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, 19 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 19 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 19 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

1 Introduction **1**
1.1 Incentive . . . 1
1.2 Objectives . . . 1
1.2.1 Generic . . . 1
1.2.2 Specific . . . 1
1.3 Structure of the project . . . 1

2 Home Automation **3**
2.1 What is Home Automation? . . . 3
2.2 Home Automation System Design . . . 7
2.2.1 Centralized Architecture . . . 8
2.2.2 Decentralized Architecture . . . 9
2.2.3 Distributed Architecture . . . 9
2.2.4 Hybrid Architecture . . . 10

3 Voice Assistance **13**
3.1 What is Voice Assistance? . . . 13
3.2 Services Voice Assistants Provide . . . 16

4 Product Analysis **19**
4.1 Home Automation Systems . . . 19
4.1.1 Philips Hue . . . 19
4.1.2 LG SmartThinQ . . . 20
4.1.3 Samsung SmartThings . . . 20
4.1.4 Google Home . . . 21
4.1.5 Apple HomeKit . . . 21
4.1.6 Somfy . . . 22
4.1.7 OpenHAB . . . 23
4.1.8 Home-Assistant.io . . . 23
4.1.9 Jeedom . . . 23
4.2 Home Automation Devices and Related Services . . . 26
4.2.1 Alarms . . . 26
4.2.2 Amazon Dash Button . . . 26

4.2.3 AV Receivers . . . 27
4.2.4 Digital Media Players . . . 28
4.2.5 Garage Door Control . . . 30
4.2.6 Garden Care . . . 30
4.2.7 Lighting . . . 30
4.2.8 Sensors . . . 33
4.2.9 Smart TV . . . 35
4.2.10 Temperature Control . . . 36
4.2.11 WiFi Sockets . . . 38
4.2.12 Xiaomi Mi Smart Home . . . 39
4.2.13 Other devices and services . . . 40
4.3 Voice Assistants . . . 41
4.3.1 Samsung Bixby . . . 41
4.3.2 Google Assistant . . . 42
4.3.3 Apple Siri . . . 42
4.3.4 Amazon Alexa . . . 43
4.3.5 Mycroft . . . 43

5 OpenHAB 45
5.1 Introduction . . . 45
5.2 History of openHAB . . . 46
5.3 Structure . . . 46
5.4 Concepts . . . 48
5.4.1 Things . . . 48
5.4.2 Items . . . 51
5.4.3 Thing Discovery . . . 53
5.4.4 Audio and Video . . . 53
5.5 A Developer Perspective on openHAB . . . 55
5.5.1 Development environment set up . . . 55
5.5.2 Platform structure . . . 56
5.5.3 OSGi . . . 59

6 Project Development 67

7 Conclusions and future work 69

Bibliography 73

List of Figures

2.1 Example of a smart home with security-oriented devices . . . 4
2.2 The Clapper, a sound-activated switch 5
2.3 The Smart Home Market revenue from 2016 to 2022 in the US[31] 7
2.4 Centralized Smart Home structure 8
2.5 Decentralized Smart Home architecture 10
2.6 Distributed Smart Home architecture 11

3.1 Google Home, a smart speaker integrated with the Google Assistant 14
3.2 Apple TV and the Siri Remote, which has a microphone to interact with the assistant 15
3.3 Estimated number of users of virtual assistants worldwide [29] 16

4.1 A Philips Hue dimmer switch, three color light bulbs and the Hue Smart Hub 20
4.2 Home-Assistant.io web user interface 24
4.3 The D-Link DCH-S150 motion sensor 33
4.4 The Netatmo Personal Weather Station 36

5.1 OpenHAB architecture 47
5.2 A simplification of the concepts of Thing and Item 48
5.3 Thing status transitions 51
5.4 Eclipse SmartHome audio stream scheme 54
5.5 A Human Language Interpreter transforms strings into other strings or commands 55
5.6 Eclipse SmartHome IDE with the OpenHAB repositories . . . 56
5.7 OpenHAB 2 structure 57
5.8 OSGi layer structure 60
5.9 Bundle state diagram 62
5.10 OSGi services [21] 63
5.11 Immediate component lifecycle 64
5.12 Delayed component lifecycle 65

