UNIVERSIDAD DEL PAIS VASCO

Bots Dependientes del Contexto

por

Guillermo Pacho

Memoria de Trabajo de Fin de Grado para el
Grado en Ingeniería Informática de Gestión y Sistemas de Información

en la
Escuela Universitaria de Ingeniería Técnica Industrial

20 de junio de 2017

UNIVERSIDAD DEL PAIS VASCO

Resumen

Escuela Universitaria de Ingeniería Técnica Industrial

Grado en Ingeniería Informática de Gestión y Sistemas de Información

por Guillermo Pacho

En este proyecto se ha desarrollado un *bot* para la aplicación Telegram cuyo objetivo es el de proveer con horarios de transporte público a los usuarios. Para ello, en el *backend* se ha utilizado un *web crawler* que recoge la información y una base de datos *MySQL* para almacenar tanto la información de horarios como la de los chats de Telegram. Asimismo, se han utilizado varias tecnologías, como los servicios *Cognitive Services* de Microsoft y los servicios de *api.ai* para que la interacción usuario-*bot* sea lo mas humana posible.

Contenido

Lista de Figuras

Definiciones

- ***Bot***: La aféresis de robot. Se trata de un *software* que trata de imitar el comportamiento humano. Existen varios tipos de *bots*, como los *crawlers*, *bots* transaccionales, *spammers*... siendo el *chatbot* el que se usa con mayor frecuencia.

- ***Chatbot***: Un tipo de *bot* que se caracteriza por funcionar en el contexto de una conversación. En la mayoría de los casos el *bot* obtendrá la información que necesite preguntándosela al usuario y mostrará las respuestas en un formato amigable para cualquier tipo de usuario. Los *bots* de este tipo tienen una gran presencia en las aplicaciónes de mensajería instantánea.

- **Telegram**: Se trata de una aplicación multiplataforma de mensajería instanánea que ha sido elegida para el desarrollo del *bot* de este proyecto. Telegram ofrece un servicio rápido, seguro y con una API y protocolo abiertos que ofrecen muchas posibilidades para el desarrollo de este tipo de *software*.

- ***Microsoft Cognitive Services***: Este servicio ofrecido por Microsoft ofrece la opción de conversión de audio a texto. A través de una API abierta y cumpliendo con el formato que ésta exige, se puede enviar un audio y recibir como respuesta una conversión de NLP (Procesamiento de Lenguaje Natural).

- ***Api.ai***: Un servicio ofrecido por Google.

- **PHP**: Acrónimo recursivo que significa *PHP Hypertext Preprocessor*, es un lenguaje de programación de propósito general y se utiliza en el lado del servidor. PHP ha evolucionado hasta el punto en que hoy en día pueda ser usado incluso en aplicaciónes gráficas independientes. A pesar de la aparición de nuevas tecnologías como *Node.js* o *ASP.NET*, PHP sigue siendo a día de hoy el lenguaje desplegado en el mayor número de servidores del mundo.

- **JSON**: Acrónimo de *JavaScript Object Notation*, es un formato de texto ligero que proviene de un subconjunto de notación del lenguaje *JavaScript*. JSON es fácil de escribir para humanos y fácil de leer y generar para máquinas y al poder ser interpretado en formato de cadena de texto, ha llegado a ser adoptado como una alternativa al lenguaje de notación XML.

- ***Scraping***: Técnica utilizara para la extracción de datos de páginas web mediante el uso de uno o más programas. Éstos suelen simular el comportamiento de un humano cuando navega por internet.

- ***Web crawler***: Término usado para identificar un grupo de *bots* caracterizado por su utilización en el rastreo de páginas web. También se les suele llamar arañas y su función principal es la de descargar o indexar las páginas web que le son asignadas para el rastreo. Son una herramienta muy potente usada en los motores de búsqueda y en sistemas de búsqueda de *links* rotos pero su versatilidad ofrece un potencial virtualmente ilimitado.

- **Python**: Lenguaje de programación interpretado y multiparadigma con un amplio uso en el desarrollo de todo tipo de *software*. Se puede utilizar para programar mediante programación orientada a objetos, programación imperativa y programación funcional. Python posee una licencia de código abierto y su filosofía consiste en tener una sintaxis que favorezca un código legible.

- ***Scrapy***: Se trata de una potente herramienta para la creación y ejecución de *web crawlers*. *Scrapy* ofrece una detallada documentación online y cuenta con una base importante de programadores que utilizan sus servicios. Los *crawlers* de *Scrapy* se programan en Python.

