UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A COGNITIVE COMPUTING EMPOWERED ROBOT ASSISTANT FOR SEMANTIC TASK AUTOMATION IN SMART PLACES

ENRIQUE SÁNCHEZ TOLBAÑOS

2017

TRABAJO FIN DE GRADO

Título:	Diseño y desarrollo de un robot asistente basado en Computación Cognitiva para la Automatización de Tareas a través de la Semántica en un Espacio Inteligente
Título (inglés):	Design and Development of a Cognitive Computing empowered Robot Assistant for Semantic Task Automation in Smart Spaces
Autor:	Enrique Sánchez Tolbaños
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A COGNITIVE COMPUTING EMPOWERED ROBOT ASSISTANT FOR SEMANTIC TASK AUTOMATION IN SMART PLACES

Enrique Sánchez Tolbaños

Junio de 2017

Resumen

Este trabajo es el resultado de un proyecto cuyo objetivo ha sido desarrollar un robot asistente utilizando computación cognitiva para la automatización de tareas en entornos inteligentes a través de reglas semánticas.

Para ello, se ha desarrollado un agente de conversación para realizar las conversaciones necesarias para crear las automatizaciones y descubrir el espacio de trabajo inteligente. Este sistema de conversación ha sido desarrollado utilizando dos alternativas, Api.ai e IBM Watson.

Se ha desarrollado una biblioteca de sonidos y movimientos para las interacciones del robot asistente. Para ello, hemos utilizado el protocolo Bluetooth creado por WooWee para controlar su robot. De esta manera, hemos hecho una interfaz visual para las conversaciones realizadas, ayudándonos a comunicarnos mejor con los usuarios.

A continuación, se ha utilizado Ewetasker para crear las reglas necesarias que hacen posibles las automatizaciones. Estas reglas también permitirán al usuario crear recordatorios y permitirán a la empresa crear interacciones definidas para momentos concretos de un día de trabajo. La tecnología semántica será el motor de esta parte del proyecto.

Finalmente, se han llevado a cabo los desarrollos necesarios para controlar los dispositivos presentes en nuestro espacio inteligente. Por este motivo, se ha desarrollado un servidor proxy que canaliza todas las acciones lanzadas por Ewetasker y las envía al dispositivo adecuado.

Como resultado, este proyecto permitirá a los usuarios conocer y controlar un entorno inteligente a través de la voz y hará posible que objetos no inteligentes formen parte de este nuevo contexto añadiendo más funcionalidad al sistema.

Palabras clave: Computación Cognitiva, Robot, Automatización de tareas, Entorno inteligente, Ewetasker, Reglas Semánticas, Agente de Conversación

Abstract

This work is the result of a project whose objective has been to develop a robot assistant using cognitive computing for task automation in smart environments through semantic rules.

For this purpose, a conversation agent has been developed in order to perform the conversations needed to create the automations and to discover the smart workspace. This conversation system has been developed using two alternatives, Api.ai and IBM Watson.

A sounds and movements library has been developed for the robot assistant interactions. To do so, we have used the Bluetooth protocol created by WooWee to control its robot. This way, we have made a visual interface for the conversations performed, helping us to communicate better with users.

To continue, Ewetasker has been used to create the rules needed that make automations possible. These rules will also allow the user to create reminders and they will allow the company create interactions defined for concrete moments of a working day. Semantic technology will be the engine of this part of the project.

Finally, we have carried out the necessary developments to control the devices present in our intelligent space. For this reason, a proxy server has been developed that channels all actions triggered by Ewetasker and sends them to the appropriate device.

As a result, this project will allow users to know and control an intelligent environment through the voice and will make it possible for non-intelligent objects to be part of this new context by adding more functionality to the system.

Keywords: Cognitive Computing, Robot, Task automation, Smart Space, Ewetasker, Semantic Rules, Conversation Agent

Agradecimientos

Gracias a mis padres por enseñarme a ser la persona que soy.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Nowadays, cognitive computing [12] inspires the research and the development efforts of numerous companies and universities. We can find examples of its application in very different fields, like trading analysis, biomedical image recognition, or self-driving cars. The purpose of all these technologies, is to go further than the capabilities of a human, at the same time that the interactions between a machine are simplified.

In the same way as cognitive computing makes progress, it increases the amount of smart devices in our homes and offices capable of implementing this technology. All these devices start from the premise of make easier life to the people. However, we found the problem of configuring each gadget, with each own application, and in each smart environment we visit.

Closely related to this, the development of autonomous service robots has received considerable attention in the last years [14, 7]. These robots should be capable to engage in meaningful, coherent, task–oriented multi modal dialogues with humans to carry out daily–life activities. To manage these tasks, they are equipped with a specific hardware, which is designed to make actions and to receive stimulus from the outside. The design

is based on considering conceptual frameworks for the description of the tasks in which domain knowledge, interaction structures, and robot capabilities are integrated in a simple, comprehensible, and coordinated form.

Other field necessary for the development of the robots mentioned, is the semantic technology. It consists on cataloguing, processing and analysing the information gave by the user, in order to bring closer the exactly meaning to the machine that is listening. As a result of the advantages of this technology, we can make interactions with the robot similarly than a human interaction.

In this project, we try to offer a solution to the problem of configuring smart devices in an intelligent environment, through a robot assistant empowered with cognitive computer technology. For the development of this goal, we will support the use of semantic technologies for task automation.

1.2 Project goals

Above all things, this project aims to create smart assistant, personified on a robot, for the automation of task in an intelligent context. With this objective, the project will be formed by a conversation module, an automation task module, a robot control module and a context detection module. Through the integration of each module we will be able to control the interactions happened on a smart office environment.

Among the main goals inside this project, we can find:

- Design a dialogue including the possible use scenarios on conversation between the assistant robot and the user.
- Make rules for handle task automations of the smart devices.
- Generate interactions with the intelligent environment from the monitoring of user activity.
- Build a library of movements and sounds for the robot responses.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is the following:

Chapter 1 explains the context in which this project is developed. Moreover, it describes the main goals to achieve in this project.

Chapter 2 provides a description of the main technologies on which this project relies.

Chapter 3 makes a requirement analysis which will enable a more complete vision of the system, listing the use cases of the system.

Chapter 4 describes the architecture of this project, including the design phase and implementation details.

Chapter 5 provides an evaluation of use cases of the project in a smart office scenario.

Chapter 6 gathers the conclusions obtained from this project and provides suggestions for possible future work to improve it.

CHAPTER 2

Enabling Technologies

2.1 Introduction

In the chapter about enabling technologies, we are going to explain and analyse the techniques used in this project. In the first place, the technology used for all these applications, the cognitive computing. We will analyse the characteristics of this technology and its possible use environments. Secondly, we are going to explain how task automation works. Thirdly, the Ewetasker [15] technology that has made possible to interact with the smart environment. In the fourth place, we must talk and compare the two conversation motors used in the project. On the one hand, the Api.ai [1] conversation tool in collaboration with Google Actions [9], and on the other hand, the Watson's Conversation Service and its integration with other IBM Watson Services [11]. In the following point we must mention the device used to carry out the Api.ai conversations, the Google Home [10]. Next, we will take a look to the beacons developed by Estimote [5], devices with lots of applications that empower the interactions in a smart place. Finally, we are going to explain the characteristics of the WooWee MiP robot [19], which is in charge of the visual interaction.

2.2 Cognitive Computing

The last decade has been marked for the upswing of the artificial intelligence in front of the traditional programmable systems. Important companies on this field like IBM[1] even talk about a jump to the cognitive era. If we look for a definition of the term "cognitive computing", we can found many possible definitions, but there is only one which has been agreed by experts of all technological companies and scientific investigators. According to the Cognitive Computing Consortium[2], the definition of cognitive computing is:

"Cognitive computing makes a new class of problems computable. It addresses complex situations that are characterized by ambiguity and uncertainty; in other words it handles human kinds of problems. To respond to the fluid nature of users' understanding of their problems, the cognitive computing system offers a synthesis not just of information sources but of influences, contexts, and insights. To do this, systems often need to weigh conflicting evidence and suggest an answer that is "best" rather than "right"".

The basic model question and answer, is outdated as compared with cognitive computing models. To use this model, before giving a response, these parameters have had to be taken into account: the machine learning, the question analysis, the natural language processing, the feature engineering and the ontology analysis. In other words, this technology makes context part of the question input, which is traduced on a specific response to the problem asked.

Cognitive computing systems are thought to be a bridge between people and smart environments. For this reason they are used on virtual assistants or task automation systems. Notwithstanding, despite its principal paper at present, it is thought that the boundaries of the processes and domains these systems will affect are still elastic and emergent.

2.2.1 Cognitive computing Properties

Cognitive systems must comply these properties to accomplish their goals as high level computer systems:

- ***Adaptable.*** They have to been trained to learn with the interactions and to learn to work while the information changes and the objectives and requisites evolve. They must handle ambiguous and unpredictable data, at the same time that they must be able to consult dynamic data on real time.

[1] http://research.ibm.com/cognitive-computing/

[2] https://cognitivecomputingconsortium.com/definition-of-cognitive-computing/

- ***Interactive.*** They must be easy for the users to access them, and consequently users can show their needs on a comfortable way. They must be connected with the devices present in the environment on so easy way like people interact with them.
- ***Iterative and stateful.*** They must be prepared to anticipate the user intent by trying to redefine a problem by asking questions when a user question is incomplete or ambiguous. They must be making a history of past interactions, to be able to use this information to resolve user present questions.
- ***Contextual.*** The must be capable of extracting information about the context in which the interaction occurs. This information can be, for example, from personal data to weather data. Likewise, they must be able to access and use both structured and unstructured digital information as well as the sensory inputs provided by their sensors in case of having them.

2.3 Task Automation

At the same time than cognitive computing goes forward, more smart devices appear implementing this technology for doing human tasks. These devices, despite being created for making human life easier, need attention and human interactions to work. As a result of this, task automation comes into play.

Task automation [4] is the use of smart systems and cognitive technologies, for implementing task or process and controlling their performance, with the minimum human intervention. Normally a task automation follows the next points:

- A definition of what task are doing, and how they relate with other tasks.
- A schedule of when and how much time is going on for doing the task.
- An implementation of what resources, equipment, and tools are needed to perform the task.
- A tracking system to control and monitor the development of the task.

2.4 Ewetasker

Ewetasker [15] is a smart automation application (developed in the Intelligent System Group), which works with ECA (Event-Condition-Action) rules. The main purpose of this app, is to control the contextual events and Internet events caused by user's interactions, and to automate the responses produced by the rule inference engine. The modules used to carry out these functions are the Task Automation Server and the Mobile App [6].

Firstly, the Task Automation Server is in charge of the management functions of rules and channels . It includes the rule engine, where rules and channel events are evaluated in order to make an action, and an action trigger to interact directly with the smart devices.

Secondly, the Mobile App(created by Antonio Fernández Llamas from Intelligent System Group), which receives events through Bluetooth from beacons or other devices, and through the Internet directly from devices like smart lights. Events are sent to the Task Automation Server for be evaluated. The Mobile App is able to make actions ordered by the Task Server, because of being the simplest way of connection between some devices.