List of Tables

4.1 Comparison between different home automation systems . . . 25
4.2 Comparison between different voice assistants 44

5.1 The statuses of Things in openHAB 2 50
5.2 Types of Items in openHAB 2 52
5.3 Bundle states description 62

Chapter 1

Introduction

Complete this after.

1.1 Incentive

What made me do this project.

1.2 Objectives

What do I want to achieve with this project.

1.2.1 Generic

Generic objectives that I want to achieve.

1.2.2 Specific

Specific objectives that I want to achieve

1.3 Structure of the project

Indicate how I have structured the project

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 ïntelligentḧome can provide a safer, more comfortable, and more economical dwelling.[35] 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.[30]

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 [5]. 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)[7] 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%.[7]
- **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

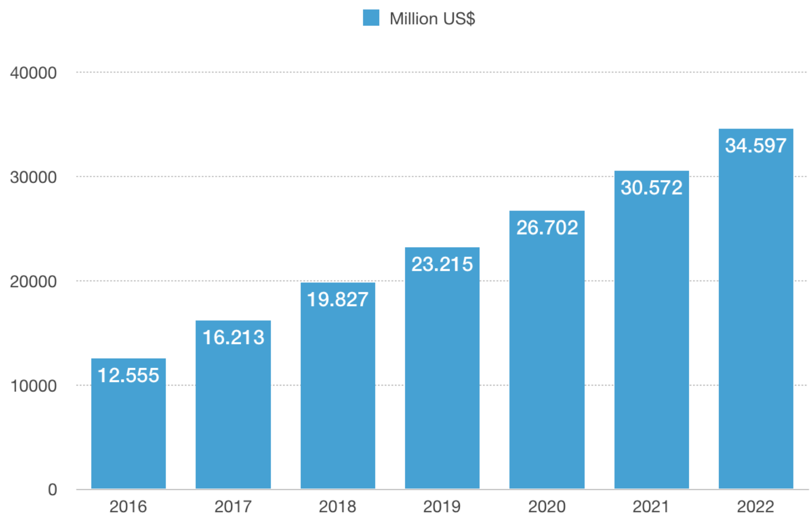

Figure 2.3: The Smart Home Market revenue from 2016 to 2022 in the US[31]

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.[24] 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

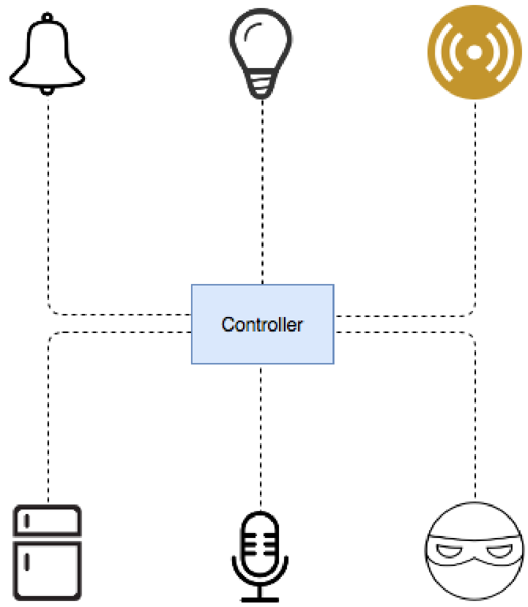

Figure 2.4: Centralized Smart Home structure

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[9]. 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.[34] 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.[23] 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*.[34]