- ***XPath***: Un lenguaje que expresa la localización de uno o más elementos en un archivo XML. Utilizado para extraer información relevante durante la ejecución del *web crawler* del proyecto.

- ***Framework***: También llamado marco de trabajo, un *framework* es un conjunto estandarizado de conceptos, prácticas, criterios y herramientas para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

- **Ataque de denegación de servicio**: Se denomina así a un tipo de actividad dañina hacia un servidor. Consiste en realizar múltiples solicitudes a una velocidad suficientemente alta como para sobrecargar la máquina y denegar así su capacidad de procesar nuevas consultas.

Capítulo 1

Introducción

Junto con todas las comodidades que nos puede ofrecer la tecnología, los ayudantes personales no son una excepción. El avance en la tecnología de la comunicación está permitiendo que la interacción entre máquina y persona sea cada vez más natural y esto le abre varias puertas a la informática para crear programas cada vez más inteligentes y humanos.

En esta sección se expondrá uno de los grupos de este tipo de programas: los *chatbots*.

1.1. Ayudantes personales

Un ayudante personal (en el contexto de la tecnología de la información) es un robot o *bot*, un *software* diseñado para cumplir con uno o varios propósitos dentro de un contexto predefinido. Se puede implementar un *bot* para que haga virtualmente cualquier cosa, desde tareas simples como que nos recuerden que debemos tomar una medicación siguiendo un horario, a otras no tan simples como que nos encuentren el vuelo más barato disponible a Los Ángeles dentro de dos semanas.

Los *chatbots* son una variante de estos programas orientada a que la interacción con los mismos se haga mediante un chat, simulando una conversación corta que ofrece una experiencia más natural.

1.2. Origen del proyecto

La idea de este proyecto proviene del docente Juan Antonio Pereira, que imparte clases de varias asignaturas en la Escuela de Ingeniería Técnica de Bilbao. Es él quien, desde hace un tiempo, desarrolla y experimenta con distintas aplicaciones usando *bots*. Tiene ideas de ayudantes para su uso personal que le ayudan en sus tareas de docencia, pero también *bots* adecuados para un uso público. En este proyecto se ha desarrollado un *bot* que satisface el segundo caso, es un *chatbot* diseñado para ayudar a encontrar horarios de transporte público, a modo de prueba de concepto para el servicio de trenes de cercanías que ofrece Renfe en la zona de Bilbao.

1.3. Motivaciones para la elección del proyecto

Mi principal motivación a la hora de escoger este proyecto fue la de conocer las diferentes tecnologías que iba a necesitar para llevarse a cabo, herramientas muy diferentes las unas de las otras pero que en conjunto tienen una versatilidad y un potencial enorme.

Conversación con el *bot*

Al ver que se trataba de un *chatbot* y al saber que su objetivo era el de simular una conversación, la palabra "Inteligencia Artificial" me vino a la mente (un campo en el que estoy muy interesado), además, leí que para ello habría que utilizar servicios de procesamiento del lenguaje natural y conversión de audio y la idea pronto atrajo mi atención.

Web crawlers

Juan Antonio me explicó que el proyecto también involucraría el uso de algún método de *scraping* y eso no hizo más que aumentar mi interés. Ya había oído con anterioridad ese término pero no conocía el potencial que tienen este tipo de herramientas. Las posibilidades que ofrecen los *web crawlers* bien implementados son casi ilimitadas y permiten aprovechar casi cualquier web a modo de fuente de datos.

1.4. Caso de uso general

Imaginemos que Ane vive cerca de Abando y que ha encontrado trabajo en Portugalete. Como no tiene coche propio, tiene que buscar la forma de viajar a su lugar de trabajo. Un amigo le ha recomendado utilizar un *bot* de Telegram, ya que usar un *bot* a través de esta aplicación es parecido a hablar con cualquier contacto real.

Para usarlo por primera vez, Ane busca al *bot* por su alias, lo añade a sus contactos de Telegram y lo activa escribiendo el comando */start*. Una vez hecho eso ya puede utilizarlo y Ane prueba a ver si hay horarios para viajes desde Abando hasta Portugalete. El *bot* le encuentra horarios de metro y de tren y ella opta por lo segundo.

Cad vez que Ane necesita consultar los horarios, no tiene más que ir a la conversación de Telegram que tiene con el mismo, escribir el comando */ask* y preguntarle por los horarios con una frase del tipo: "Quiero ir a Portugalete desde Abando".

Un día en que Ane se queda dormida sale corriendo de casa, por lo que al tener mucha prisa no puede escribirle al *bot* para consultar los horarios y ver si le va a dar tiempo. En este caso, simplemente le envía una nota de voz mientras corre hacia la estación y sin ningúna otra complicación, el *bot* le devuelve la lista de horarios que necesita.