Finally, Ewetasker's election is empowered by the accessibility of its functions. The management of rules and channels can be accomplished from the web interface or making API calls, so it is easily integrated in our project, and give us plenty functionalities to automate smart devices tasks.

2.5 Conversation Tools

The number of tools for developing conversations has been increased along few years ago, at the same time as cognitive computing and low cost computers have brought closer the software development to many people. These natural language APIs provide us a good and under-exploited platform to make smart bots with AI skills and conversational human-computer interactions.

However, despite the fact that these tools are very powerful, the logic under them, has to be implemented for doing more than a simple conversation without any context or action. Normally, all conversation tools have in common parameters or fields to be configured, such as entities or intents.

Once the tool has been chosen, we have to create a Conversational Agent, and design the conversation nodes, with the intents to pass across them and with the entities to be required as a parameter in each intent. Every interaction is based on the context, and on

the last user input. Thus, agents can request information (if an entity is required), ask for confirmation (if there is a node bifurcation) or perform some type of action depending on program developed under the conversation interface. At the end, agents have to been trained for response correctly the users inputs. The training can be carry out by the developer, by adding examples of user inputs, or by the agent while the conversation is happening.

In this project we are going to use two of these conversational tolls: API.AI and IBM Watson Conversation.

2.5.1 Api.ai

Api.ai [1] is a conversation tool to create conversation interfaces for our applications. It is based on the use of user queries, entities and intents to build the interactions, like usual conversation tools, but it also offers new useful concepts like actions, events and parameters. The actions can be programmed to be read by our main application and act consequently. The events can be triggered by our main application in order to start a conversation without the user interaction. Finally, the parameters can be required for the conversation agent to make possible the actions programmed in our app. To make up for the lack of conversation nodes, it gives us the context concept, which helps us to define sequences of interactions. Another Api.ai feature is the ability of changing conversation responses by programming an external module like a Webhook [8], this ability is known as *Fullfilment*. However, the main feature of this conversation platform is the ability to integrate with other applications like Google Actions, Line, Telegram or Facebook.

If we try to make a conversation agent through Api.ai platform, we will realize the power of this tool. Api.ai platform is designed for developing agents using only a web interface. Firstly we have to define the purpose of our agent to name it and start thinking on the intents to develop. Before creating the intents, it is better to create entities first, because the interface to create intents can recognize the intents created saving much effort. Once we have intents and entities, we go on programming the context of each intent. This help us to connect intents in order to create a dialogue sequence. We can try how our conversation agent works by using the testing interface provided by Api.ai. Finally it is time to add actions, fullfillments or integrations with other apps, if we want to make our conversation

agent more powerful. Besides, we can program events to make some dialogues start without human interaction.

To put our application into operation, Api.ai matches the user query to the most suitable intent, considering information contained in the intent like examples defined, entities, contexts, or parameters required. This search will by empowered by machine learning models of the tool. The query response is a JSON object which contains all information about the intent including the answer speech. The conversation tool provides us SDKs to make text, events, or entities queries in the following programming languages: Android, iOS, Cordova, HTML, JavaScript, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP, and Java. The languages supported are Brazilian Portuguese, Chinese (Cantonese), Chinese (Simplified), Chinese (Traditional), English, Dutch, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Ukrainian.

2.5.2 IBM Watson Conversation

IBM Watson Conversation[3] is a conversation tool, characterized,like other similar applications, by the use of dialogues, entities and intents to build the interactions. It is built on a neural network (one billion Wikipedia words), that permits it to understand these parameters. The service can be developed with the following SDKs: Node SDK, Java SDK, Python SDK, iOS SDK and Unity SDK. Furthermore, the Watson's tool is available on these languages: Brazilian Portuguese, English, French, Italian, Spanish, German, Traditional Chinese, Simplified Chinese, and Dutch.

Let's see how a conversation agent can be made with IBM tool. The dialogue nodes are created by using a web interface, at the same time we can create the entities and the intents examples. This visual design of dialogue nodes is very effective to understand how conversation will work but, despite it is possible to make changes in our workspace using the SDKs provided, it is difficult to build the agent conversation by that way. We may need to write thousands of lines of JSON to build the workspace of a basic app conversation. On the other hand most SDKs offered by IBM have not implemented yet all the methods to manage the conversation agent, thus it makes difficult the developer tasks.

[3]https://www.ibm.com/watson/developercloud/conversation.html

Although, IBM Watson Conversation Service is the main service we are going to use in the Watson's demo, to make possible to use the Conversation Agent with a voice interface, we are going to integrate this application together with other IBM Watson Services like Speech to Text and Text to Speech.

2.5.2.1 IBM Watson Text to Speech

IBM Watson Text to Speech[4] converts written text into natural sounding audio employing its speech-synthesis capabilities. The service includes a long list of languages, including English, French, German, Italian, Japanese, Spanish, and Brazilian Portuguese. In order to make the speech more natural and more adaptive to users needs, the system allows developers to select among female and male voices, or even a specific dialect. Using the US English voice, we are allowed to indicate a speaking style of "goodnews", "apology", or "uncertainty", or to control voice aspects such as pitch, rate, and timbre. In addition we are facing a very powerful tool, with which we can specify how it must pronounce unusual words present in its input.

Taking a look at the technical features, the service is available through a HTTP REST API and a WebSocket API. In addition, there is an SDK supporting the same languages than the conversation service previously mentioned. These web interfaces enable the use of SSML for all supported languages. SSML is a specification for voice browsers developed by the W3C. *"It is designed to provide a rich, XML-based mark up language for assisting the generation of synthetic speech in Web and other applications. The essential role of the mark up language is to give authors of synthesizeble content a standard way to control aspects of speech output such as pronunciation, volume, pitch, rate, etc. across different synthesis-capable platforms."*[5]

Other used by the Text to Speech Service are the International Phonetic Alphabet (IPA) or IBM Symbolic Phonetic Representation (SPR). They provide us the language to define unusual pronunciations for our application. The integration of these phonetic representations on a SSML, is carry out by phoneme tag.

Use example of phoneme tag:

```
<phoneme alphabet="ibm" ph=".1Tru">through</phoneme>
<phoneme alphabet="ibm" ph=".1Sa.0kIG">shocking</phoneme>
```

[4]https://www.ibm.com/watson/developercloud/text-to-speech.html
[5]https://www.w3.org/TR/speech-synthesis/

Lastly, the audio formats available to the output are: Ogg format, Waveform Audio File Format (WAV), Free Lossless Audio Codec (FLAC), Web Media (WebM) format, Linear 16-bit Pulse-Code Modulation (PCM), mu-law (u-law), or basic audio.

2.5.2.2 IBM Watson Speech to Text

IBM Watson Speech to Text[6] allows us to convert audio voice input into written text. It is programmed to recognize the information provided by the human voice by using its intelligence capabilities to combine data about grammar and language structure with knowledge of the composition of the audio signal. The service improves its already good results, by training itself while listening more voice inputs. Despite of its goods results, the response will be a bit delayed if the audio voice input is quite long. The languages supported are Brazilian Portuguese, French, Japanese, Mandarin Chinese, Modern Standard Arabic, Spanish, UK English, and US English.

Moreover of these features, this service also includes the basic functions for developers. The last point to consider is the audio input formats, which are the same as Text to Speech Service output with the exception that the input can not be higher than 100 MB.

2.6 Google Home

Google Home [10] is a smart speaker designed by Google to be a personal assistant for our homes. Its is controlled by voice, using the conversation motor Google Assistant [9]. This personal assistant is prepared to control all tasks that can be handled with our smart devices and, of course, it is totally compatible with other Google smart devices using the same Google account. In order to bring a conversation personalized with the user, it takes data from our account, for example to remember a date in our calendar. Furthermore, it is a clear example of the power of cognitive computing, since for the development of its answers, it uses data from the environment where it is, from internet sources, and from the history of conversations already made with the user.

In this project, we will use Google Home, for being a perfect platform to the conversations built in Api.ai application, because of the good integration developed between them.

[6] https://www.ibm.com/watson/developercloud/speech-to-text.html

Figure 2.1: Google Home Device

2.7 Beacons

Beacons[2] are BLE (Bluetooth-Low-Energy) powered devices that have been designed to emit data packets every little time. These data are sent each time on a regular interval of time that can be configured until ten seconds. Consequently, their battery life expands considerably, making them a perfect device to empower interactions in smart environments.

To carry out the interactions is necessary to use other BLE devices like our smartphone.The beacon's signal strength will be detected by our smartphone to estimate the distance between them. For the development of our project, we have chosen Estimote Beacons [5] because of the great results that the Intelligent Systems Group has been getting over the past few years. Their features are:

- Available for Android and iOS, operating systems present in most smartphones.
- Broadcast interval can be changed depending on your needs.
- Attractive design, making it perfect for any environment.
- Battery life can reach two years.

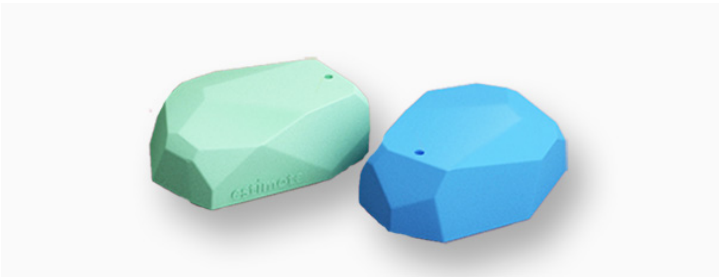

Figure 2.2: Estimote Beacons

2.8 Woowee MiP Robot

The Woowee MiP Robot [19] is a multifunctional and autonomous robot used as a toy for most of the people or as a platform for programming for others. Released at the end of 2014, the robot can perform interesting actions such as gesture and obstacle recognition, sound detection, weight lifting or tracking.

The robot makes easier to build a personal assistant because of the hardware capabilities it has, and especially as a result of the API available to control it through the Bluetooth Low Energy protocol. This official API, available in several programming languages, allows us to control all the robot's functions like lights, sounds, movements or modes, and with all this abilities create a visual interface for our application.

The script we have created to control the robot is easily applicable to both conversations apps. Using the BLE Protocol with our Python application, we have developed actions and behaviours for each case of use we have imagined in a smart place. This way we can decide the robot actions in each talk case.

As we have said, the MiP Robot uses Bluetooth Low Energy protocol to communicate with our application. The API implements a wide range of functions, which we can use through it or we can communicate directly by using the table of bluetooth messages provided by WooWee Labs.

Furthermore, the MiP Robot has been designed to be flexible, fast and easy to handle. Thus, the time that it is not getting an order, it stays active in default mode, programmed to interact with the people by using the hardware technologies it implements. In conclusion, this robot is a good option to create a personal assistant.

2.8.1 Technologies implemented

MiP's hardware include some edge technologies to get the ability of interact physically with a human. These technologies are used by the robot when it is in the free mode, but they can be programmed to use with our application giving us a lot of possibilities. Therefore we are going to see what is going to provide us each technology.