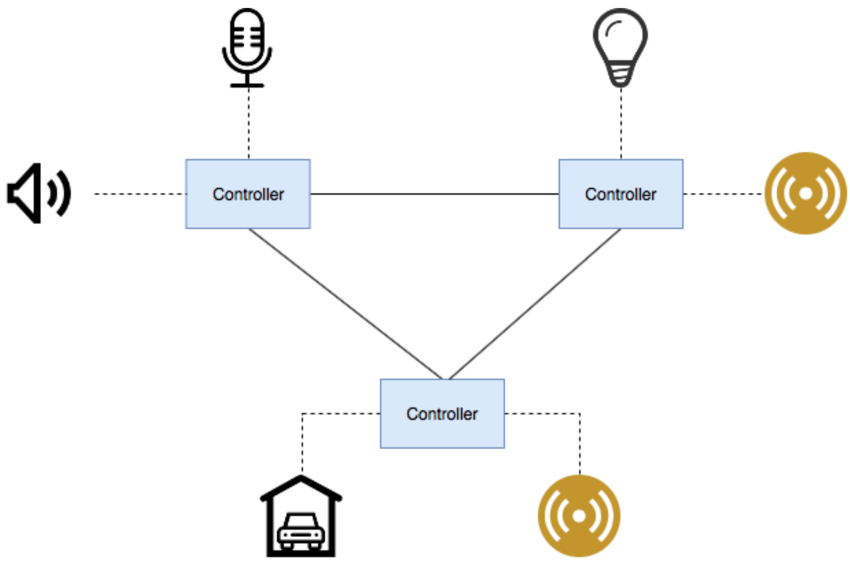

Figure 2.5: Decentralized Smart Home architecture

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[23]. 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

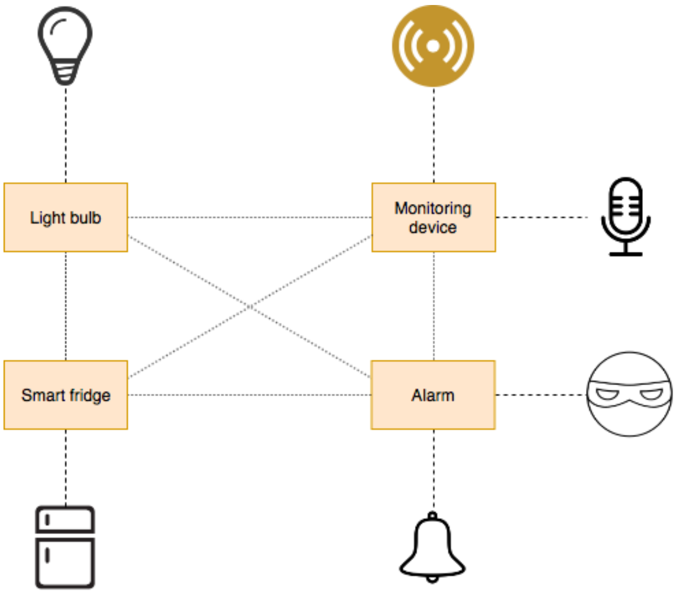

Figure 2.6: Distributed Smart Home architecture

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.[6] 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.[33] 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.[12]

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.[36] 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