Capítulo 2

Planteamiento inicial

Este capítulo comprende la exposición de los objetivos, las herramientas utilizadas, la arquitectura, el alcance y la planificación del proyecto.

2.1. Objetivos

El establecimiento de los objetivos es un paso importante en todo proyecto ya que define las bases sobre las que ir avanzando durante el desarrollo. En esta sección se tratará de enumerar los objetivos iniciales del proyecto con la máxima claridad posible.

Respecto al *bot* de Telegram

- **Robustez de cara al usuario**: Siendo el *bot* de este proyecto un *chatbot*, lo ideal sería que su interacción con el usuario fuese lo más natural posible. Para esto se puede acudir a servicios de procesado de audio y del lenguaje natural que, junto con un diseño robusto del flujo de control de estados del *bot*, posibilite al mismo para que reaccione adecuadamente a la entrada de datos del usuario.

- **Robustez de cara a actualizaciónes futuras**: El proceso de mejora continua y las futuras ampliaciones en la información de la base de datos instan a que con el tiempo, el *bot* sea capaz de ofrecer cada vez más opciones de transporte al usuario. Esto requiere que el *bot* ejecute el comando teniendo este concepto en cuenta y así, las incorporaciones de nuevos servicios de transporte solo afectarán a los *web crawlers* y no al *bot*.

- **Manejo sencillo del comando**: El usuario no debería de invertir demasiado tiempo ni esfuerzo en ver procesada su consulta, para esto se requiere un diseño eficaz que minimice el número de pasos a dar durante su ejecución.

Respecto al *bot* de *Scrapy*

- **Evitar ataque de denegación de servicio**: La potencia que ofrece *scrapy* es tal que si no se controla debidamente puede presentar un riesgo para los dueños de las páginas que son rastreadas. Para conseguirlo, habrá que limitar las consultas por segundo realizadas con los *crawlers*, evitando un posible ataque de denegación de servicio no intencionado.

- ***Web crawler* robusto**: Un problema que a menudo surge a la hora de programar una araña es el hecho de que el contenido de las páginas rastreadas puede cambiar cada vez que se realiza algúna modificación (de diseño, por ejemplo) en las mismas. Para minimizar el efecto que ello podría tener en nuestro rastreo, se deberían utilizar unas expresiones *XPath* eficaces y lo menos específicas posibles (a mayor genericidad, mayor robustez).

Documentación

- **Claridad y simpleza**: La documentación del proyecto y del código deberá presentarse de forma sencilla y clara en caso de que en el futuro la aplicación se expanda o por si algún programador externo deseara realizar cambios por cualquier motivo.

Configuración

- **Archivos de configuración para cada parte del proyecto**: Hay varios parámetros del *bot* y del *crawler* que pueden ser modificados y estos deberían ser fácilmente localizables y modificables. Para ello, se dispondrá de un archivo de configuración para cada parte del proyecto.

2.2. Herramientas utilizadas

En esta sección se muestra un listado de las herramientas utilizadas para el desarrollo del proyecto.

PHP

Uno de los 3 lenguajes principales en los que está desarrollado el proyecto, PHP es un lenguaje de programación que ha sido utilizado para crear todas las funcionalidades del *bot* de Telegram, incluyendo las comunicaciones con los servicios de *Microsoft Cognitive Services* y de *api.ai*.

cURL

Una biblioteca de PHP desarrollada por Daniel Stenberg. Permite la conexión y comunicación con distintos tipos de servidores y para hacerlo, soporta el uso de varios protocolos y certificados.

Python

El segundo de los lenguajes principales del proyecto. Python se ha usado para codificar el *web crawler* y también para los archivos de testeo del *bot* de Telegram.

Scrapy

El *framework* que se ha utilizado para crear el *web crawler*, scrapy es una herramienta con una versatilidad enorme, desarrollada en Python y bien documentada en su página web.

PHP longman telegram-bot

Una de las bibliotecas principales que se usan para desarrollar *bots* de Telegram. Contiene las funcionalidades básicas de un *bot* y está preparado para ser extendido con cualquier funcionalidad nueva, lo que lo convierte en la base principal sobre la que está desarrollado el proyecto.

Composer

Paquete utilizado para gestionar dependencias en PHP. Se configura a través de un archivo JSON para instalar y mantener actualizadas las bibliotecas de un proyecto. El objetivo de usar este paquete es mantener actualizadas todas las bibliotecas usadas por el *bot* de Telegram.