- ***Gesture Sensor:*** This sensor permit us make movements with our hand in front of the robot and get a response from it depending on the configuration mode.
- ***Bluetooth:*** The robot uses Bluetooth Low Energy protocol, a technology used for communicating small devices like smart-phones, computers or beacons in a smart

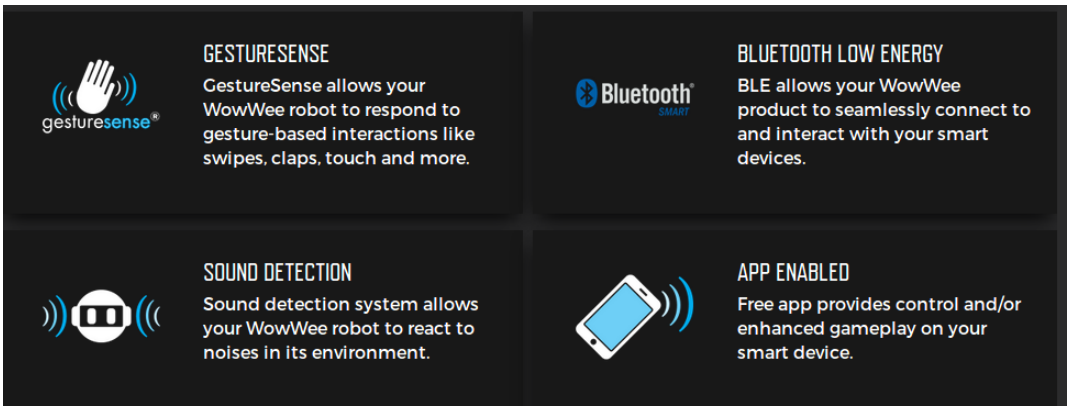

Figure 2.3: MiP Technologies

place. It is well known for its reduced power consumption and cost, while maintaining a good communication range.

- ***Sound Detection:*** A microphone permits the robot to listen basic noises like clap or hit and make a consequently response to them.
- ***Mobile App:*** WowWee has develop an app which permit us to control the robot or to choose a play mode. Our project does not interfere with the application, so the robot can be our personal assistant and immediately after, we can play with it through the app.

2.8.2 Response Actions

Accordingly to the previous point, the MiP robot has technologies to receive stimulus or messages, but now we are going to examine what kind of responses can it make.

- ***Sound:*** We can choose a sound between a library of 105 sounds the robot has, but we cannot reproduce any other kind of sound with it.
- ***Movement:*** We can order to the robot to move and to turn in any direction, and we can choose the speed or the time it has to move too.
- ***Light:*** The robot has a led light in the middle of it body, which colour can be configured by modifying the RGB parameters. This light normally indicates the current configuration mode of the robot
- ***Bluetooth:*** In the same way we can send Bluetooth messages to the robot to order actions, the robot can send us, through Bluetooth, messages which informs of the battery state, the obstacle detections or the weight on its tray.

CHAPTER 3

Requirement Analysis

3.1 Introduction

The purpose of this chapter is to describe the requirement analysis using different scenarios. This is one of the principal tasks when developing a software, so it is necessary to make a detailed analysis of the possible use cases. To be able to explain each case of use carefully, we will use the Unified Modeling Language (UML), because it allows us to specify, build and document a system through the use of graphic language.

Through this chapter, we will obtain a complete specification of the requirements for each module thought-out in the design stage. Thus, it will be key for knowing the actors and their interactions, being able to distinguish the importance of their functions in the development of each case of use.

3.2 Use cases

The sections below will analyse the use cases of the project, obtaining a complete specification of the uses of the system, and accordingly, defining a complete list of requisites. To start, we will take a look to the participating actors and their functions. Next, we will rep-

resent them in a UML diagram to know their rol in the system in the cases of use. However the use cases will be explained completely in following chapters.

3.2.1 System actors

Identifying the actors of the system is the first stage to perform when we are making an analysis of a system. The actors involved in the scenario are:

- ***User:*** Final user of the system, and the main actor. It accesses to the smart environment and interacts with it through the use of its voice and its smart-phone. It can make queries to the system by using the conversation application deployed on the Google Home device and it can also send events to Ewetasker application in order to trig actions defined by semantic rules.

- ***Google Home:*** This is a principal actor. It listens the user queries and sends them to the Api.ai conversation agent for receiving a response. Once the response is received, it makes a speech with it.

- ***Api.ai:*** This is other principal actor. It receives the user queries through the Google Home device and generates a response. The queries are sent to the Webhook application in case of making a response enriched with other actors data, and for triggering actions performed by other system actors.

- ***IBM Watson:*** This is an alternative to the Api.ai actor. It receives the user queries through a computer microphone and generates a voice response. The queries are processed by an application deployed in the computer with a similar function to Webhook application.

- ***Ewetasker:*** This is other principal actor. It takes over of the rules generated by the user, in function of the channels available in the smart environment. Events generated by the user will be sent to it in order to evaluate, and trigger the pertinent actions.

- ***Smart Devices:*** Together, they form a secondary actor in charge of sending events to Ewetasker and of carrying out own actions.

- ***Robot:*** This is a secondary actor. It is in charge of visual part of the system. Depending on the conversation dialogs and the actions triggered by the Ewetasker rules, it performs movements and sounds to transmit events happened or to improve and complement the responses obtained on a user query.

- ***Proxy:*** This is other secondary actor. Its function it is to trigger actions generated in the conversation or in the Ewetasker's rule engine.
- ***Daemon Time Module:*** This the last secondary actor. It is in charge of send time events each minute to Ewetasker's time channel.

3.2.2 Use cases

After that, we will present the use case diagram. In Fig. 3.1 it is shown the principal application use cases, and the connections between the system actors.

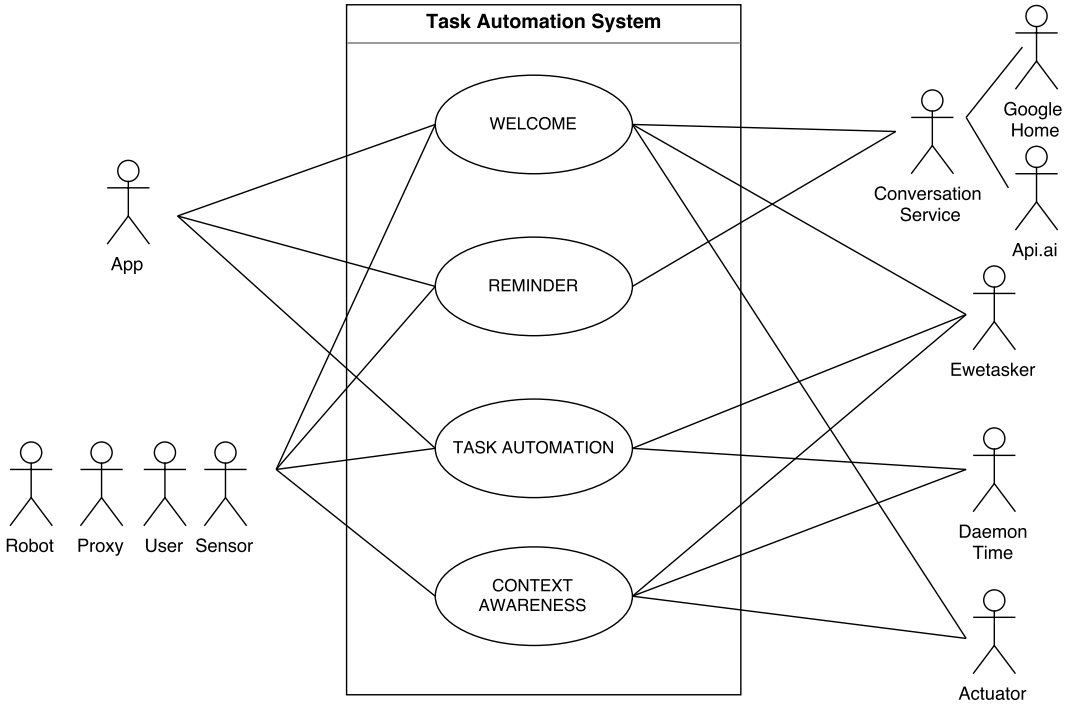

Figure 3.1: Use case diagram

3.2.2.1 Welcome Use Case

In this case, as shown in Fig. 3.2, different actors take part, even though their appearance is provoked by the user, the main actor. This actor starts the use case when it arrives to the lab door. Its smartphone detects a beacon Bluetooth signal and the Ewetasker application processes the event and triggers the actions associated to the rule *presence detected at door*. Those actions consist on sending a toast message to the smartphone application in order to ask for the door password, and making the robot goes to the door. Once inside the lab another beacon detects the user inside and throws other actions to make the robot go in front of the Google Home and the smart tv, and to start a video presentation using Google Chromecast. This video ends asking the user to start a conversation with the Google Home assistant.

Once finished this first part of the welcome case of use, it starts the second part when the user begins the conversation with the other actor, the Api.ai conversation agent using the Google Home actor. This action activates the application in charge of control the movements of the robot by sending the commands through the proxy actor. Then the user can ask questions about the lab space, and about its history or its rooms. Each response reproduced by the Google Home actor is accompanied by a robot performance.

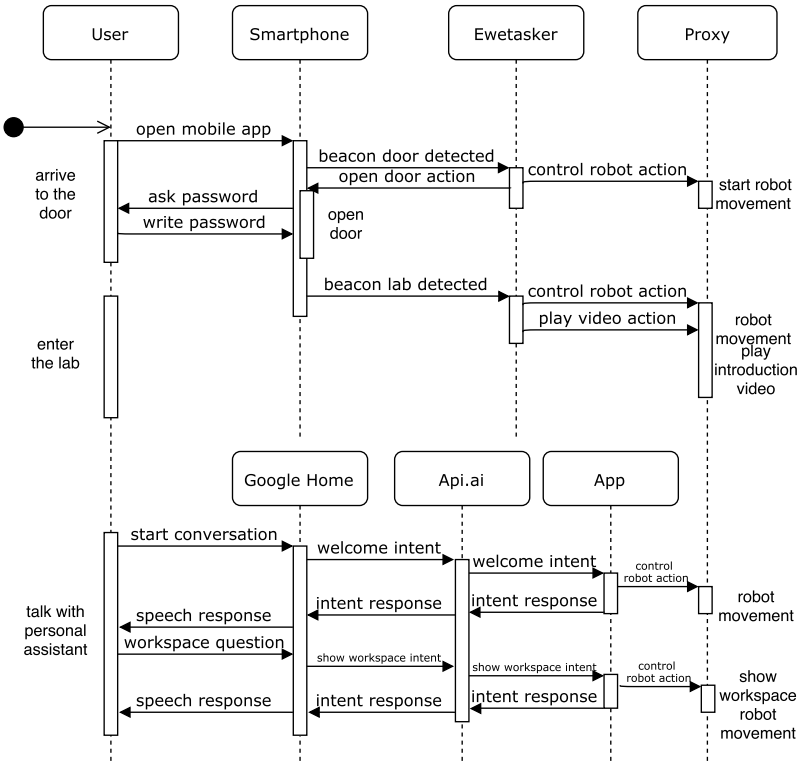

Figure 3.2: Welcome case diagram

3.2.2.2 Reminder Use Case

The actor of this use case, represented in Fig. 3.3, is the user. In this case, the user starts the conversation with the Api.ai conversation agent using the Google Home device. Once conversation is started, the user asks to create a reminder. This action leads into an order to the application to start saving the reminder parameters, like text or time of trigging. When the dialogue ends, the application makes a post petition to Ewetasker service in order to create a rule with a time event channel and an Android Toast action channel. This rule will be evaluated each minute because other actor, the Daemon Time module. This module will be sending time events each sixty seconds. If the rule is satisfied, Ewetasker will trigger an action to the smartphone to show the user a message with the previously saved reminder.