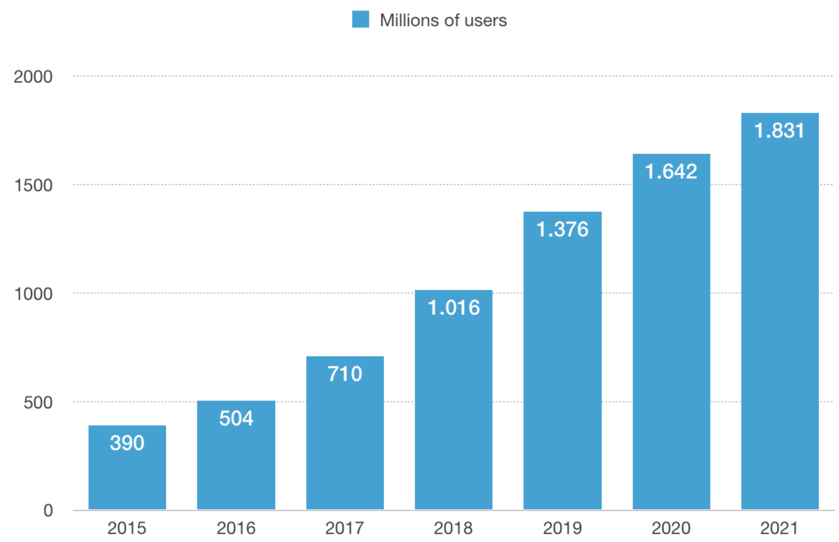

Figure 3.3: Estimated number of users of virtual assistants worldwide [29]

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*.[33]

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.[29]

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.[22]

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.[15]

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.[26]

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.[27] 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.[3] 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).*[28]

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. [18] 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.[17]

4.1.8 Home-Assistant.io

Home-Assistant.io, or simply Home Assistant, is a open source home automation platform running on Python 3.[11] 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.[13] It is aimed for individuals and professionals, and provides custom support for both. They also sell what they call

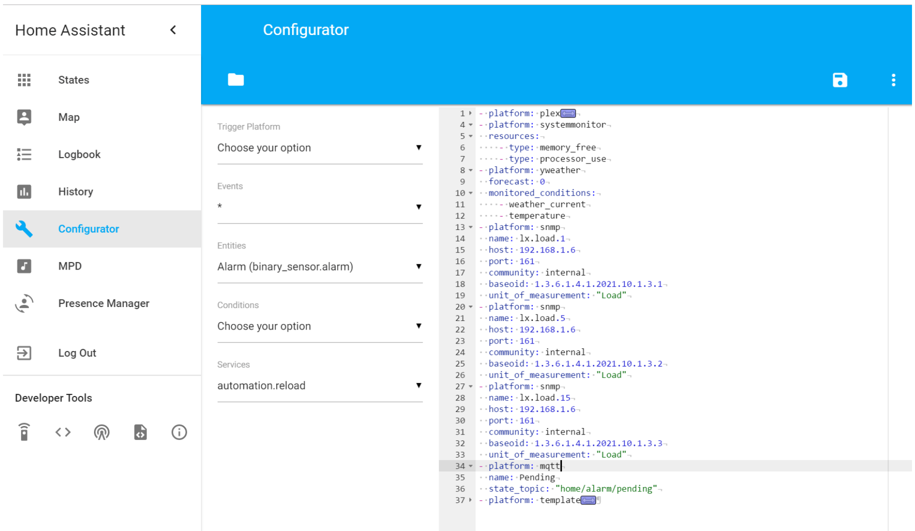

Figure 4.2: Home-Assistant.io web user interface

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[18], 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:

- Philips 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.[25]

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.[10]

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.[4] 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.[1]

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.[16] 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

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

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.

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.[18]

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.

