL_LIGHTS", "OFF")
```

So, if the user says, *Can you please power off the system?*, the program will call the function *power_off_pi()*, specified above. But that would also work if the user says, "Power off the system". More flexibility in the voice recognition can be added by reducing the granularity of the substring search.

Integrating More Languages in the Assistant A desirable function is that the assistant can understand and respond in languages other than English, so users from any part of the world would be able to use it.

However, the Google Assistant is a rather young project and there is not event an official version for many languages. Google does not offer any documentation either about changing the language of the AIY Kit. Some users have reported that they have been able to change the languages of their AIY Kits, but as far as I have researched, it is a complex process that can lead to secondary problems, which is neither intuitive nor usable by the standard user. So, for now, I've decided to leave this issue for the Google Assistant part.

Luckily, we can make the assistant speak other languages for our custom commands, because it depends on *pico2wave*, the TTS engine that comes with this distribution. In my case, I have set the en-GB voice, that sounds much clearer than the default en-US. But it is possible to modify the responses and write them in other languages, and then set the voice according to it. This way, the assistant will be able to respond in other languages. For example:

- Setting the assistant in English from Great Britain:
  aiy.i18n.set_language_code('en-GB')
- Setting the assistant in Spanish from Spain:

aiy.i18n.set_language_code('es-ES')

- Setting only a response in Spanish from Spain:
  aiy.audio.say('encendiendo luces', lang='es-ES')

Nevertheless, it is more complicated when it comes to recognizing speeches in different languages. In my script, I have used the Google Assistant Library, which does the job, but only in English. The best alternative for professional voice recognition, that detects more than 110 languages, is the Google Cloud Speech API, a paid option. Though if we would like to keep the whole system open source, a good alternative is CMUSphinx[11] (*pocketsphinx* in the Raspberry Pi), which is language-independent but needs an acoustic model and a language model.

Managing All the Lights With One Command

In the previous examples, when the script had to process the command *turn on/off all the lights*, a call was produced, indicating the item(s) and their new state (*ON* or *OFF*). In this case, I did the call to *ALL_LIGHTS*, which we can assume is a group composed by all the lights in our home.

At this point, it is imperative to underline the importance of groups in this system. Groups bring a great deal of flexibility to the system, as they allow users to send commands to a specific part of the home, or to a specific kind of devices, just with one command.

Luckily, OpenHAB 2 allows the user to specify as many groups as possible, and to make an item part of one or many groups.

Groups in openHAB 2 For OpenHAB, groups are a special type of Item. As I have mentioned before, Items represent a functionality of the real entities (e.g. the color of a Philips Hue light bulb) that is used by the application. There are Items that represent dates, images or locations, and then there is the group Item.

Group items collect other Items into groups and can themselves be members of other group items.

OpenHAB 2 presents two ways of specifying groups. The first one is inherited from their previous version (OpenHAB 1 and its Basic UI) and requires to manually edit the **.items* files. For example, if we would like to collect two items into a group item, called *Lights_ALL*, the items file should have a structure like the following one:

```
/* Group all the lights */
Group Lights_ALL
```

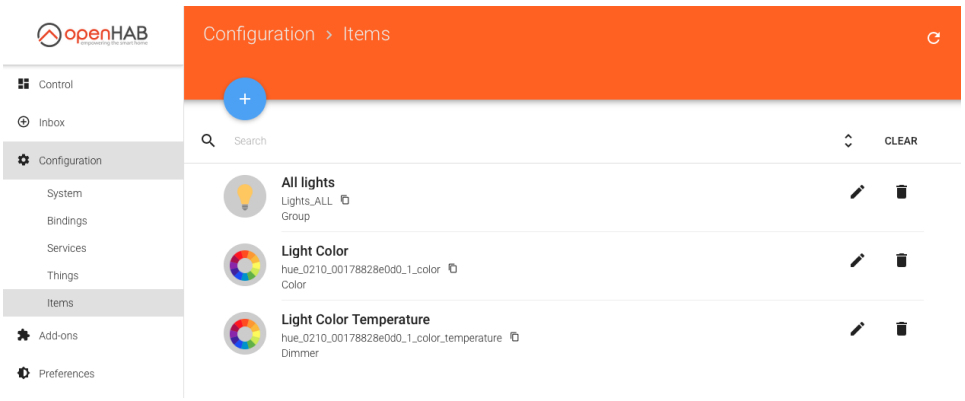

Figure 6.17: Definition of group Items in PaperUI, OpenHAB 2

```
/* Philips Hue Light 1 */
Color HueColor1 "Light Color" (Lights_ALL)
Dimmer HueDimmer1 "Light Color Temperature" (Lights_ALL)
```

The second way is to specify the groups in PaperUI, the GUI that comes with OpenHAB 2. This is one of the most important new features in OpenHAB 2, and they encourage its use, as many processes are greatly simplified. The Items defined in the items file do not appear in PaperUI, so in my case I have to create the same ones but in this UI. The final result, which is literally the translation of the previous code in PaperUI, is shown in the figure 6.17.

Once we have these items grouped in PaperUI, it is recommendable to check if everything is working OK and ready to receive commands from the assistant. A GET request to the following URL:

http://192.168.30.103:8080/rest/items?recursive=false

Returns the following JSON, assuming that 192.168.30.103 is the IP of the machine.

```
[
  {
    "link": "http://192.168.30.103:8080/rest/items/
        hue_0210_00178828e0d0_1_color",
    "state": "OFF",
    "type": "Color",
    "name": "hue_0210_00178828e0d0_1_color",
    "label": "Light Color",
    "category": "colorwheel",
    "tags": [],
    "groupNames": [
```