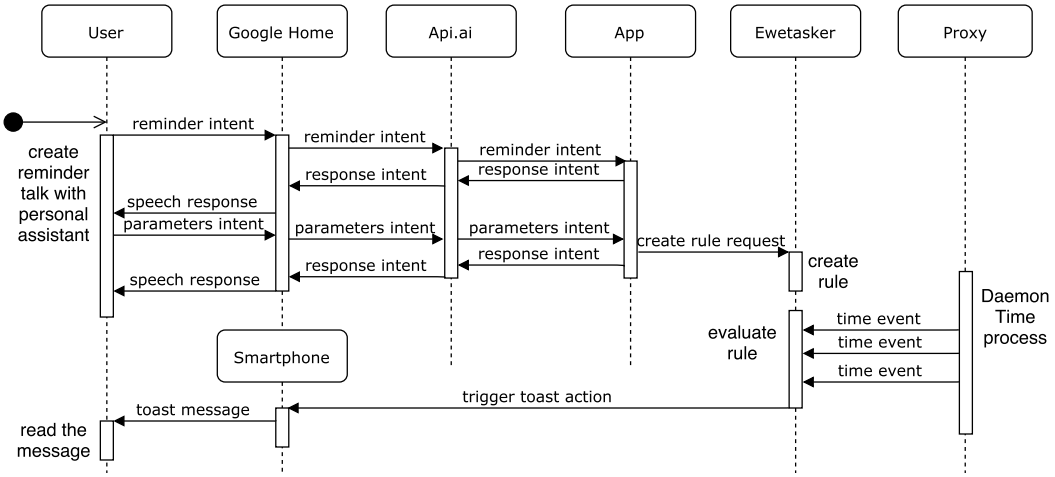

Figure 3.3: Reminder case diagram

3.2.3 Conclusions

Through the use cases described we have introduced some of the basic functionalities we have been implemented in this project. Now we can understand the way in which actors interact each other. In addition, they can be a base for further development and different use cases that can emerge in an intelligent environment like the one treated in this project. This is a large development field, hence it can be part of future work lines.

CHAPTER 4

Architecture

4.1 Introduction

The main purpose of this chapter is to explain the architecture of this project, going through the design phase and the implementation details. Firstly, we will present a global vision about the project architecture in the overview, looking at the modules which form the system and its connections. Next, we will focus on each module explaining its function in this project.

4.2 Overview

In this section we will present the global architecture of the project, defining the different subsystems that participates in the entire system. We can identify the following subsystems:

- ***Conversation system:*** This is the main system of the project, its function is to manage user queries, in order to enrich the responses, to make actions with the smart devices present in the environment and to generate rules for automation tasks with Ewetasker system. It is divided into two implementations, on the one hand, the

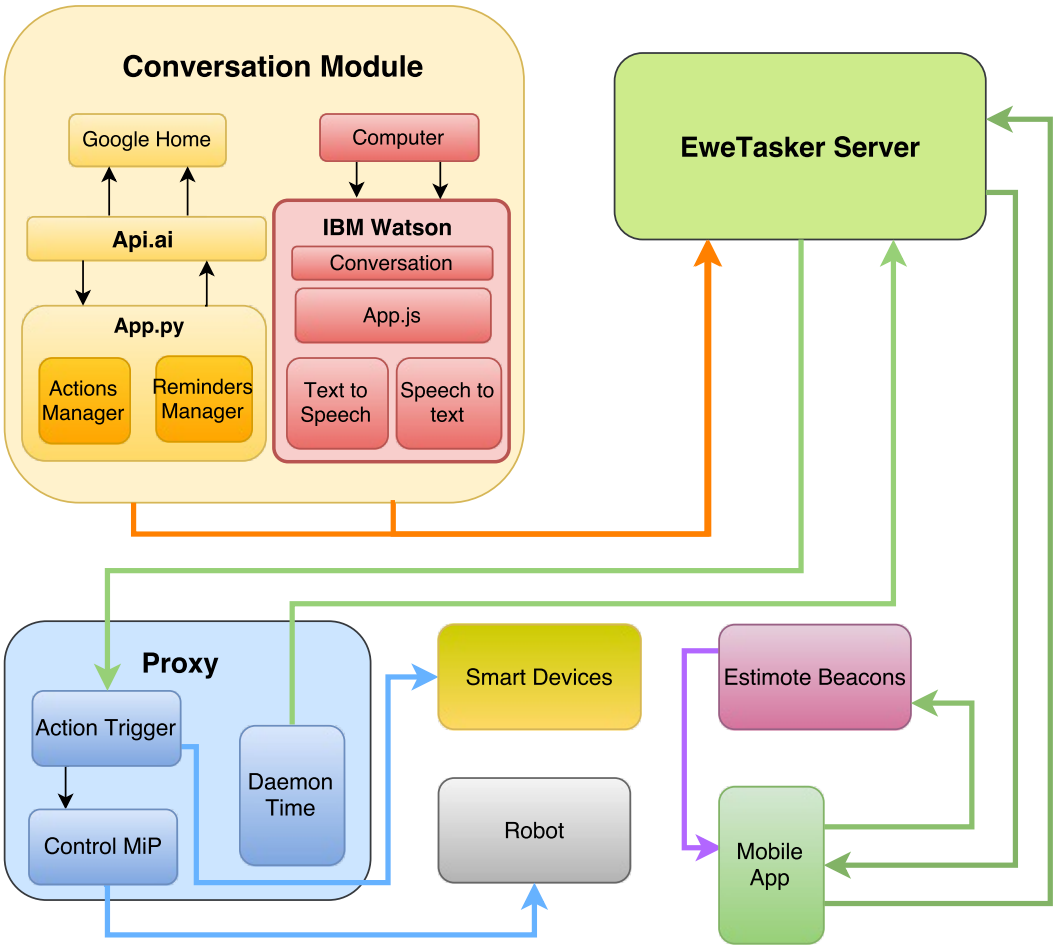

Figure 4.1: Architecture

Api.ai module, which is connected to a Python Webhook submodule named ***App*** and also connected with two Python submodules to carry out the functions previously mentioned named ***Action Manager*** and ***Reminder Manager***. And on the other hand, the ***IBM Watson*** module formed by the three Nodejs submodules ***Conversation, Text To Speech*** and ***Speech To Text*** controled by the submodule ***App*** developed in Nodejs too.

- ***Ewetasker server:*** This server is in charge of creating rules and of triggering actions for the smart environment.

- ***Proxy server:*** The server is composed of three modules: Daemon Time, Action Trigger and Control MIP. ***Daemon Time*** is a python module in charge of sending events to the Ewetasker time channel. Also it is composed of PHP submodule, named ***Action Trigger***, to trigger actions to the smart devices that are connected to the smart environment network. Finally, ***Control MiP***, is a Python library that provides

an API for moving the MIP robot.

- ***Mobile Application:*** The Android application is a part of the system which receives beacons signals and send events to the Ewetasker system, and also performs actions like showing messages to the user when certain conditions are met.

In the following sections we are going to describe deeply subsystems involved in the project.

4.3 Conversation system

One of the main purpose of conversation agents is to be a tool of information transmission for people, with only a human interlocutor, who is ultimately the one who asks for and receives the information. Currently, this stage has been completely overcome by the implementation of cognitive computing technology in conversation agents, opening new investigation fields. At the same time that this was happening, new developments appear in the Internet of Things field, what puts us in the challenge of making a conversation agent into a tool for tasks automations emerged in smart environments.

This is the starting point of the project and of the development of our Conversation System. Like we have previously told, we have made two implementations with Api.ai and IBM Watson, in order to know which of them adapts better to our study cases.

Lets start with Api.ai module, that has been implemented using a Webhook in Python, and two other submodules in Python too. For using this module, the conversation agent created has been implemented into a Google Home device.

4.3.1 Google Home

In this project, we look for making a smart space more accessible to the people which join this space. Thus, we start to think the best way to integrate our conversation agent in a device capable of carrying out our conversations easily. In the same way we started this search, we were trying to decide which conversation tool will be the best for our case. Here is when we realize that Api.ai application allows us to integrate the conversation agent created into a Google Home device. For this reason, we choose Google Home as physical tool to make our conversations possible.

This device allows us to install on it our conversation agent through the Api.ai web interface. This way, we can give a calling name to the agent for invoking it similarly to any

order that the Google Assistant can understand. Once this is done, we will have a perfect voice interface to communicate with our Api.ai conversation agent and consequently with the other submodules which integrate the conversation module.

4.3.2 Api.ai

Api.ai application, permits us to make a conversation agent, with plenty of functions, integrations and parameters to characterize. Due to this, and of course, its easy integration with the Google Home device, we have choose it for giving voice to our personal assistant.

To make our talk agent we have used the Api.ai web interface. With that interface, we have designed firstly the entities, which are groups of words with the same meaning that can appear during the conversation. These words will serve us like parameters of our system functions. One example of the entities designed for our project could be these: robot, channel, company or workspace.

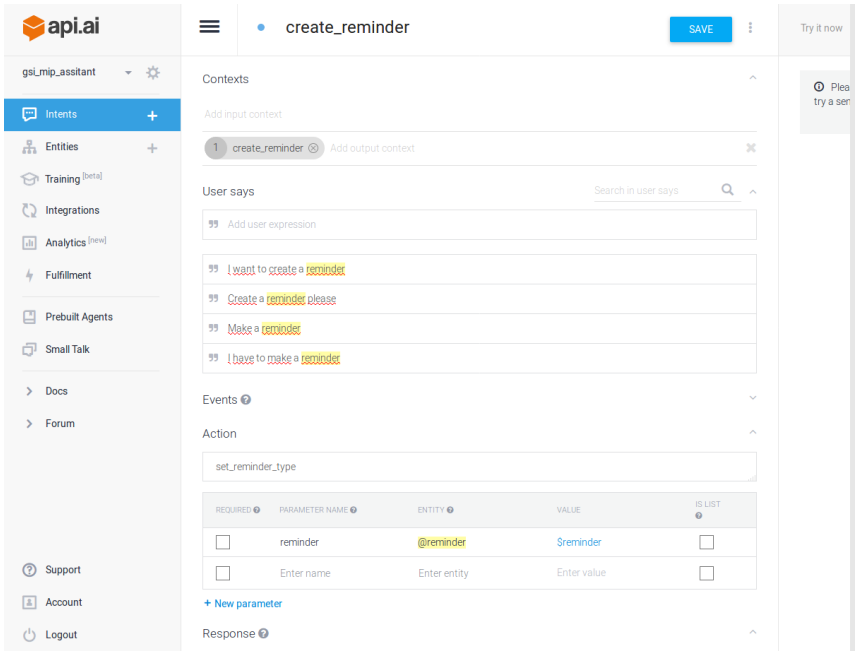

Figure 4.2: Api.ai web interface

Secondly, we have developed the intents, which are based on user queries. For a good performance, they have to be similar between them and with the same purpose. Once they are received, agent responds to the queries. Intents include a context field to organize them in a way they make sense on a dialogue. The responses can be enriched or changed using a external web application, connected to our agent, like we are going to use in our project.