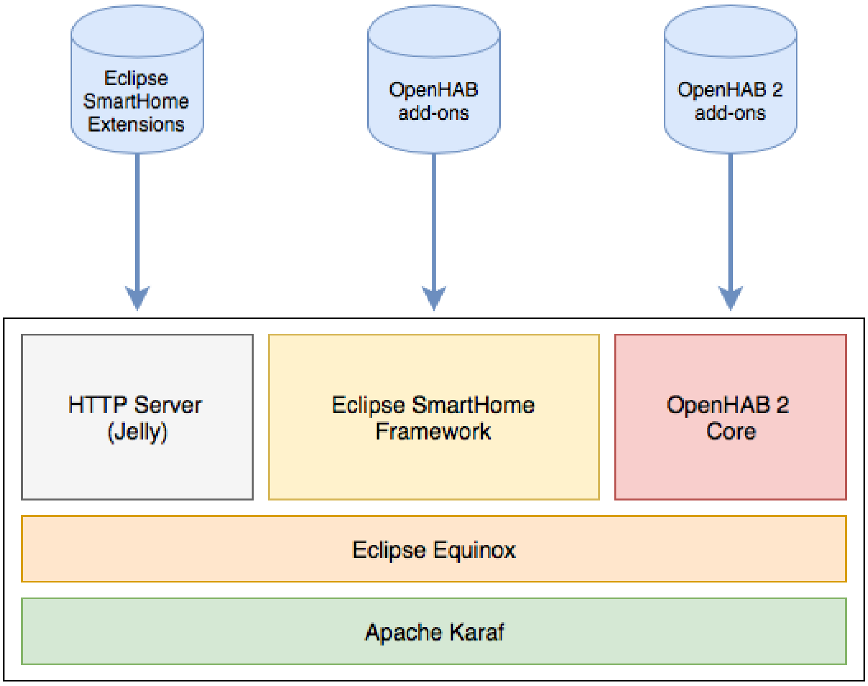

Figure 5.1: OpenHAB architecture

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.[2]

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.[32]

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.[14] Secondly, we can find the Eclipse SmartHome Framework , the framework to build end user solutions on top like openHAB, that I mentioned before.[8] 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.

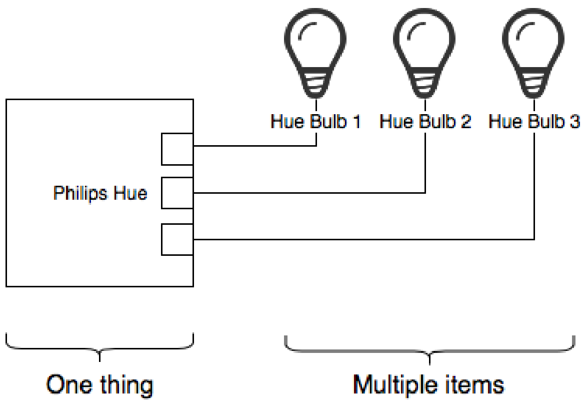

Figure 5.2: A simplification of the concepts of Thing and Item

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.[18]

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.[20]

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: The statuses of Things in openHAB 2

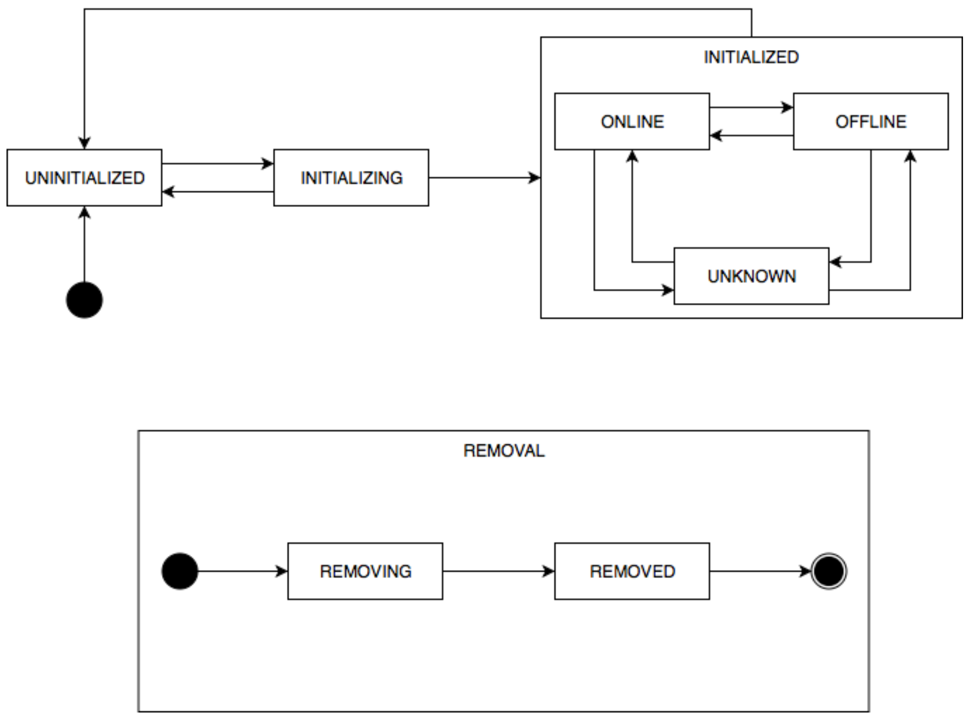

Figure 5.3: Thing status transitions

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

Color	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*.