```
      "Lights_ALL"
    ]
  },
  {
    "link": "http://192.168.30.103:8080/rest/items/
        hue_0210_00178828e0d0_1_color_temperature",
    "state": "OFF",
    "type": "Dimmer",
    "name": "hue_0210_00178828e0d0_1_color_temperature",
    "label": "Light Color Temperature",
    "category": "colorwheel",
    "tags": [],
    "groupNames": [
      "Lights_ALL"
    ]
  },
  {
    "members": [],
    "link": "http://192.168.30.103:8080/rest/items/Lights_ALL",
    "state": "OFF",
    "type": "Group",
    "name": "Lights_ALL",
    "label": "All lights",
    "category": "light",
    "tags": [],
    "groupNames": []
  }
]
```


As we can see, all the specified items are working and accessible from the REST API. As the script sends PUT requests to this API, we can tell that it will reach the group and change all the states of its members.

Finally, the script that manages the assistant should be modified according to the number of groups the user has created, and their names. I will explore this further in the following section.

Final Result

The final script triggers the assistant by pressing the button and by saying "OK Google", has some customized commands for power and OpenHAB management, and supports responses in a variety of languages. But there is still room to improve, as I will state in the next section.

6.4.9 Improving Command Processing

The current structure of the custom assistant script can recognize portions (substrings) on the transcribed command. Although this solution works, it requires the user to know exactly what to say to trigger his or her desired command, losing all the flexibility that an assistant can have. Moreover, it is a very clumsy solution if we would like to add more custom commands,

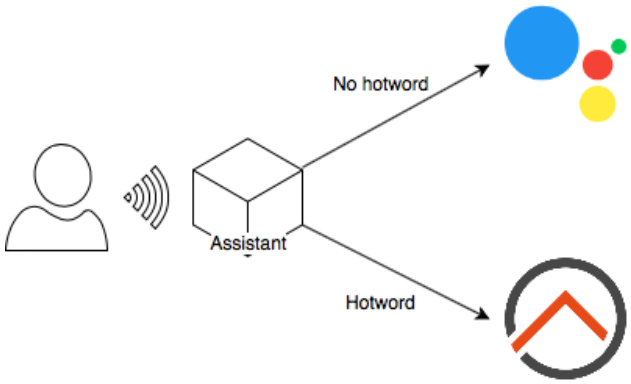

Figure 6.18: Basic diagram of the custom voice assistant

and they substitute the actions that Google Assistant can perform given the same command. So, the challenge now is to complete the following points:

- **Create a more flexible assistant**: make the assistant perform the same action from a broad range of different commands with the same meaning.
- **Make the assistant more extendible**: modularize and optimize the current script so it is easier to add more commands. Do not force the assistant to check all the possible commands after making the transcription.
- **Do not substitute any command that the Google Assistant could perform**: we want to keep the Google Assistant, but adding a layer for controlling OpenHAB. To avoid substituting the Google Assistant commands with our own custom commands, one possibility is to define another *hotword*, so that we can use it to access our custom commands.

For this purpose, we can define a simple grammar that is going to help us to process the commands and will make possible to cover all the previous points.

Grammar Definition

According to the Speech Recognition Grammar Specification v1.0 by the W3C, a token is typically an orthographic entity of the language being recognized. In the field of speech recognition, tokens can have a variety of forms (a single word, a number, a pair of words...), but they all need to go through the same process, composed by: tokenization, white space normalization, token normalization and pronunciation lookup.[59]

OK Google! Hotword	Verb	Noun	Adjective
	What to do	*What to change*	*To which state*

Table 6.4: Example sentence form for the assistant

In our script, the Assistant Library takes care of this and returns us a clean, clutter-free string with the spoken phrase. But we will consider tokens from another perspective for the next step: command processing.

First of all, I am going to define a *hotword*, this is, a word or a couple of words that can identify the whole range of available commands that we have defined to manage OpenHAB, so after the assistant identifies it, it can discard triggering Google Assistant. It could be placed at the beginning of the sentence (prefixal), in the middle of it (infixal) or at the end (suffixal). In this case, I am going to consider only the prefixal placement, so I let Google Assistant to process all the other sentences, even if they have the hotword placed elsewhere.

The hotword that I am going to use is *home*. It is easily recognizable by the assistant and cannot produce much confusion in its pronunciation.

The hotword will be followed by a sentence whose form will be more or less free. At this point, trying to identify a substring of more than two words is not very useful, so we can check for some keywords inside the sentence and discard the other options, step by step. For that, we need to know how sentences are formed. In our case, the most common form will be as indicated in table 6.4.

So, the assistant has to process the part that goes right after the *hotword*. In my case, I am going to check for actions, like *turn* or *power*. If the action is contained anywhere in the sentence, it will check for derived words. For example, it makes sense to check for *on* or *off* after these previous words. But before that, just to keep the specified order, I will check the noun. In other words, the device to which the action is directed. As the script checks only if the word is contained in the sentence, the order that it follows for checking words is not a big issue. But why does not it keep track of the order?

1. “Turn on all the lights”
2. “Turn all the lights on”
3. “Set the office light to red”
4. “Turn red the office light”

The sentences 1 and 2 and, on the other hand, the sentences 3 and 4, mean exactly the same. In an assistant that expects to receive a relatively small amount of commands, this should be enough, though in more complex assistants checking the word order matters.

The sentence check will always end with the execution of a function. If it recognizes the correct words, it will trigger an OpenHAB command or a system one, for example. If the sentence does not match what the assistant expects, it will say that the command has not been recognized, in any case. To give it a more *human* touch, it will exactly say *Sorry, I cannot do that yet.*

We can see that this way we can extend a lot the functionality of the assistant, while keeping a well-organized code. Despite it might have to do more checks than the previous version in some cases, in the long run with more commands, this will be much more efficient, because it has the ability to discard many other options and go straight to the requested command.

Although I have tried to make a user-friendly script, it is clear that any house arrangement will need a specific configuration in the script, depending on the devices and states that they can have.

Improving the Range of Actions

At this moment, the assistant is able to turn on and off all the lights and to manage the system (shutting down and rebooting).

Taking advantage of the improved command processing that I have specified above, I have implemented a few more actions. The following example has been made over a system composed by a Philips HUE color light. It is really easy to implement more lights; the user just needs to add their item IDs and a name for each light to the beginning of the file. The script will automatically recognize them, and it will be able to send commands to them. Now, the actions that it can perform are:

- For all the lights (through a *Group* item):
  - Turn on and off.
- For each light:
  - Turn on and off.
  - Change the light color (red, yellow, blue, pink, green), maintaining its brightness.
  - Change the light color temperature (cold, warm, natural).
  - Increase and decrease its brightness.

- For the system:
  - Turn off.
  - Reboot.

The last three commands specific to a single light require a new function, which makes a GET request to the REST API of OpenHAB, in order to get the current state of a light. The state of a Philips HUE light is composed by three numbers: its color (defined between 0 and 360 degrees, like a color wheel), its saturation and its brightness (between 0 and 100). On each POST request, I need to send these three values, so the script uses GET requests to send the same value that it had before in the parameters that we want to maintain. This new function is really simple, and it uses the Python *Requests* library.[45]

```
1 def openhab_get_state(item):
2   url = 'http://localhost:8080/rest/items/' + item + '/state'
3   r = requests.get(url)
4   return r.text
```

Final Result

The full script is included in the appendix A.

6.4.10 Adding Automation via IFTTT

An often desirable function of a home automation system is the possibility to add and manage rules. Rules enable users to set actions when a specific situation happens, so they do not have to interact directly with the system to perform these actions. This is called automation, and an example of it can be lowering the shutters after 8 PM or turning on the house lights when a motion detector perceives movement.

Introduction to IFTTT

There is a free platform that has become more popular over the last years called "If This Then Else", or IFTTT. This web-based service allows users to create chains of conditional statements, known as applets, and they can be really powerful. A conditional statement could be one of the conditions mentioned above.[23]

Apart from the website, IFTTT offers an application for iOS and Android, and integration with openHAB, which is the most interesting part for

us. Additionally, it is possible to connect other web services to IFTTT, such as social networks, and it is possible to use them as well to trigger actions in OpenHAB.

The main disadvantage of IFTTT and other similar solutions is that the rule must be implemented to use it, and the user may not be able to carry out the automation the way he wants. However, there are plenty of rules that can meet most requirements.

OpenHAB and IFTTT

As we can see, IFTTT perfectly complements OpenHAB, adding a functionality that, in spite of being present in other proprietary home automation solutions, has many more possibilities.

IFTTT is available to all openHAB users through myopenHAB, which is an instance of the openHAB Cloud Service hosted by the openHAB foundation. Thanks to this service, us and other apps can access and interact with our openHAB instance from the Internet.[36]

Installation and Configuration

By default, openHAB cannot connect to any openHAB Cloud instance, but there is a binding named *openHAB Cloud Connector* that we can install for this purpose. The binding is installed through the PaperUI. After this, two new files are created in the installation folder of openHAB, in my case:

```
/var/lib/openhab2/uuid
/var/lib/openhab2/openhabcloud/secret
```

These files contain the UUID and secret key of the instance of openHAB. We need to create an account on myopenHAB, indicating an email, password and these previous keys. Once the account is created, we can go back to our instance of the platform and configure the binding (under *Configuration* > *Services* > *IO* > *openHAB Cloud*).

The user must add some items to expose so IFTTT can manage them. Then, back to myopenHAB, we should be able to see the panel of our instance. Thanks to this service, it is now accessible as well from any part of the world and to external services, like IFTTT.

After this, I have created an account on IFTTT, which is also a very quick process. Once the account is created, IFTTT shows a list of the services that it is capable to handle. OpenHAB is between them, and I am taken back to myopenHAB to allow their connection.

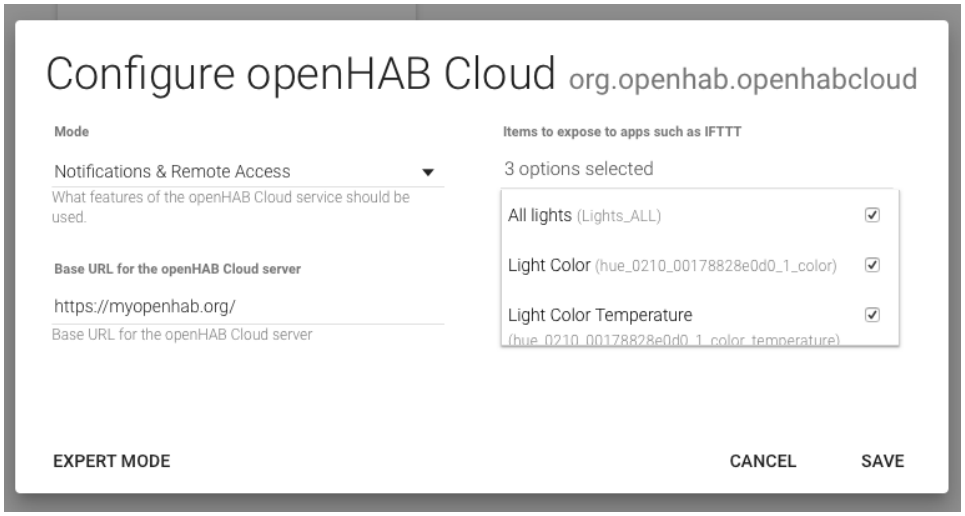

Figure 6.19: openHAB Cloud Binding configuration

Now, I am able to set IFTTT rules. I will set the first one on their web platform. The process begins with creating a new applet of openHAB.

I can only select three different triggers, as shown in figure 6.20. In this example, I am going to make lights turn blue if the humidity is over 70%, so the if condition is going to be that *item state raises above*. Then, I need to specify the item (previously made accessible from openHAB), and give the minimum value, "70" in this case. After this, I am taken back to the service selector, that offers us again a broad range of actions to accomplish if this condition happens. I choose again openHAB, the item that is going to receive the command, and the command itself, which is simply: 260, 100, 100 (which turns the light blue, at 100% of brightness and saturation).

As this situation is not likely to happen in the moment I am writing these lines, I am going to set a new rule, which combines openHAB and another service, like Twitter. Now the light is going to turn red every time I send a new tweet. I put Twitter as the *if condition* and openHAB in the *then condition*. IFTTT is fully integrated with Twitter and includes very useful shortcuts. However, with openHAB we need to write the full command, as if we were dealing with its REST API. This time, the command that is sent to the light color item is pretty similar: 0, 100, 100 (zero indicates the red color, and the other parameters are the same as in the previous command).

The applet works finely. I publish a tweet and, after some minutes, the light turns red. Although I have used this as an example, it might not be really useful in a real environment, but the end user could use a similar rule as a notification system every time he is mentioned on Twitter, or when he receives new emails. It is also possible to set very useful rules with the

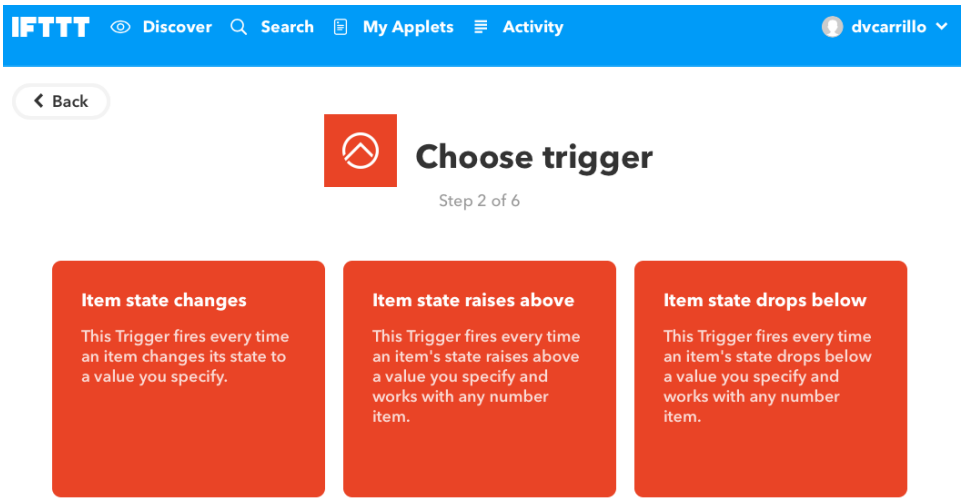

Figure 6.20: Configuration of an openHAB IFTTT applet

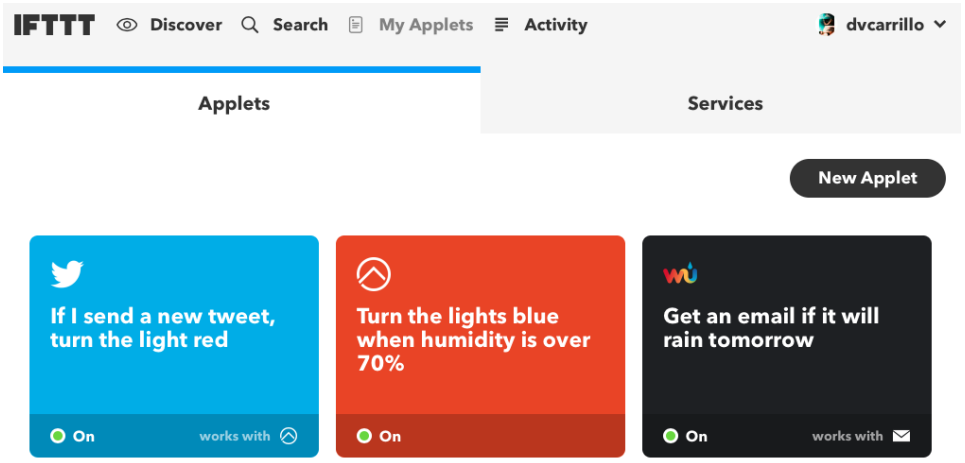

Figure 6.21: IFTTT applet panel

date and time as the *if condition*, like turning on the lights after 8 PM. Possibilities seem to be endless when combining openHAB and IFTTT.

6.4.11 Adding Global Access to the System

An openHAB instance can, by default, only be accessed from the same network as the device on which it is installed. This is also the case in some other home automation systems and smart devices.

However, a home automation system that can be accessed from any place, regardless of the network the user is connected to, can greatly extend its usability. For example, it would be possible to turn the oven on 30 minutes before arriving home and have the food cooked at the moment the user arrives. Or checking the temperature inside home from the office.

In this section I will explore the different options we have to implement this feature, from doing it manually to making use of the possibilities offered by openHAB.

Making the Runtime Accessible From the Public Internet

The first option is to make the local instance accessible from the public Internet. This is a process that can be done with any software that is hosted in a computer connected to the Internet and that listens to any port and consists in making a route between the device and its port, and the public Internet.[32]

In most homes and businesses, routers use a Network Address Translator, or a NAT, which is mainly what disallows to make devices directly reachable from Internet. The NAT takes the public IP address for itself and assigns local IP addresses for the computers and devices on the local network, so that they are all effectively sharing the same IP address. So, if we want to make a single device reachable from outside, we need to tell the router how to address a specific request. This is called port forwarding and, as the name indicates, it makes use of the ports. Ports are usually specific to each server-based application, and this is also the case in openHAB, which uses the port 8080 by default.

Most of the routers nowadays provide a software where we can make any necessary change, which is accessible via HTTP in the local address of the router. Here it is possible to set the ports that we want to forward.

For setting a port forwarding rule we usually need to specify some properties, as the protocol to use, the start and end ports or the host IP. After that, the router should be able to address any request with the specified properties to the device.

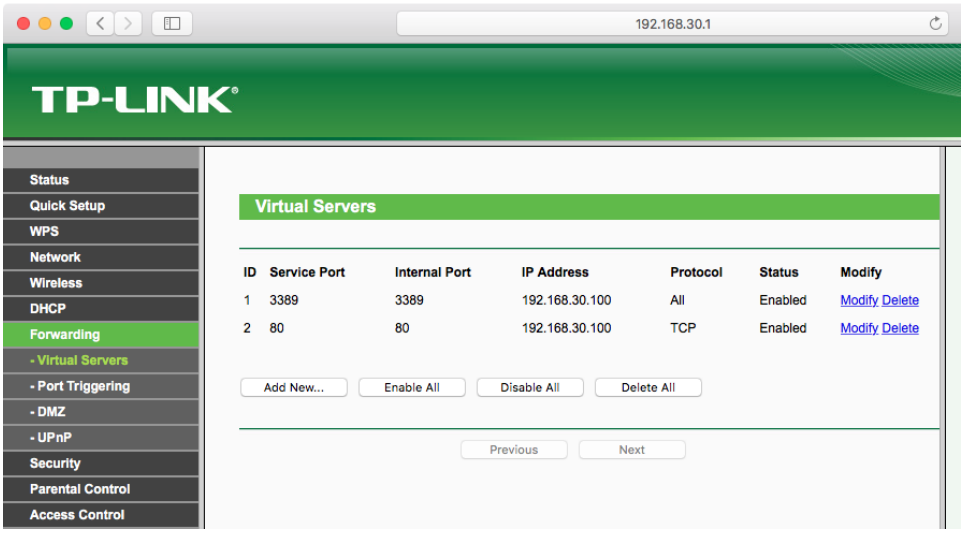

Figure 6.22: Port Forwarding section of the router used in this project

Although this is an easy and fast process, I will not go into detail too much, as this is a very unsafe solution. OpenHAB does not provide any authentication method nor any restricted access at this moment, so opening a port must be avoided, because anyone could access our system and view all data as if they were us.

Running openHAB Behind a Reverse Proxy

This solution will make our openHAB instance accessible from the Internet without depending on any third-party servers. However, we need our own domain first. This solution is the best one if the user has already purchased a domain. If not, this would require spending between EUR 10 and EUR 20 every year[52], so it might be more appropriate to use myopenHAB Cloud Service, which is free to use.

Running openHAB behind a reverse proxy allows to access the openHAB runtime via port 80 (HTTP) and 443 (HTTPS). It also provides a simple way of protecting the server with authentication and secure certificates.

The first step is to set up a HTTP server, like NGINX or Apache, which support reverse proxying. NGINX is based on configuration files, and we need to create one to allow it to proxy openHAB. The following code is the configuration to apply, as long as the openHAB instance is located in the same machine as the reverse proxy, as in my case:

```
server {
  listen                                    80;
```

```
  server_name                             openhabpi.me;

  location / {
    proxy_pass                            http://localhost:8080/;
    proxy_set_header Host                 $http_host;
    proxy_set_header X-Real-IP            $remote_addr;
    proxy_set_header X-Forwarded-For      $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto    $scheme;
  }
}
```

Then, I save the file in sites-available and *activate*it by linking it in the sites-enabled folder. After it, we only need to restart NGINX and it should be ready.

The system can be secured thanks to the authentication system that NGINX provides, which needs to be configured with an authentication user file and with the utility htpasswd, included in the *apache2-utils* package. The following command creates a file that we can use with NGINX. I am using *dvcarrillo* as the username:

```
sudo htpasswd -c /etc/nginx/.htpasswd dvcarrillo
```

Once it has been created, I have to move it to the NGINX directory (in my case, */etc/nginx/sites-enabled/openhab*), and include these lines in the configuration file, underneath the *proxy_** settings:

```
auth_basic                      "Username and Password Required";
auth_basic_user_file            /etc/nginx/.htpasswd;
```

NGINX should be restarted to apply these changes now.

Another way to secure the system without needing to enter a username and password is to restrict the access to only some specific IPs, blocking the access to everyone else. This can be done thanks to the NGINX directives *satisfy* and *deny*. For example, by adding these lines inside the location block:

```
satisfy  any;
allow    192.168.0.1/24;
allow    127.0.0.1;
deny     all;
```

NGINX will allow anyone within the range from 192.168.0.1 to 192.168.0.24 and the *localhost* to connect without a password.

To pick this method or the previous one is totally up to the user's needs and his own idea of the system. The password authentication is suitable if he wants to connect to his openHAB instance from several networks and he

does not know their IP beforehand. But if he knows exactly the networks and devices that he is going to use, the second option provides extra security.

Another option is to combine both, which would require to set up a password and then configure the IP allowance. This way, the allowed IP addresses would directly access openHAB and the rest of them would be asked to log in using the specified username and password.

The reverse proxy choice also provides, as I mentioned, the possibility to enable HTTPS, so communications between the client and the server are encrypted. This is an important step that will protect against eavesdropping and possible forgery, and I would recommend it to anyone who chooses this method.

The process to enable HTTPS in the reverse proxy begins with using OpenSSL to generate a self-signed certificate. Although if we have a valid domain and can change the DNS to point towards your IP, openHAB recommends using Let's Script.[26]

OpenSSL can generate a certificate which will be valid for a year with the following command:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/
    ssl/openhab.key -out /etc/ssl/openhab.cr
```

Then, I need to tell NGINX that there are SSL certificates and their location. This is done by adding the following to the configuration file, underneath the *server_name* variable, assuming we placed the certificate in the directory */etc/ssl/*:

```
ssl_certificate                 /etc/ssl/openhab.crt;
ssl_certificate_key             /etc/ssl/openhab.key;
```

Then, I must tell the server to listen on the HTTPS port by changing the listen parameter:

```
listen                          443 ssl;
```

After restarting NGINX, we will be using a valid HTTPS certificate. Finally, we must redirect the HTTP traffic to HTTPS. To get this done, we only need to tell the server to direct all incoming connections on the port 80 (http) to the https version of the domain.

Final Result To summarize, the final NGINX configuration file will look like the following:

```
server {
  listen                          80;
  server_name                     openhabpi.me;
  return 301                      https://$server_name$request_uri;
}
server {
  listen                          443 ssl;
  server_name                     openhabpi.me;

  ssl_certificate                 /etc/ssl/openhab.crt
  ssl_certificate_key             /etc/ssl/openhab.key

  location / {
    proxy_pass                                http://localhost:8080/;
    proxy_set_header Host                     $http_host;
    proxy_set_header X-Real-IP                $remote_addr;
    proxy_set_header X-Forwarded-For          $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto        $scheme;
    satisfy                                   any;
    allow                                     192.168.0.1/24;
    allow                                     127.0.0.1;
    deny                                      all;
    auth_basic                                "Username and Password
        Required";
    auth_basic_user_file                      /etc/nginx/.htpasswd;
  }
}
```

This is the safest method I propose, and I believe that under normal circumstances, this security will be more than enough. Although it can even be improved further, indicating specific cyphers and SSL settings. OpenHAB proposes the following snippet for this purpose:

```
ssl_protocols                   TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers       on;
ssl_dhparam                     /etc/nginx/ssl/dhparam.pem;
ssl_ciphers                     ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-
    RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
    SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:
    ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256
    -SHA:HIGH:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!CBC:!EDH:!kEDH:!PSK
    :!SRP:!kECDH;
ssl_session_timeout             1d;
ssl_session_cache               shared:SSL:10m;
keepalive_timeout               70;
```

Using myopenHAB Cloud Service

MyopenHAB is an instance of the openHAB Cloud service, which is hosted by the openHAB Foundation e.V.

This service is fully free and it is meant to allow users to quickly check out its features without having to set up and host a personal instance.

Unlike the previous solution, this requires almost zero installation or con-

figuration from the user, the set up process is easy and fast. As I mentioned in the previous IFTTT section, the openHAB Cloud Connector binding would allow our system to be reachable from anywhere in the world, thanks to myopenHAB.

Then, the first step is to install the Cloud Connector bundle on the local openHAB runtime. This bundle establishes the connection to myopenHAB.org and authenticates against it. After this, I can log in to myopenHAB website (*https://myopenhab.org*). This will give us remote web access to the web UIs of our openHAB instance and will also let us check for the state of the items, notifications and events.

On the downside, the reliability of this service depends on third parties. They explicitly specify in their website[31] that we must be aware that they cannot offer any SLAs regarding availability. However, they promise to keep it up and running to the best of their capabilities.

Integration With the Mobile Application The advantages of using myopenHAB are also related to the openHAB mobile application. By default, we are able to access the system if we are connected to the local network where the openHAB server is located. However, if we use myopenHAB and enter *https://myopenhab.org* as a remote URL and the user and password in the credentials, we will be able to manage the openHAB runtime from anywhere in the world, and from the mobile phone. This is a very positive point to consider when choosing one of these options.

Conclusion

In this case, I will choose myopenHAB Cloud Service. After testing it, I find it reliable enough for a domestic use, although in other cases where we need to assure its functionality for the longest time, it is more recommendable to run openHAB behind a reverse proxy. MyopenHAB is also free and easy to use, and its integration with the mobile application works perfectly.

However, I would never recommend using the first proposed solution (making the instance accessible from the public Internet), as it presents major security problems.

Chapter 7

Conclusions and future work

Since the beginning of this project, the main objective has been maintained: to create an affordable, functional and usable voice-driven home automation controller. At this point, I can tell that this objective has been mostly reached, although there are many things to improve in a future.

When I began working on this system, I only knew some the most popular commercial virtual assistants with a home automation system, like Google Home or Amazon Alexa, and none of them had been launched in the Spanish market. Today, almost ten months after, only Google Home is available. It was when I started this project that I realized how many open source solutions were available, like openHAB. This allowed me to work over an existing basis and to improve its capabilities, always maintaining the main objective of affordability and functionality. The end result, although much less flexible, can offer most of the features offered by the main products on the market. The custom voice assistant, that uses Google Assistant to answer all the commands not related to openHAB, makes this product very useful for all kinds of situations.

One of the main issues that I find is that this system needs to be configured by a technical person at least in its installation. The installation of openHAB itself requires some technical knowledge, but its configuration and, most importantly, the configuration of the voice assistant, requires coding. A very convenient future improvement would be to automate the generation of the voice assistant script every time that a device is added. Of course, it would also be a good improvement to automate all the process.

This project relies heavily on third party solutions, like the Google services for processing the speech or the service *myopenHAB*. However, I have tried to explore and provide alternatives in every case, that would cover any different situation.