Once introduced these concepts, we have thought the actions we want to perform with our agent. Like we have previously mentioned, we are developing a personal assistant in a smart environment. For this reason, actions to perform have to be designed for making tasks related with smart devices and taking advantage of the possibilities of Ewetasker semantic technology to automate them.

Now, let's take a look to the submodules which integrate the Api.ai module.

4.3.2.1 App Module

This submodule is developed using the web development method named Webhook. This method, that consists in providing and receiving data from other web applications using a callback, was created in 2007, and nowadays is a simply way to integrate different web services.

For this, Api.ai application recommends to develop talk agents which require to collect information of other web applications, or to implement functions for the management of other applications. The programming language used for developing this submodule is Python, due to its ease of programming.

In our case, we use both of them. We need to show information of Ewetasker Server to the user, and we need to send requests to trigger actions of the smart devices and to create semantic rules.

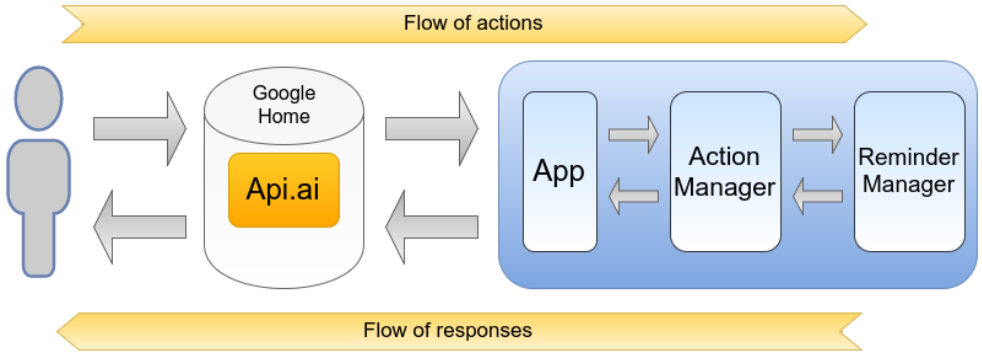

Figure 4.3: Api.ai Modules Flow Diagram

The operation mode is simple, our conversation agent running on Api.ai server sends POST requests to the web direction of our app. In this request, it is included JSON document, shown in 4.1, with all data fields of the intent triggered in the conversation. Once the data is received, the app is in charge of examining fields like *intent name*, *parameter* or *action* to decide which action has to perform and the suitable answer. For giving a visual

interface to the talks, App sends a message to the Proxy Server which contains a submodule to control robot movements. In case an action field includes an action, the app calls to the Action Manager submodule, which manages the the call and returns a response accordingly.

Listing 4.1: Example of response in JSON format from Api.ai:

```
{
...
  "contexts":[
      {
      "name":"create_reminder"
      ...
      }
    {
      "name":"ewetasker_infochannel",
      "parameters":{
        "workspace":"workplace",
        "reminder":"reminder",
        "workspace.original":"workspace",
        "reminder.original":"reminder"
      },
    }
  ],
  "metadata":{
    "intentId":"5e08c951-32be-4642-b9e1-20ff10e09f84",
    "webhookUsed":"true",
    "webhookForSlotFillingUsed":"false",
    "intentName":"create_reminder"
  },
  "fulfillment":{
    "messages":[
        {
          "speech":"Ok, if you want to create a reminder, let me make some
              questions. What do you want to remind?"
        }
    ]
  }
...
}
```

4.3.2.2 Action Manager

The submodule Action Manager is part of the Api.ai module. It carries out the actions programmed in the action field of the intents performed by our conversation agent. This means that it triggers the asked actions by the user. It is developed in Python language to get a perfect integration with the submodule App. That module is in charge of parsing user queries and send them to this module if necessary. It also sends user queries to the submodule Reminder Manager in case of performing the action *create reminder*.

The main purpose of the submodule is being a voice interface to control the smart devices present in the environment. In the same way that a user can automate tasks for the smart devices using Ewetasker rules, we realize that it will be interesting for the user not only to program actions, but also to perform them by the conversation agent help.

As a result, the module is designed to order actions to the smart devices through the Action Trigger submodule located in the Proxy server. As we have said, the submodule also makes functions to get the parameters needed to create a reminder or make an automation. In addition, it makes a response to the user query.

4.3.2.3 Reminder Manager

Finally, the last submodule involved in the Api.ai module is the Reminder Manager. It is developed in Python like other submodules and its principal function is to manage the creation and the shipping of reminders and automations to the Ewetasker server.

To carry out these functions, it collects from the conversation, and more specify through its functions called by the Action Manager submodule, the required parameters to create a rule. Once all parameters are collected, the Action Manager submodule calls to the Reminder Manager function named *makeReminder*. This function decides between making an automation or a reminder. Then, it sends a POST request to the Ewetasker server. Finally, it makes a response for the conversation reporting the success of the call.

4.3.3 IBM Watson

Other alternative for the conversation module developed in this project is IBM Watson. This service, based on many different tools, has a part for building conversation agents named Conversation. It is very similar to the Api.ai conversation platform. However, it is necessary to use and configure two more services whose names are Speech To Text and Text To Speech. With these two services we get a talk agent that can be controlled by voice.

The principal features of the Watson Conversation service are the use of intents and entities to design our conversation agent. Still, there is a huge difference between it and Api.ai service in the way they organize the intents to make a conversation with coherence. Api.ai service uses a context parameter whereas that Watson Conversation service organizes its intents using nodes which relate to each other. This allows us to design our conversation agent on a very intuitive way through its web interface.

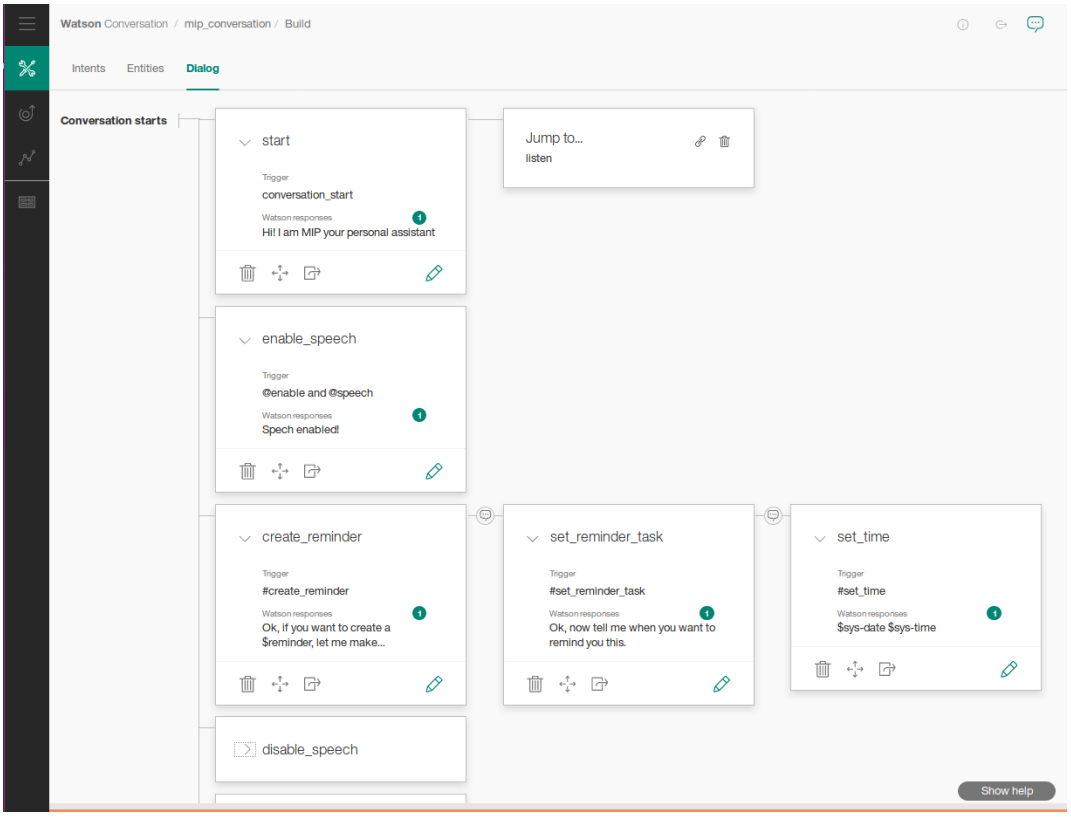

Figure 4.4: Watson Conversation web interface

Now, lets explain how the rest of submodules work. They provide an alternative conversation module to the API.ai one.

4.3.3.1 App Module

This submodule is based on a NodeJS web server, which controls the Watson submodules and provides a web interface to users in order to enable the dialogue with with the conversation agent. This interface allows users to follow the conversation by showing the inputs and outputs produced. Moreover, in order to make the interface more dynamic, its is possible to make intents with voice by using a computer microphone. On the other hand, we also use speakers to get the response in sound form.

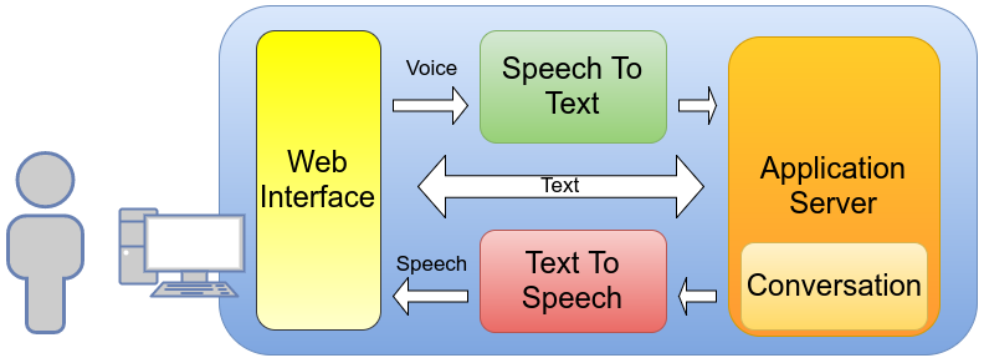

Figure 4.5: Watson Modules Flow Diagram

More in detail, server application is designed on a similar way to the app developed for the Api.ai module. However, for interactions between our app and Watson API, we have used the IBM Watson SDK for NodeJS.

Like we have mentioned, user inputs can be collected from web interface or from the computer microphone (in which case the sound captured is recognized by the Speech To Text submodule). Once got it, they are sent to the conversation method which makes a call to the Conversation API. This method returns a JSON data with the response to the intent, including important fields like parameters or dialogue node names. These parameters will be parsed by the application to trigger the actions in consequence. After that, the server sends the response text to the web interface and to the submodule Text To Speech in case of the user wants to listen the response too.

The actions that server application can perform are those of the Reminder case study. This means it make POST requests to create rules on Ewetasker server and to order actions to the smart devices present in the environment.

4.3.3.2 Conversation Module

The conversation submodule is integrated in the server application submodule in order to facilitate the data flow between the functions that are developed on the server.

Its function consists in sending the user queries to the Conversation api through the use of Conversation SDK. As we have said, this request returns a JSON data, shown in 4.2, with all information related to the intent made and with the conversation agent response. The information obtained will be analysed by the application.