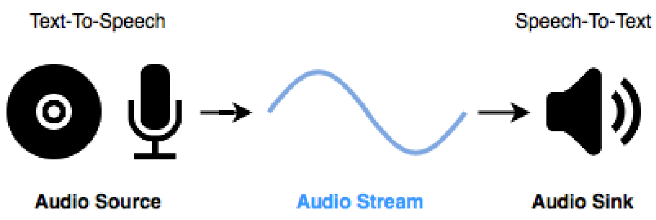

Figure 5.4: Eclipse SmartHome audio stream scheme

- **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.

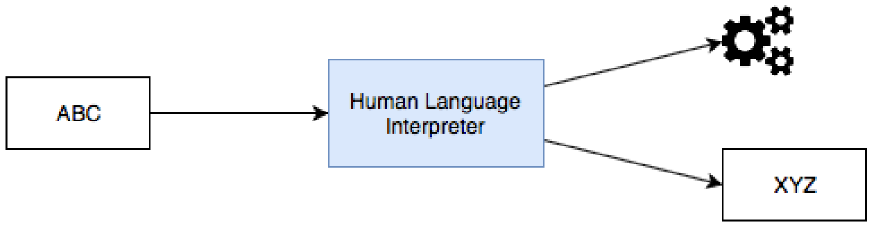

Figure 5.5: A Human Language Interpreter transforms strings into other strings or commands

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.

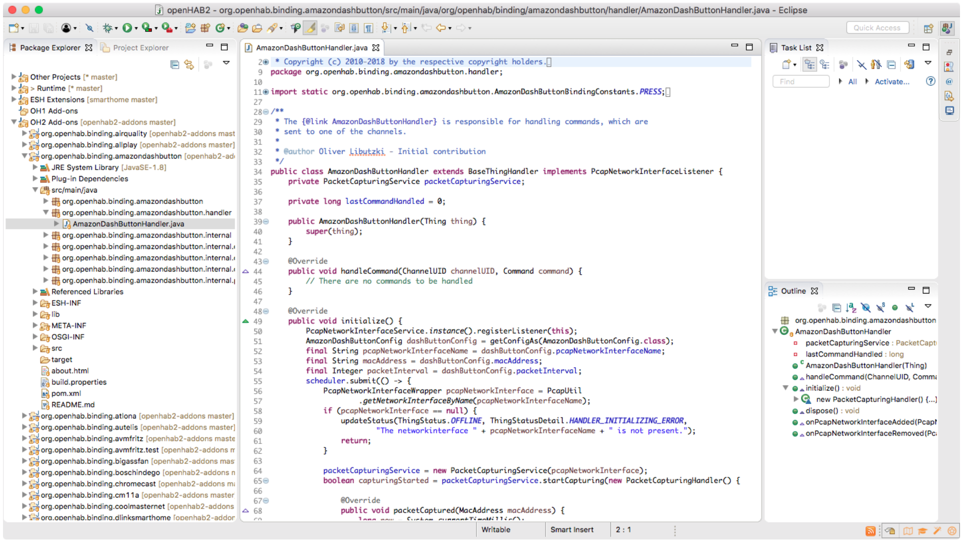

Figure 5.6: Eclipse SmartHome IDE with the OpenHAB repositories

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. [19]