Another important future improvement is related to privacy. We ob-

Objective	Fulfillment	Observations
1. Integrate a home automation system in a embedded system	100%	I have successfully installed and integrated openHAB in the Raspberry Pi
2. Integrate a voice assistant in the same embedded system as the home automation system	100%	The custom assistant integrates Google Assistant and a voice assistant for openHAB
3. Explore current home automation systems and voice assistants, focusing on open-source solutions	80%	I have explored some of these systems on Chapter 3, but the inclusion of open source solutions has been reduced
4. Explore automation possibilities and implement an automation service in the domotic system	70%	I have successfully implemented an automation service using IFTTT and the REST API of openHAB. Some other services might have been explored
5. Explore options for managing the system from a mobile application	90%	The system is currently manageable thanks to the Cloud Connector and the mobile application of openHAB
6. Explore options for providing global access to the system and implement one	100%	I have explored these options on Chapter 6 and I have tested the service myopenHAB
7. Explore safety and privacy concerns related to the home automation system	30%	Some safety concerns have been covered through this work, but I have not covered the main aspects related to privacy
8. Provide an adaptive and responsive user interface, usable on touch and non-touch screens	100%	I explained how to configure Basic UI and PaperUI. Both user interfaces comply with these points
9. Connect the virtual assistant to openHAB, so it can manage the devices present in the system	100%	The custom virtual assistant communicates with openHAB thanks to the REST API of openHAB and the Requests Python library
10. Test domotic devices in the final system and present an usable solution	70%	The current solution is completely functional and usable. However, more devices should have been tested

Table 7.1: Fulfillment of the specific objectives presented in Chapter 1

served that many devices, such as the Philips Hue Smart Bridge, are in constant communication with the cloud, when in many cases this is not necessary. Some way of blocking this communication and restricting it to the local network should be explored.

The table 7.1 presents the fulfillment level of the specific objectives that I presented in the Introduction chapter.

Bibliography

[1] AIY projects: Voice. https://aiyprojects.withgoogle.com/voice/. [Online, accessed August 23th, 2018].

[2] AIY Voice Kit SD image. https://dl.google.com/dl/aiyprojects/vision/aiyprojects-2018-01-03.img.xz. [Online, accessed August 23th, 2018].

[3] Amazon: Alexa. https://developer.amazon.com/es/alexa. [Online, accessed August 13th, 2018].

[4] Apache Karaf: the enterprise class platform. http://karaf.apache.org. [Online, accessed August 16th, 2018].

[5] Apple: ios - home. https://www.apple.com/ios/home/. [Online, accessed August 12th, 2018].

[6] Apple: iOS - Siri. https://www.apple.com/ios/siri/. [Online, accessed August 13th, 2018].

[7] Betanews: The history of home automation from the beginning. https://betanews.com/2015/08/24/the-history-of-home-automation-from-the-beginning/. [Online; accessed August 6th, 2018].

[8] Botsociety blog: Voice User Interface (VUI) – a definition. https://botsociety.io/blog/2018/04/voice-user-interface/. [Online, accessed August 10th, 2018].

[9] Cleveroad blog: 3 efficient ways to supply your app with a virtual assistant. https://www.cleveroad.com/blog/how-to-create-virtual-assistant-apps-like-siri-and-google-assistant. [Online, accessed August 26th, 2018].

[10] CMUSphinx FAQ. https://cmusphinx.github.io/wiki/faq. [Online, accessed August 26th, 2018].

[11] CMUSphinx wiki. https://cmusphinx.github.io/wiki/. [Online, accessed August 25th, 2018].

[12] Craftworkz blog: Speech recognition. https://blog.craftworkz.co/. [Online, accessed August 25th, 2018].

[13] Direct energy: Advantages of a Smart Home. https://www.directenergy.com/learning-center/modern-home/advantages-smart-home/. [Online; accessed August 7th, 2018].

[14] Eclipse SmartHome: Documentation. https://www.eclipse.org/smarthome/documentation/index.html. [Online, accessed August 16th, 2018].

[15] Embedded: Home automation system design: the basics. https://www.embedded.com/design/connectivity/4431025/Home-automation-system-design--the-basics. [Online, accessed August 8th, 2018].

[16] Globalme blog: Speech recognition technology overview. https://www.globalme.net/blog/the-present-future-of-speech-recognition. [Online, accessed August 25th, 2018].

[17] Google Cloud Speech-to-Text website. https://cloud.google.com/speech-to-text/. [Online, accessed August 25th, 2018].

[18] Google: Google Assistant. https://assistant.google.com/. [Online, accessed August 13th, 2018].

[19] Home Assistant: Documentation. https://www.home-assistant.io/docs/. [Online, accessed August 12th, 2018].

[20] IBM archives: IBM Shoebox. https://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html. [Online, accessed August 10th, 2018].

[21] IBM developerWorks Recipes. https://developer.ibm.com/recipes/tutorials/raspberry-pi-4/. [Online, accessed August 25th, 2018].

[22] IBM Watson STT website. https://www.ibm.com/watson/services/speech-to-text/. [Online, accessed August 25th, 2018].

[23] IFTTT website. https://ifttt.com. [Online, accessed August 26th, 2018].

[24] Jeedom: Documentation. https://jeedom.github.io/documentation/. [Online, accessed August 13th, 2018].

[25] Jetty: Servlet engine and HTTP server. `http://www.eclipse.org/jetty/`. [Online, accessed August 16th, 2018].

[26] Let's encrypt. `https://letsencrypt.org`. [Online, accessed August 27th, 2018].

[27] LG SmartThinQ: Discover LG smart and connected appliances. `https://www.lg.com/us/discover/smartthinq/thinq`. [Online, accessed August 11th, 2018].

[28] Microsoft cortana dev center. `https://developer.microsoft.com/en-us/cortana/`. [Online, accessed August 26th, 2018].

[29] Mozilla developer network documentation. `https://developer.mozilla.org`. [Online, accessed August 26th, 2018].

[30] Mycroft documentation. `https://mycroft.ai/documentation/`. [Online, accessed August 13th, 2018].

[31] myopenHAB. `http://www.myopenhab.org`. [Online, accessed August 27th, 2018].

[32] NCH Software Knowledge Base: Making your computer accessible from the public internet. `https://www.nch.com.au/kb/10046.html`. [Online, accessed August 26th, 2018].

[33] openHAB: Add-ons. `https://www.openhab.org/addons/`. [Online, accessed August 12th, 2018].

[34] openHAB community. `https://community.openhab.org`. [Online, accessed August 18th, 2018].

[35] openHAB: Documentation. `https://www.openhab.org/docs/`. [Online, accessed August 12th, 2018].

[36] openHAB documentation: IFTTT. `https://www.openhab.org/docs/ecosystem/ifttt/`. [Online, accessed August 26th, 2018].

[37] openHAB: GitHub profile. `https://github.com/openhab`. [Online, accessed August 18th, 2018].

[38] openHAB: PHC binding. `https://www.openhab.org/addons/bindings/phc/`. [Online, accessed August 17th, 2018].

[39] OSGi Alliance: The dynamic module system for Java. `https://www.osgi.org`. [Online, accessed August 19th, 2018].

[40] Philips Lightning: Meethue. `https://www2.meethue.com/`. [Online, accessed August 11th, 2018].

[41] Python documentation. https://docs.python.org. [Online, accessed August 26th, 2018].

[42] Raspberry Pi documentation. https://www.raspberrypi.org/documentation/. [Online, accessed August 23th, 2018].

[43] Raspberry Pi forums. https://www.raspberrypi.org/forums. [Online, accessed August 26th, 2018].

[44] Raúl Carretero: Por qué y cuando elegir un sistema domótico centralizado o distribuido. http://www.raulcarretero.com/. [Online, accessed August 9th, 2018].

[45] Requests library documentation. http://docs.python-requests.org/en/master/. [Online, accessed August 26th, 2018].