Listing 4.2: Example of response in JSON format from IBM Watson Conversation:

```
{
  "intents": ["create_reminder"],
  "entities": ["reminder": "reminder"],
  "input": {
    "text": [
    "I want to make a reminder"],
  },
  "output": {
    "text": [
    "Ok, if you want to create a reminder, let me make some questions. What
         do you want to remind?"
    ],
    "nodes_visited": ["start", "create_reminder"],
    "log_messages": []
  },
  "context": {
    "conversation_id":"6c3863f1-9016-4432-8643-1594c0953245",
    "system": {
      "branch_exited": true,
      "branch_exited_reason": "completed"
    },
  }
}
```

4.3.3.3 Text To Speech Module

Text To Speech submodule is needed to convert the conversation agent responses into voice. This allows users listening the conversation instead of reading it on the web interface. The submodule final goal, with Speech To Text submodule, is to give users a voice interface able to emulate the Google Home function in the Api.ai module.

The submodule works using the Conversation SDK. The app sends the conversation response to the text to speech submodule. Once inside it, it makes an API call that receives again the response in form of sound. Finally, this submodule reproduce the sound through obtained the computer speakers and ends the process. In addition, this feature of the conversation module can be enabled or disabled by the user with a request to the talk agent.

4.3.3.4 Speech To Text Module

This submodule, Speech To Text, makes the opposite function to the previous submodule. In other words, it can convert the sound obtained by the microphone into text. It permits us to send the user query to the conversation agent in the same way that writing it.

Submodule operation starts when user clicks on the chat icon to activate the recording through computer microphone. Then, user talks to the microphone saying the query content. Once finished the process, the sound is sent to the IBM Watson Speech to Text API, that responds us with the text recognized. The transcription will be saved on txt file. After that, this file will be read and converted to a string in order to send it to the app submodule. In this way, it will be sent like a user query to the conversation agent.

For the development of speech to text submodule, we have used the Speech To Text SDK for NodeJS, like in other previous submodules.

4.4 Ewetasker server

The main purpose of this module, designed by the Intelligent Systems Group, is to manage certain tasks handling events and triggering actions in an automated way. For this reason, it fits perfectly on the development of our task automation system. To understand better the operation of this module, we are going to introduce six concepts present on it:

- ***Channel:*** It defines subjects which can generate Events, provide Actions or both. Sensors and actuators are also described as channels, therefore they produce events or provide actions.
- ***Event:*** They define the realization of a fact. These facts are defined on a channel, but they are used on the rules. When a rule is created, one or more events should be defined to trigger the action. Events also let users describe with parameters under which conditions should they be triggered.

- ***Action:*** It defines an operation or process that can be performed by a channel. Like events, they are defined on the channel creation. Besides, when creating a rule, we can define parameters necessaries to make the action.
- ***Rule:*** It is defined as an "Event-Condition-Action" (ECA) rule. This rule is triggered by an event, and then, it executes an Action. Its purpose is to define the connections between the Events and Actions present on two or more channels. In order to perform this, when user creates it, includes the configuration parameters set for both of them.
- ***Sensors:*** They are individuals capable of recognizing the facts defined in an event. It means that they have the ability to obtain the parameters waited by the rules. They are closely related to the channels, when they are not directly implemented like one fo them.
- ***Actuators:*** They are individuals capable of performing the actions defined on a channel. Same as sensors, they are closely related to the channels, when they are not directly implemented like one fo them.

Once explained this, we are going to talk about which of them have been necessary to the development of our project.

4.4.1 Channels Created

The project is based on two principal channels, that have been developed in order to carry out two functions: make reminders or automations and control a robot. These channels have been implemented with Ewetasker web interface.

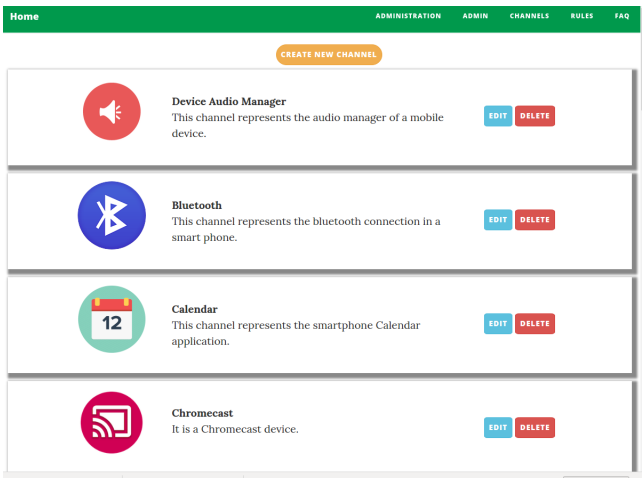

Figure 4.6: Ewetasker Channel Administration Interface

4.4.1.1 Time Channel

To make the first one, we have to understand the reminder concept. A reminder is a kind of alarm, that it is activated at a specific moment. This meaning is closely related to the time. For this reason, we have thought that the best way to create a reminder is making a time channel. In addition, this channel can give us the same function when we make a task automation.

This channel is defined only with a type of event, Time Instant. The event waits for receiving a time parameter following the OWL-Time ontology [17]. There are not actions defined for this channel, so the actions will be performed by another channel assigned on the rule created.

4.4.1.2 Robot Channel

This channel is designed to perform the actions programmed for the robot. More in detail, we have defined an action, Control robot, with a movement parameter. The parameter will be configured by user when creating a rule. When control robot action is triggered, the parameter will be sent to the Control MiP submodule to make the movement. There are not events for this channel because its function is implemented as an actuator.

This channel, is defined using the EWE [3] ontology created in the Intelligence System Group. Moreover, the channel uses RDF syntax [16] to define it likewise other channels present on Ewetasker application.

4.4.2 Sensors involved

The base of this project is the interactions between the user and the smart environment that surrounds it. Consequently, the role of the sensors becomes more important. The Ewetasker system incorporates some actuators controlled by defined channels, and we will incorporate the Daemon Time module, in order to implement the Time Channel. The sensors involved in our project are:

- ***Beacons:*** They send a BLE message to the user smartphone. This message is parsed in the Mobile Application module to know the distance between the user and the beacon.
- ***Daemon Time module:*** It sends Instant Time event to Ewetasker server each minute. Due to this is possible to create rules activated by time.

4.4.3 Actuators involved

They form the visual part of the project because they are in charge of performing the actions ordered by the user or triggered by Ewetasker server. The actuators involved in our project are:

- ***Robot:*** It is capable of make movements, sounds and lights. It can be controlled by Ewetasker rules or by the conversation agent. For this, we have developed submodule named Control MiP with a movement library.
- ***Light:*** It is a smart light, that can be controlled making an API call. Its configurable parameters are the light state and the brightness.
- ***Smartphone:*** Controlled by the Mobile Application module, it can be configured and show messages to the user.

4.5 Proxy Server Module

The main purpose of the Proxy Server module is to control the smart devices present on the environment. This task will be performed by the submodule named Action Trigger. Besides, it incorporates a submodule named Daemon Time, which function have been commented before, but that we are going to explain in more detail. Lets start talking about the functions and features of the server.

4.5.1 Action Trigger

Thinking on the development of the actions that would happen during the case studies of this project, it emerged the problem of controlling all the devices present on the environment from the same interface. This problem was bigger when we locate Ewetasker server outside the network used by the smart devices. For this reason, we decided tp create a proxy server. This server receives the requests to trigger actions from Ewetasker server and from the applications developed for the conversation systems.

The server is developed using PHP language. It waits for requests, shown in 4.3, that contain the parameters required to activate the actions. Once obtained, it executes the function in charge of the smart device.

Listing 4.3: Example of POST request to Proxy Server for switching on the light using CURL:

```
curl --data "channel=HueLight, action=TurnOn" http://irouter.gsi.dit.upm.es
    /actionTrigger.php
```

4.5.2 Control MiP

This submodule is in charge of controlling the robot movements. To carry out this, we have developed a library of actions. The library is programmed using Python language and it uses Bluetooth technology to communicate with the robot. This library uses a repository named mip [18], which implements the Bluetooth Protocol [13] created by WooWee company to control a WooWee MiP robot in Python.

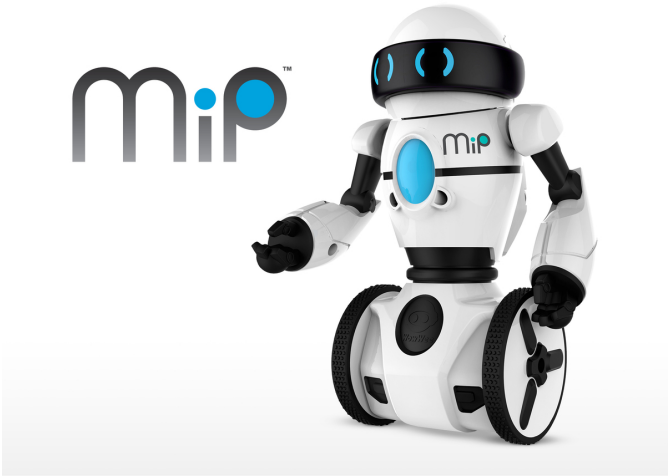

Figure 4.7: MiP WooWee Robot

The Control Mip submodule is executed by Action Trigger submodule. That submodule sends the movement parameter to Control Mip. Then, it calls to the function required. Each function is a succession of orders to send Bluetooth messages to the robot. Once the process ends, the robot returns to the default mode, in which it can interact with the user through gesture recognition or clap detection.

For the development of this submodule, we have created a library of movements and a library of sounds. The first one includes movements designed to be performed with Ewetasker rules, which means, they are executed by the occurrence of an event. The other movements programmed have been created with the goal of be a visual interface for the

conversation agents. These movements are briefer than the others and include more sounds and more light performances.

Sound library is a selection between more than one hundred sounds that robot can reproduce. These selected sounds are divided into emotions and meanings. In order to make dynamic conversations, the functions developed to be used by the conversation agent use a specific method for the sounds. This method consists in choosing randomly the sound reproduced between the selection made for the movement.

4.5.3 Daemon Time

The operation mode of this submodule is very simple. As the Action Trigger submodule, it is working all time. However, its function its totally opposite to it, while Action Trigger is waiting for requests, Daemon Time is sends a request each minute. For this reason, the reminders and automations made with Time channel of Ewetasker, waits for a Time event with the precision of a minute.

For developing this submodule, we have programmed a daemon process in Python. This process sends a event Time with the current time of the moment of sending. Once made it it sleeps for sixty seconds until the following sending.

4.6 Mobile Application

A Mobile Application is used in this project to show reminder messages to the user. We decided to take the existing application [6] developed by the Intelligence System Group and improve it for accomplishing our goals.

This application is programmed to interact with the beacons located on our smart environment. This way, the application is listening for the BLE messages broadcasted by the beacons. When a beacon is recognized by the app, the app sends an event to the Ewetasker server. Events will include parameters like distance, or the beacon ID in order to be evaluated by rules. If the rule is satisfied, Ewetasker will send the action to the appropriate device.

CHAPTER 5

Case study

5.1 Introduction

This chapter describes the process we have followed to implement the study cases of Welcome, Reminder, Task Automation and Context Awareness described in Chapter 2. For this purpose we will go over them to show the main features.