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

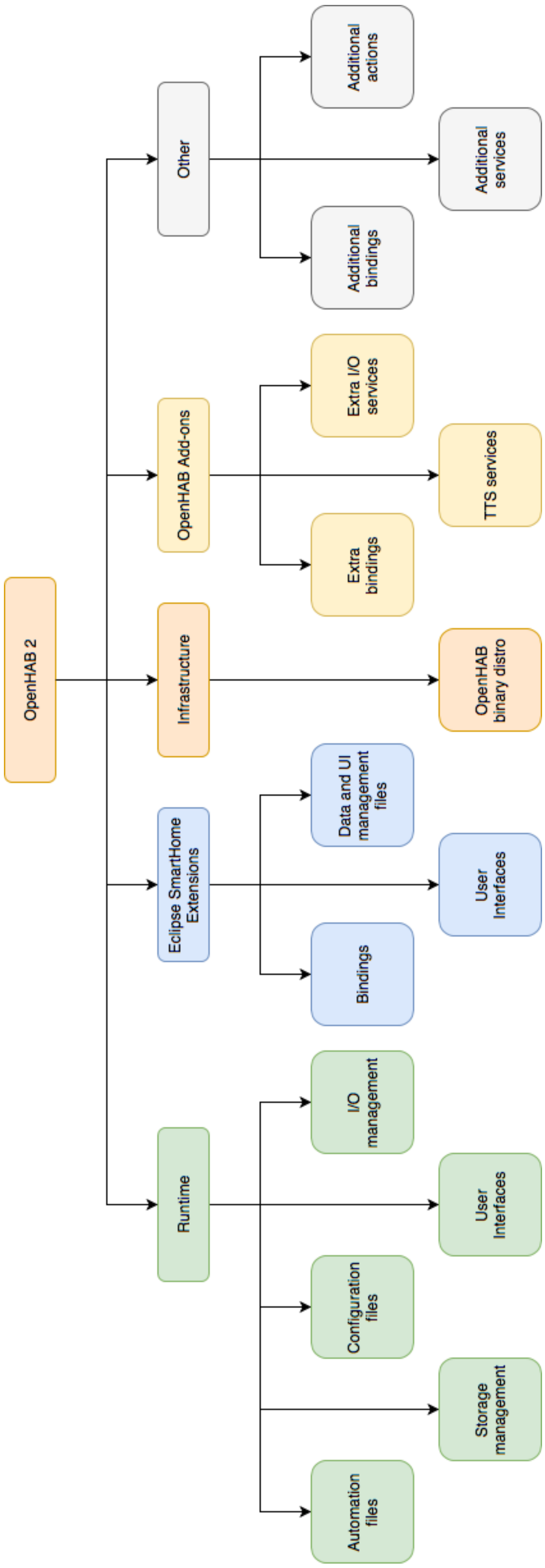

Figure 5.7: OpenHAB 2 structure

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.[18] 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.