[46] Reuters: Research and markets: Global Home Automation and control market 2014-2020. https://www.reuters.com/article/research-and-markets-idUSnBw195490a+100+BSW20150119. [Offline, last checked August 7th, 2018].

[47] Samsung: Bixby. https://www.samsung.com/es/apps/bixby/. [Online, accessed August 13th, 2018].

[48] Samsung: SmartThings. https://www.samsung.com/es/apps/smartthings/. [Online, accessed August 11th, 2018].

[49] Smarthome beginner: Best SmartThings compatible devices – top 15 choices in 2018. https://www.smarthomebeginner.com/best-smartthings-compatible-devices-2018/. [Online, accessed August 11th, 2018].

[50] Somfy: Our story. https://www.somfysystems.com/about-us/our-story. [Online, accessed August 12th, 2018].

[51] SQA: Functional and non-functional requirements. https://www.sqa.org.uk/e-learning/SDM03CD/page_02.htm. [Online, accessed August 21th, 2018].

[52] Start Blogging Online: How much is a domain name? pricing & registration fees. https://startbloggingonline.com/how-much-does-a-domain-name-cost/. [Online, accessed August 27th, 2018].

[53] Statista: Digital Assistants - always at your service. https://www.statista.com/chart/5621/users-of-virtual-digital-assistants/. [Online, accessed August 11th, 2018].

[54] Statista: Installed base of home automation/smart home systems in Europe from 2012 to 2019 (in millions). https://www.statista.com/statistics/286815/smart-home-systems-installed-in-europe/. [Online; accessed August 5th, 2018].

[55] Statista: Smart Home - United States. https://www.statista.com/outlook/279/109/smart-home/united-states. [Online, accesed August 8th, 2018].

[56] The Eclipse Foundation: Equinox. http://www.eclipse.org/equinox/. [Online, accessed August 16th, 2018].

[57] UML Diagrams: UML use case diagrams. https://www.uml-diagrams.org/use-case-diagrams.html. [Online, accessed August 22th, 2018].

[58] Voicebot: Voice Assistant timeline: A short history of the voice revolution. https://voicebot.ai/2017/07/14/timeline-voice-assistants-short-history-voice-revolution/. [Online, accessed August 10th, 2018].

[59] W3C - speech recognition grammar specification version 1.0. https://www.w3.org/TR/speech-grammar/. [Online, accessed August 26th, 2018].

[60] Zulu.org. http://zulu.org. [Online, accessed August 23th, 2018].

[61] Muhammad Asadullah and Ahsan Raza. An overview of Home Automation systems. 2016.

[62] David Ascher, Anna Ravenscroft, and Alex Martelli. *Python Cookbook, 2nd Edition*. 2013.

[63] Costin Bādicā, Marius Brezovan, and Amelia Bādicā. An overview of Smart Home Environments: Architectures, technologies and applications. 2013.

[64] Mark D. Gross. Smart House and Home Automation technologies. 1998.

[65] Bruce T. Lowerre. The HARPY speech recognition system. 1976.