5.2 Welcome Use Case

This use case is meant to facilitate the adaptation of a workplace to new employees. For this scenario we have chosen the Intelligent Systems Group as the Company. The laboratory of the Group will be our workspace. The location of the lab is inside our School, in the Technical University Campus in Madrid.

First, we will identify the main actors in this use case:

- ***New User*** This user goal is to experience a nice welcome into his new workplace. In order to this, it will attend a presentation of the company, which will include the first interaction with the conversation agent. In this conversation, user will know

more information about the company and about the smart environment, making his transition to the new workplace easier.

- ***Conversation Agent*** The goal of this actor is to help user in its welcome to its new workplace. It will be capable of responding questions about the company, the projects and the work rooms and of course it will show the devices present on the smart environment. This way the company can ensure that new employees are properly introduced to the workspace.

For the design of this case of study, we have thought the actions that can be developed between the moment that user enters for first time in the workplace until that user goes to its own desk. These actions will be performed by different secondary actors.

To get an idea of the actions triggered by the user from entering the door, we have this diagram:

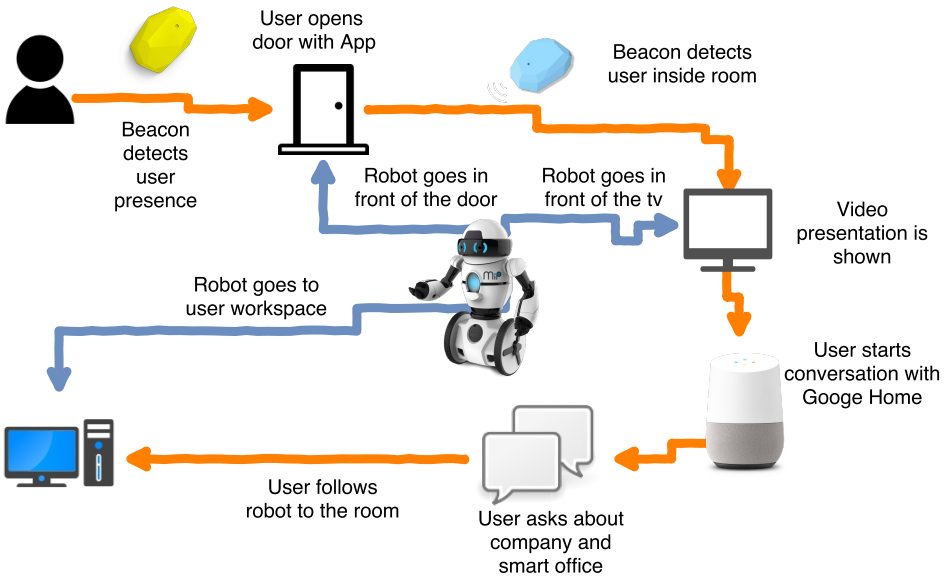

Figure 5.1: Case Use: Welcome

More in detail, the actions start when user goes in front of the door. To open the door it will have to open the Mobile App, that will detect the beacon located outside the door. Then the app will ask for the password to open the door. It will be the moment in which robot goes in front of the door. Once inside, other beacon will detect user. The robot goes in front of the tv, and a video presentation starts. The video finishes asking user to start a conversation with Google Home.

In the conversation between user and the conversation agent, there are many company

topics to talk. The user can ask about company history, about the projects in which are involved, or about the smart devices present. Besides, it can ask about the rooms which form the office and their uses, or about its own workplace. In this last two cases, the robot will show the spaces to the user.

This diagram shows an example of the conversations that our conversation agent can make for the Welcome use case:

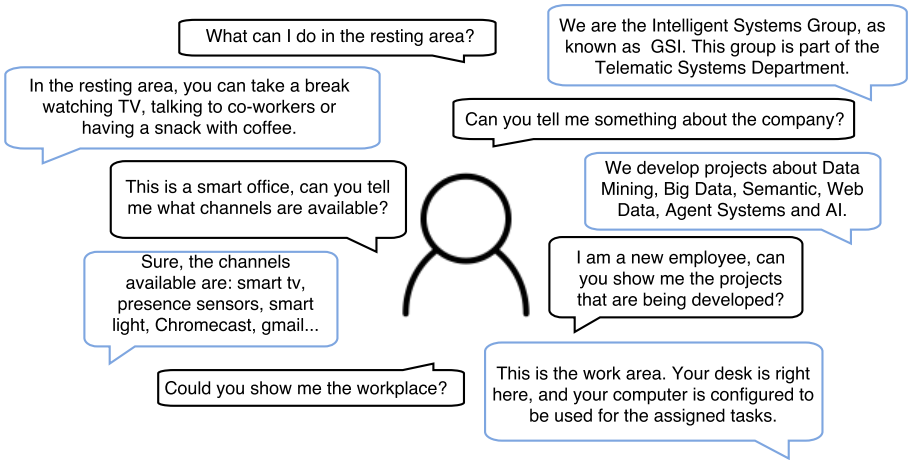

Figure 5.2: Welcome Use Case Conversations

The following picture shows the environment where the welcome use case takes place and some agents involved:

Figure 5.3: Welcome Use Case Environment

5.3 Reminder Use Case

The purpose of this use case is to provide a tool to the employees for remembering tasks to perform during their working day. For this scenario, we also use the laboratory of the Group as our workspace.

First of all, we will identify the main actors in this use case:

- ***User*** This user goal is to create a reminder for a work task that he want to be reminded on a concrete time. To carry out this, it will start a conversation with the conversation agent, in which it will explain the purpose of the reminder and at what time it wants to be warned.

- ***Conversation Agent*** The goal of this actor is to carry out the dialogues needed to create a reminder, saving the parameters to configure it and finally sending to Ewetasker the order to create it. It will prepare to receive user speeches that includes the reminder data, and it will ask user if a parameter is misunderstood.

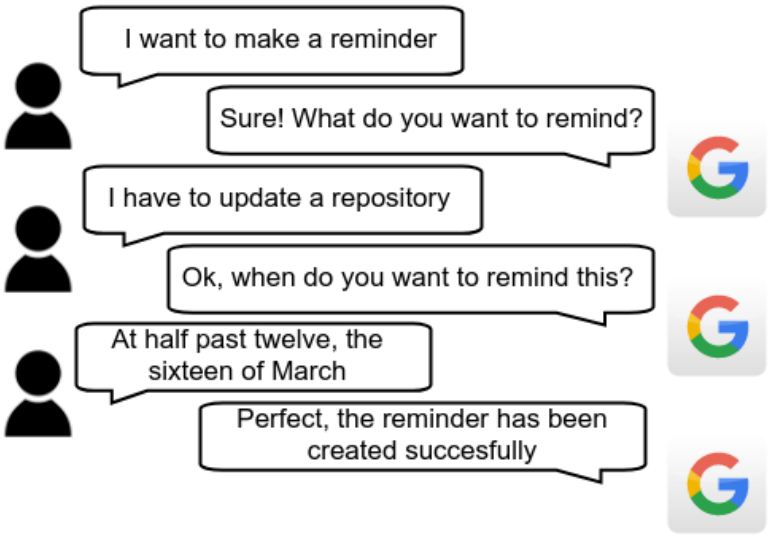

Figure 5.4: Case Use: Reminder using Google Home

More in depth, the case starts when user talks to the conversation agent. In this moment, the user says to the agent that it wish to create a reminder. Then agent starts a succession of dialogues. First, it asks about what user wants to remember. Once responded, it asks again for a date and a time to trigger the warn. Finally, if the reminder has been created in Ewetasker, the agent responds with success.

The second part of this case of use, starts in the moment that the time for the reminder is accomplished. In this moment, the Ewetasker rule created for the reminder is satisfied and it activates the action. It will consist on showing a Toast message in the smartphone application with the reminder created by user.

It is important to add that to carry out this case use, we have implemented two different technologies for the conversation agent. The conclusions will be explained on next chapter.

5.4 Task Automation Use Case

This case of use is based on the previous case. In the same way that user can need remember something on a concrete moment, it is interesting to be able to use this development to automate smart devices tasks. This development is very useful in smart places, where user has at its disposal many devices. For this scenario, like previous cases, we have used the laboratory of the Group as our workspace.

The main actors that participate in this use case are:

- ***User*** This user goal is to make an automation for a device in order to perform a task on a concrete time. To achieve this, it will start a conversation with the conversation agent, in which it will select the device to automate, the task to perform and at what time it wants to be performed.
- ***Conversation Agent*** The purpose of this actor is to perform the conversation developed to create an automation. In the conversation, user will be asked for the parameters to configure the device and finally it will be send to Ewetasker the rule which contain the automation. The conversation agent will know the devices available, the parameters required in each case, and the actions they can perform. If during the conversation a parameter is missing, it will ask again to user in order to create the automation with success.
- ***Smart Device*** This mission is to perform the tasks automated by user. These tasks will be activated when the Ewetasker rule created is satisfied for a Time event.

Taking a look to the interactions occurred for the development of this use case, we can summarize them this way. First, the user talks to the conversation agent to create a new automation. During the conversation, the application that controls the agent saves the data necessary to make the automation. Once it has all parameters, the application sends

a request to Ewetasker to create a rule. This rule will be satisfied when the event Time received each minute coincide with the time that user has programmed for the task. As a result, Ewetasker triggers the action, and the task is performed by the appropriate device.

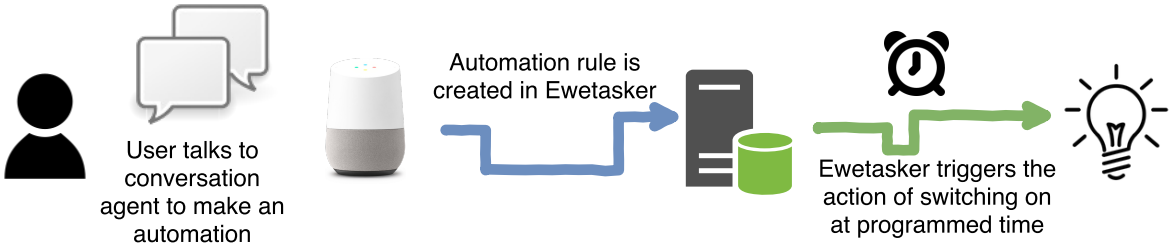

Figure 5.5: Case Use: Task Automation

5.5 Context Awareness Use Case

This use case goal is to be able to give life to the environment formed for the objects and furnitures present in a workplace. For this purpose, we will use Estimote stikers as an employee, that are similar to the beacons used in previous cases. With this we can make objects interacts with us, without being smart devices.

The application examples of this technology are very numerous and varied. In this project, we developed three of them, which will help employees to have healthy habits and to control the time they devote to leisure in the company. For this scenario, we have chosen the laboratory of the Group as our workspace.

For this use case we only need one main actor, that will be the user. As a employee, it will be performing its tasks and it will interact will the context on a natural way. However, its mobile phone will be in charge of collecting the signals broadcast by the stickers present in the office environment.

This way we will know for example, the times that user has taken a coffee or the time he has spend sitting in its chair. These data are saved in the mobile application, which will show a Toast message to the user when the counter programmed for each sticker reach a concrete value. Thus, we achieve the objective of making user conscience about its habits. Besides, we empower the possible interactions with the context, making our company smart environment more alive.