API de *Microsoft Cognitive Services*

Microsoft ofrece un potente paquete de servicios en relación a la interacción usuario-máquina. Este proyecto utiliza el servicio *Microsoft Cognitive Services*, para convertir archivos de audio a texto.

API de *api.ai*

API que ofrece crear lo que llama un "agente", al que se le deben asignar unas reglas de tratamiento de texto y un entrenamiento para que finalmente tenga la capacidad de extraer la intención de un texto (*intent*) e incluso distinguir la información clave o relevante para la intención detectada.

FFmpeg

Herramienta de conversión y editado de archivos de audio y vídeo. Permite codificar, decodificar, multiplexar, demultiplexar, filtrar, reproducir, etc, prácticamente todos los formatos de audio y vídeo conocidos.

Image Magick

Un paquete de PHP que permite la creación desde cero de varios tipos de archivos de imagen. Permite crear efectos, aplicar filtros e incluso crear archivos GIF.

YAML

Se trata de un formato de serialización de datos legible para el ser humano y en este proyecto, se utiliza para crear los archivos de configuración que contienen tanto la configuración para el entorno de desarrollo como el de producción.

SQL

El tercero, aunque no menos importante lenguaje utilizado para el desarrollo del proyecto. A pesar de que su presencia en el código sea menor, es la herramienta utilizada para intercomunicar las distintas partes del proyecto a través de la base de datos.

MySQL

El sistema de gestión de bases de datos utilizado en este proyecto. MySQL ha permitido la creación de las tablas y la gestión de los datos guardados en el servidor. Aparte de las tablas obtenidas del paquete de PHP longman telegram-bot, ha hecho falta la creación de tablas nuevas para gestionar el almacenamiento de los horarios.

Git

Software de control de versiones que aporta una mayor comodidad a la hora de actualizar y mantener el código de cualquier proyecto. La ayuda que ofrece Git marca en ocasiones una diferencia tremenda en lo que a mantenimiento y supervisión de los archivos fuente de un proyecto se refiere.

Atom

El editor de texto utilizado en este proyecto. Entre las características destacables de Atom se encuentras su rapidez, el sistema de instalación de paquetes de que dispone, su diseño claro y atractivo y su integración con los servicios de Git. Muestra en tiempo real el estado de las ramas de los proyectos de Git que se hayan abierto.

Bitbucket

Servicio de guardado de proyectos de Git y Mercurial. Bitbucket ofrece crear y guardar un ilimitado numero de proyectos, es rápido y se puede navegar por su interfaz de forma intuitiva. Se ha elegido Bitbucket en lugar del más popular Github para poder hacer uso de repositorios privados de forma gratuita.

Sharelatex

Para redactar la memoria, se ha hecho uso del sistema de creación y desarrollo de proyectos en LaTeX que ofrece la página sharelatex.com.

Cacoo

Herramienta para la creación de diagramas utilizada para generar muchas de las figuras insertadas en la memoria del proyecto. Permite la creación de todo tipo de esquemas y diagramas, guardándolos en la nube y pudiendo acceder a ella desde cualquier dispositivo conectado a la red.

2.3. Arquitectura

En esta sección se muestra lo referente a la arquitectura del *bot* de Telegram, su funcionamiento y la del comando desarrollado, así como la arquitectura del *web crawler* creado con Scrapy.

Bot de Telegram

Los requisitos para hacer funcionar un *bot* de Telegram básico son disponer de un servidor con Apache2 instalado y una de las bibliotecas disponibles para su desarrollo. El *bot* se puede configurar para que gestione las solicitudes de los usuarios por medio de dos enfoques distintos: *polling* y *push*. El primero se basa en que el *bot* realice consultas constantemente para ver si el usuario ha enviado alguna petición, generando un hilo de comunicación constante con el éste. Con el segundo, el *bot* y el usuario establecen una conexión activa y cuando el usuario realiza alguna consulta, "avisa" al *bot* para que éste la procese en cuanto la reciba. Dependiendo del tipo de *bot* que se quiera, habría que tener una base de datos MySQL y/o un certificado SSL.

En este proyecto se utilizará un servidor con el paquete de desarrollo de *bots* PHP *longman telegram-bot* (el *bot* estará programado en PHP) y el manejo de las peticiones al servidor se hará mediante un *webhook*. También se hará uso de una base de datos MySQL para guardar la información relativa a los chats (en lo que al *bot* se refiere).

A continuación, en la figura 2.1 se detalla un esquema del funcionamiento del *bot* de Telegram.