Appendix A

Voice Assistant Script

This appendix includes the full Python script of the voice assistant script that can communicate with Google Assistant and openHAB.

```
#!/usr/bin/env python3
# Copyright 2017 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
Run a voice recognizer that uses the Google Assistant Library and TTS,
with customized commands for interacting with OpenHAB 2 via REST

The Google Assistant Library has direct access to the audio API, so
    this
Python code doesn't need to record audio.

Hot word detection "OK, Google" and button push are supported.

The Google Assistant Library can be installed with:
env/bin/pip install google-assistant-library==0.0.2

It is available for Raspberry Pi 2/3 only; Pi Zero is not supported.

Modified from Google AIY demo scripts by David Vargas
(https://github.com/dvcarrillo)
"""

import logging
import subprocess
import threading
```

```
38 import requests
39 import sys
40
41 import aiy.assistant.auth_helpers
42 import aiy.assistant.device_helpers
43 import aiy.audio
44 import aiy.voicehat
45 from google.assistant.library import Assistant
46 from google.assistant.library.event import EventType
47
48 # ----- CONFIGURATION -----
49 # openHAB server location
50 openhab_ip = "localhost"
51 openhab_port = "8080"
52
53 # Set the group containing all the lights
54 all_lights_group = 'Lights_ALL'
55
56 # Set the Items on OpenHAB related to your light bulbs
57 # The lights_ids array will be used for identifying each light in the
58 # spoken commands
59 light_colors = ['hue_0210_00178828e0d0_1_color']
60 light_color_temps = ['hue_0210_00178828e0d0_1_color_temperature']
61 lights_ids = ['office']     # Example: "turn on the office light"
62
63 # Hotwords that triggers the OpenHAB actions
64 custom_hotword = 'home'
65
66 # Show debug information (0 = false, 1 = true)
67 debug = 1
68 # -------------------------
69
70 # The en-GB voice is clearer than the default en-US
71 aiy.i18n.set_language_code('en-GB')
72
73 logging.basicConfig(
74   level=logging.INFO,
75   format="[%(asctime)s] %(levelname)s:%(name)s:%(message)s"
76 )
77
78 # -- OpenHAB 2 commands using OpenHAB REST API --
79
80 def openhab_send(item, state):
81   url = 'http://' + openhab_ip + ':' + openhab_port + '/rest/items/' +
        item
82   headers = { 'content-type': 'text/plain',
83               'accept': 'application/json' }
84   r = requests.post(url, headers=headers, data=state)
85
86   if (debug):
87     print('REQUEST [POST]: ' + url + ' STATE: ' + state)
88
89   if r.status_code == 200:
90     aiy.audio.say('OK')
91   elif r.status_code == 400:
92     if (debug):
93       print('ERROR [HTTP 400]: ' + r.text)
94     aiy.audio.say('There has been an error: bad command')
95   elif r.status_code == 404:
96     if (debug):
97       print('ERROR [HTTP 404]: ' + r.text)
98     aiy.audio.say('There has been an error: unknown item')
```

```
99    else:
100     aiy.audio.say('Command failed')
101
102 def openhab_get_state(item):
103   url = 'http://' + openhab_ip + ':' + openhab_port + '/rest/items/' +
          item + '/state'
104   r = requests.get(url)
105   return r.text
106
107 # -- Power management commands --
108
109 def power_off_pi():
110   aiy.audio.say('Powering off the system. Good bye!')
111   subprocess.call('sudo shutdown now', shell=True)
112
113 def reboot_pi():
114   aiy.audio.say('Rebooting the system. Hold on!')
115   subprocess.call('sudo reboot', shell=True)
116
117 def not_recognized():
118   aiy.audio.say('Sorry, I cannot do that yet')
119
120 def device_not_found():
121   aiy.audio.say('Sorry, I don\'t know that device')
122
123 def test_speech():
124   aiy.audio.say('Hello. This is a Text to Speech test.')
125
126 # -- Helper functions --
127
128 def any_idx(iterable):
129   idx = 0
130   for element in iterable:
131     if element:
132       return idx
133     else:
134       idx = idx + 1
135
136 # -- Class MyAssistant --
137
138 class MyAssistant(object):
139   """
140   An assistant that runs in the background.
141
142   The Google Assistant Library event loop blocks the running thread
          entirely.
143   To support the button trigger, we need to run the event loop in a
          separate
144   thread. Otherwise, the on_button_pressed() method will never get a
          chance to
145   be invoked.
146   """
147   def __init__(self):
148     self._task = threading.Thread(target=self._run_task)
149     self._can_start_conversation = False
150     self._assistant = None
151
152   def start(self):
153     """
154     Starts the assistant.
155
156     Starts the assistant event loop and begin processing events.
```

```
157     """
158     self._task.start()
159
160   def _run_task(self):
161     credentials = aiy.assistant.auth_helpers.get_assistant_credentials
            ()
162     model_id, device_id = aiy.assistant.device_helpers.get_ids(
            credentials)
163     with Assistant(credentials, model_id) as assistant:
164       self._assistant = assistant
165       for event in assistant.start():
166         self._process_event(event)
167
168   # State management and event processing
169   def _process_event(self, event):
170     status_ui = aiy.voicehat.get_status_ui()
171     if event.type == EventType.ON_START_FINISHED:
172       status_ui.status('ready')
173       self._can_start_conversation = True
174       # Start the voicehat button trigger.
175       aiy.voicehat.get_button().on_press(self._on_button_pressed)
176       if sys.stdout.isatty():
177         print('\nSay "OK, Google" or press the button, then speak. '
178           '\nTrigger the openHAB actions by saying \"' +
                custom_hotword +
179           '\" at the beginning of each command.'
180           '\nPress Ctrl+C to quit.\n')
181
182       elif event.type == EventType.ON_CONVERSATION_TURN_STARTED:
183         self._can_start_conversation = False
184         status_ui.status('listening')
185
186       elif event.type == EventType.ON_END_OF_UTTERANCE:
187         status_ui.status('thinking')
188
189       elif event.type == EventType.ON_RECOGNIZING_SPEECH_FINISHED and
            event.args:
190         print('You said:', event.args['text'])
191         text = event.args['text'].lower()
192
193         # CHECK HOTWORD
194         if text.startswith(custom_hotword):
195           self._assistant.stop_conversation()
196           text = text + " "
197
198           # CHECK ACTION
199           # Power, turn, set, change
200           if any(token in text for token in (' turn ', ' power ', '
                set ', ' change ')):
201             # CHECK DEVICE
202             # For all the lights
203             if ' all ' in text and ' lights ' in text:
204               if ' on ' in text:
205                 openhab_send(all_lights_group, 'ON')
206               elif ' off ' in text:
207                 openhab_send(all_lights_group, 'OFF')
208               else:
209                 not_recognized()
210
211             # For a specific light
212             elif ' light ' in text:
213               # Get the index of the mentioned item in the light
```

```
                    arrays
214                 idx = any_idx(token in text for token in lights_ids)
215                 if (idx != None):
216                   direct_to_color = light_colors[idx]
217                   direct_to_color_temp = light_color_temps[idx]
218                   current_state = openhab_get_state(direct_to_color).
                          split(',')
219
220                   if ' on ' in text:
221                     openhab_send(direct_to_color, 'ON')
222                   elif ' off ' in text:
223                     openhab_send(direct_to_color, 'OFF')
224                   elif ' red ' in text:
225                     new_state = "0,100," + current_state[2]
226                     openhab_send(direct_to_color, new_state)
227                   elif ' yellow ' in text:
228                     new_state = "100,100," + current_state[2]
229                     openhab_send(direct_to_color, new_state)
230                   elif ' blue ' in text:
231                     new_state = "260,100," + current_state[2]
232                     openhab_send(direct_to_color, new_state)
233                   elif ' pink ' in text:
234                     new_state = "340,100," + current_state[2]
235                     openhab_send(direct_to_color, new_state)
236                   elif ' green ' in text:
237                     new_state = "140,100," + current_state[2]
238                     openhab_send(direct_to_color, new_state)
239                   elif ' cool ' in text:
240                     openhab_send(direct_to_color_temp, "0")
241                   elif ' warm ' in text:
242                     openhab_send(direct_to_color_temp, "100")
243                   elif ' natural ' in text:
244                     openhab_send(direct_to_color_temp, "50")
245                   else:
246                     not_recognized()
247                 else:
248                   device_not_found()
249
250               # For the system
251               elif ' system ' in text:
252                 if ' off ' in text:
253                   power_off_pi()
254               else:
255                 not_recognized()
256
257             # Increase, raise
258             elif any(token in text for token in (' increase ', ' raise '
                    )):
259               if ' brightness ' in text and 'light' in text:
260                 idx = any_idx(token in text for token in lights_ids)
261                 if (idx != None):
262                   direct_to_color = light_colors[idx]
263                   current_state = openhab_get_state(direct_to_color).
                          split(',')
264
265                   new_brightness = int(current_state[2]) + 25
266                   if (new_brightness > 100):
267                       new_brightness = 100
268
269                   new_state = current_state[0] + ',' + current_state[1]
                          + ',' + str(new_brightness)
270                   openhab_send(direct_to_color, new_state)
```

```
271                 else:
272                   device_not_found()
273               else:
274                 not_recognized()
275
276             # Decrease, reduce
277             elif any(token in text for token in (' decrease ', ' reduce
                    ')):
278               if ' brightness ' in text and 'light' in text:
279                 idx = any_idx(token in text for token in lights_ids)
280                 if (idx != None):
281                   direct_to_color = light_colors[idx]
282                   current_state = openhab_get_state(direct_to_color).
                          split(',')
283
284                   new_brightness = int(current_state[2]) - 25
285                   if (new_brightness < 0):
286                       new_brightness = 0
287
288                   new_state = current_state[0] + ',' + current_state[1]
                          + ',' + str(new_brightness)
289                   openhab_send(direct_to_color, new_state)
290                 else:
291                   device_not_found()
292               else:
293                 not_recognized()
294
295             # Reboot, restart
296             elif any(token in text for token in (' reboot ', ' restart '
                    )):
297               if ' system ' in text:
298                 reboot_pi()
299             else:
300               not_recognized()
301
302         elif event.type == EventType.ON_CONVERSATION_TURN_FINISHED:
303           status_ui.status('ready')
304           self._can_start_conversation = True
305
306         elif event.type == EventType.ON_ASSISTANT_ERROR and event.args
                and event.args['is_fatal']:
307           sys.exit(1)
308
309     def _on_button_pressed(self):
310       # Check if we can start a conversation. 'self.
              _can_start_conversation'
311       # is False when either:
312       # 1. The assistant library is not yet ready; OR
313       # 2. The assistant library is already in a conversation.
314       if self._can_start_conversation:
315         self._assistant.start_conversation()
316
317 def main():
318   MyAssistant().start()
319
320 if __name__ == '__main__':
321   main()
```