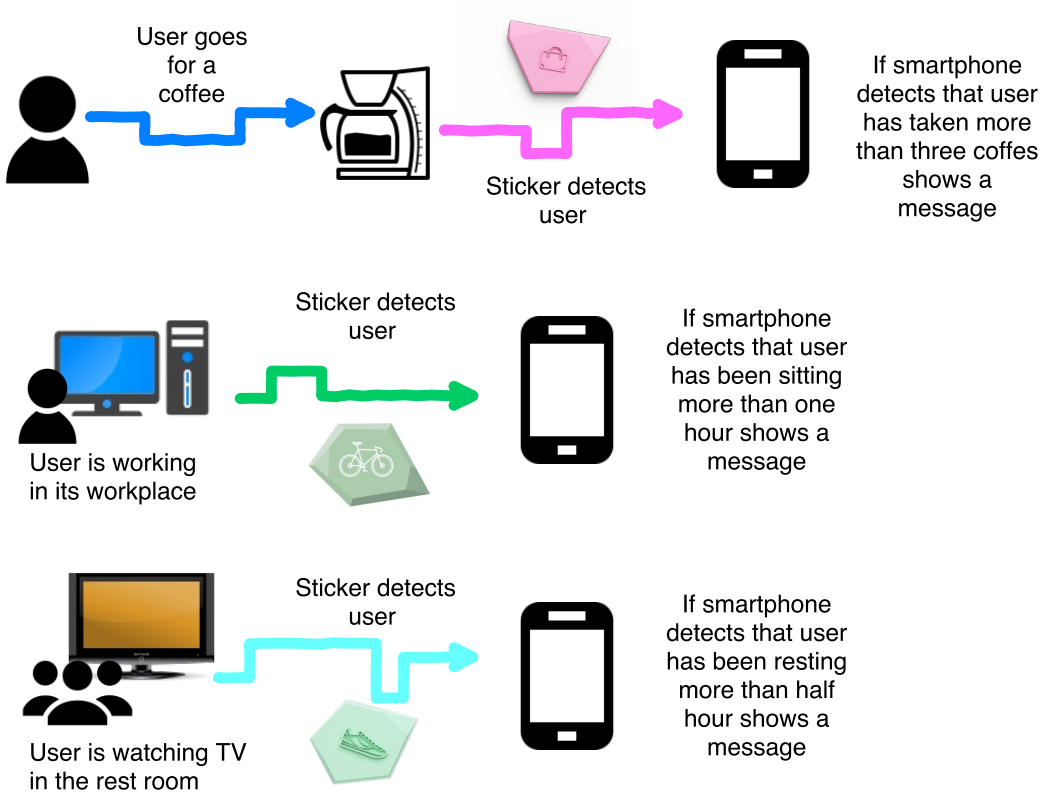

Figure 5.6: Case Use: Context Awareness

5.6 Conclusions

In this chapter we have presented the study cases we have designed for task automation using semantic rules through a conversation agent in a smart office. We have explained how the interactions between the different actors involver in each case made, with which purpose occur, and what actions trigger.

To sum up, these cases of use look for covering some of the principal needs that can be present in a labour smart environment. At the same time, we have been able to observe the perfect integration of semantic rules on an automation system and more specifically, the multiple choices that its use gives us along with the conversation tools.

CHAPTER 6

Conclusions and future work

6.1 Introduction

In this chapter we will describe the conclusions extracted from this work, problems, achievements and thoughts about future work.

6.2 Conclusions

In this project we have created a conversational interface to task automation using semantic rules. It is based on Api.ai and IBM Watson conversation tools to make dynamic and interactive conversations. The semantic part is empowered by Ewetasker system, which provides us a perfect platform to create semantic rules for our automations. In order to achieve this goals, we have developed submodules in charge of concrete tasks resulting a complete and structured system. Lastly, we have created a visual interface for our conversations through a MiP robot, making it our personal assistant in the smart office.

This project is formed by three subsystems, the main one is the conversation system described above. Another important system is Ewetasker, which is in charge of creating the rules ordered by the conversation system and evaluating them. Lastly, the proxy system

which principal function is to control the smart devices found in the office environment in order to perform tasks previously automated.

We will now look at the advantages and disadvantages of each of the conversation tools used in this project.

6.3 Api.ai vs IBM Watson

Although both conversation tools exceptionally fulfil their function, during the development of the project we found some differences between them. These small differences can make us choose between one option or another depending on the characteristics of our project. Here is a list with the conclusions drawn.

- ***Voice recognition:*** Both services integrate this feature, but IBM Watson offers it as a service apart from its conversation service. For this reason the developer has to undertake the interaction between the two services. However, the IBM voice recognition service seems to be more reliable than Api.ai.
- ***Speech quality:*** As in previous point, IBM Watson offers this service apart from others. Nevertheless, it achieves a better pronunciation and fluency than Api.ai.
- ***Development interface:*** Both offer an intuitive interface for creating conversations. On the one hand Api.ai offers a more complete and better organized interface to create intentions. On the other hand IBM Watson offers a more basic interface but with a strong point, its tree of conversation nodes. While in Api.ai we have to remember the connections between intents made through contexts, IBM Watson is able to present all intents on a map with their connections.
- ***Account limitation:*** Api.ai outperforms IBM Watson, as its platform is completely free. The IBM service has limitations of all kinds such as the duration of the account, the number of queries made or the entities designed.
- ***App integration:*** Another factor to consider is the integration with third-party applications. Api.ai can integrate into services such as Telegram, Twitter, Skype or Google Actions. IBM Watson is designed for integration with its own products, and this integration has to be done manually.

Our main objective with the development of this project is to bring the user the multiple options offered by an intelligent environment on a simplest way. For this reason the

conversation agent has to be dynamic and easy to interact with users. We chose Api.ai option because of its perfect integration using a Google Home device. This voice interface gets more advantages than the web interface designed for the Watson alternative. Besides, its use can be extrapolated to other environments like homes, while web interface limits its use to a working place.

In the following sections we are going to describe carefully the achieved goals, the problems faced and some suggestions for a future work.

6.4 Achieved goals

In the following section I will explain the achieved features and goals that are available in this project.

Make a conversation interface to control smart enviroment This was the main point of our development, creating a voice interface capable of carrying out the conversations needed to interact with the smart environment present on a company workspace.

Automate task using semantic rules Other important objective in our project was to give to the conversations created the power of make automations for the smart devices through semantic rules. For this purpose we got to integrate the conversation interface with the methods for managing rules of Ewetasker system.

Improve the interactions between a user and the smart context This goal looked for giving a new dimension to the interactions occurred in a smart environment. For this reason, apart from being capable of controlling the smart devices, we wanted that non smart objects could participate interacting with user.

Make a visual interface for a robot assistant The last objective was to accomplish that a robot designed as a toy was capable of performing movements and sounds as a personal assistant. With this, we looked for giving our conversations a visual interface.

6.5 Problems faced

During the development of the project we had to face some problems. These problems are briefly described in this section:

Integration of the conversation agent on a device: One of the main problems once raised our system was to choose how to carry out the talks. As we mentioned earlier, we chose Api.ai for its seamless integration into a Google Home device. In addition to this, we solved other problems that emerge when developing the IBM Watson alternative, such as the use of microphones and speakers, making the system less dynamic.

Manage rule functions without using Ewetasker's interface: Another problem that arose in the project was the use of the Ewetasker semantic rules management system without using the interface. For this reason, an analysis of the needs required for the implementation of these functions was carried out. The result was the improvement of Ewetasker and the perfect integration of its new methods in our system.

Control smart devices outside Ewetasker network: Finally the last problem found in the final part of the project was the control of the devices of our intelligent space. This control was designed to be performed from the Ewetasker server or from the mobile application. However when connecting all our devices to a subnet different than the Ewetasker server we had to program a proxy in charge of launching these actions.

6.6 Future work

In the following section I will explain the possible new features or improvements that could be done to the project.

Make conversations to control Ewetasker functions In the project we have approached the use of Ewetasker for the creation of certain rules. However, its possibilities are numerous. For this reason, conversations could be created with the necessary skills to replace the use of the web interface.

Increase interactions with non-intelligent objects Stickers have also been used in this project to achieve interactions with non-intelligent objects in the environment. Nevertheless, this technology can be implemented for the use of many more functions not collected in our study cases.

Give the conversation system the ability to analyze feelings Another of the most outstanding uses of cognitive technologies is the ability to analyse feelings through speech. It would be possible to implement conversations based on the user's feelings and thereby adapt the intelligent environment to their needs.

Bibliography

[1] Api.ai. Api.ai Conversational User Experience Platforms. https://docs.api.ai/.

[2] Moonok Choi, Wan-Ki Park, and Ilwoo Lee. Smart office energy management system using bluetooth low energy based beacons and a mobile app. In *Consumer Electronics (ICCE), 2015 IEEE International Conference on*, pages 501–502. IEEE, 2015.

[3] M. Coronado. EWE Ontology Specification, 2013. http://www.gsi.dit.upm.es/ontologies/ewe.

[4] Miguel Coronado and Carlos A Iglesias. Task automation services: Automation for the masses. *IEEE Internet Computing*, 20(1):52–58, 2016.

[5] Estimote. Estimote Beacons and Stickers. https://estimote.com.

[6] Antonio Fernández Llamas. Design and implementation of a Semantic Task Automation Rule Framework for Android Devices. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, feb 2016.

[7] Luis Ferreira, Antonio Neves, Artur Pereira, Eurico Pedrosa, and Joao Cunha. Human detection and tracking using a kinect camera for an autonomous service robot. *Advances in Aritificial Intelligence-Local Proceedings, EPIA*, pages 276–288, 2013.

[8] Github. Webhooks. https://developer.github.com/webhooks/.

[9] Google. Actions on Google for build apps for the Google Assistant. https://developers.google.com/actions/.

[10] Google. Google Home Smart Speaker. https://madeby.google.com/home/.

[11] IBM. Ibm Watson, the AI platform for business. https://www.ibm.com/watson.

[12] John Kelly III and Steve Hamm. *Smart Machines: IBM Watson and the Era of Cognitive Computing*. Columbia University Press, 2013.

[13] WooWee Labs. Wowwee Mip Bluetooth Low Energy Protocol. https://github.com/WowWeeLabs/MiP-BLE-Protocol.

[14] Marcus Mast, Michael Burmester, Birgit Graf, Florian Weisshardt, Georg Arbeiter, Michal Španěl, Zdeněk Materna, Pavel Smrž, and Gernot Kronreif. Design of the human-robot interaction for a semi-autonomous service robot to assist elderly people. In *Ambient Assisted Living*, pages 15–29. Springer, 2015.

[15] Sergio Muñoz López. Development of a Task Automation Platform for Beacon Enabled Smart Homes. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, jan 2016.

[16] W3C Team. RDF/XML Syntax Specification, 2004. https://www.w3.org/TR/REC-rdf-syntax/.

[17] W3C Team. Time Ontology in OWL, 2017. https://www.w3.org/TR/owl-time.

[18] Vlimit. Experiments with Mip. https://github.com/vlimit/mip.

[19] WooWee. Woowee Mip Robot. http://wowwee.com/mip/.