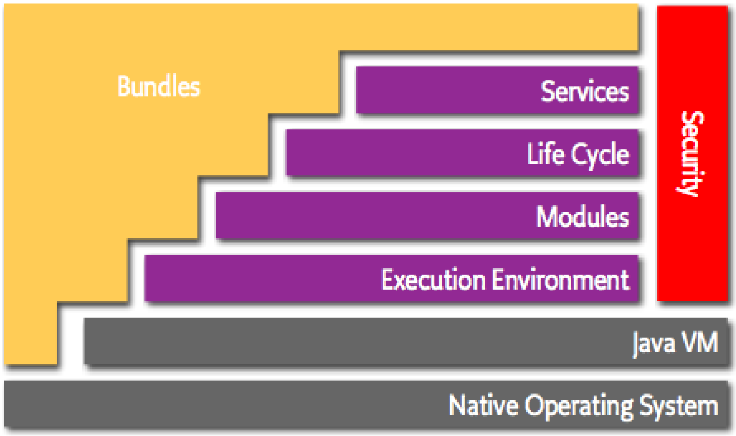

Figure 5.8: OSGi layer structure

- **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

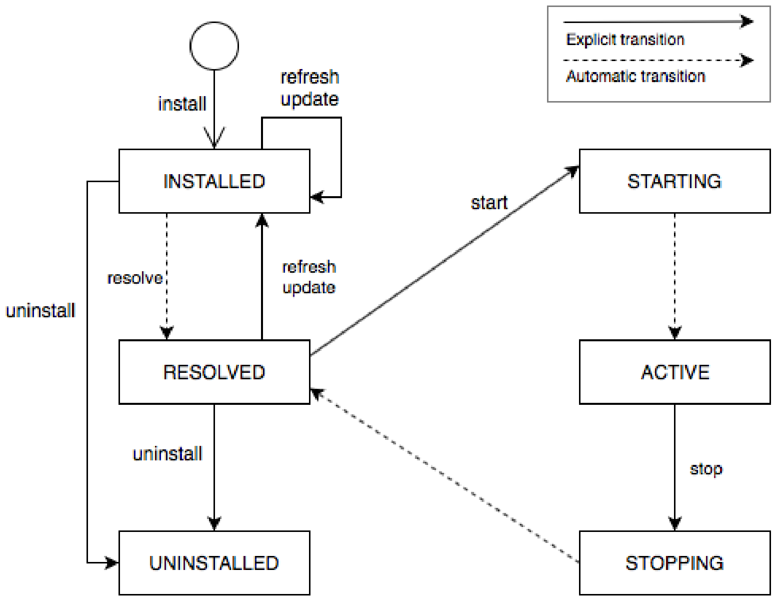

Figure 5.9: Bundle state diagram

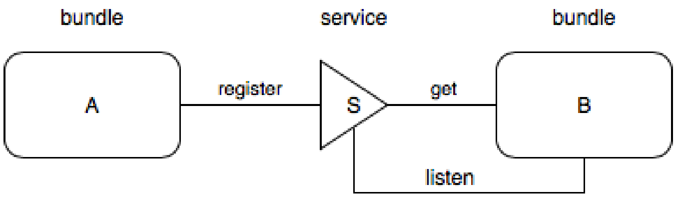

Figure 5.10: OSGi services [21]

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

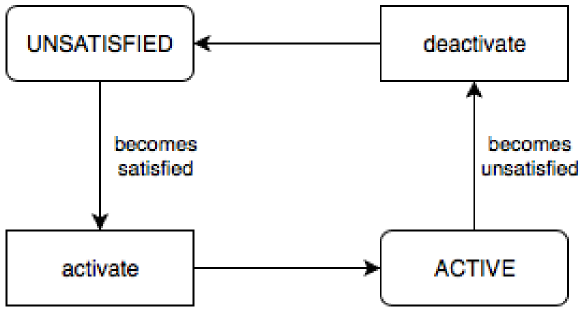

Figure 5.11: Immediate component lifecycle

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:

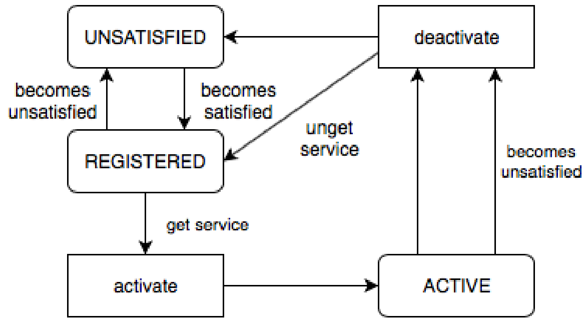

Figure 5.12: Delayed component lifecycle

- 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

Project development

Chapter 7

Conclusions and future work

Conclusions and future work.

Bibliography

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

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

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

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

[5] 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].

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

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

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

[9] 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].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[24] 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].

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

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

[27] 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].

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

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

[30] 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].

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

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

[33] 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].

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

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

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