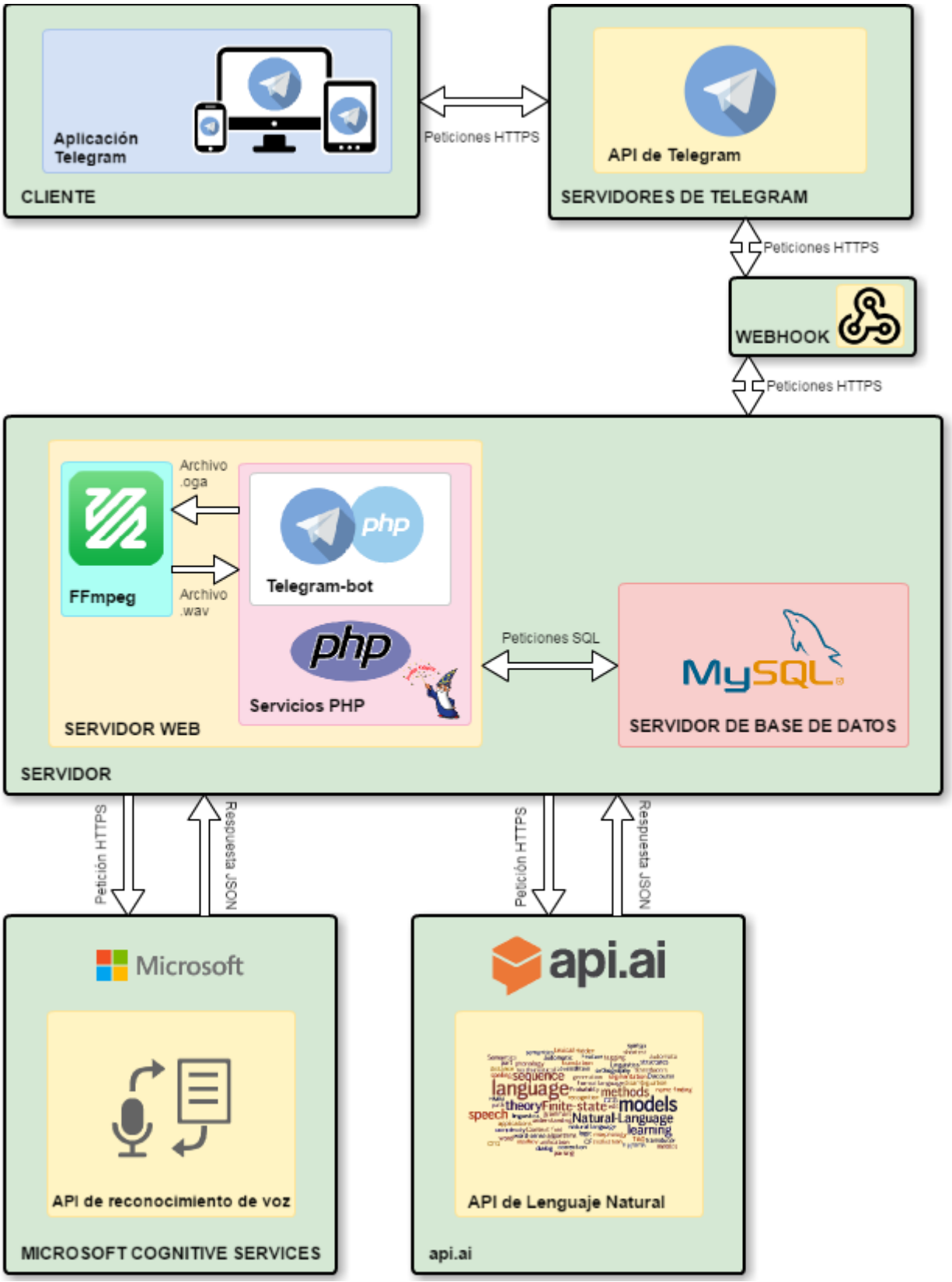

FIGURA 2.1: Esquema de la arquitectura de un bot de Telegram.

Los mensajes que recibe el *bot* desde cualquier dispositivo, pasan primero por los servidores de Telegram. El servidor detecta que el *bot* está registrado a un *webhook*, por lo que manda el mensaje al servidor donde éste reside, y así obtiene la respuesta que corresponda devolver.

Los mensajes son enviados mediante el protocolo HTTPS que asegura la encriptación de los mismos y aporta un canal apropiado para este tipo de información sensible. Adicionalmente, Telegram utiliza su propio protocolo para la encriptación interna de los mensajes, el protocolo MTProto.

Cuando la petición de Telegram llega al servidor que custodia al *bot*, el intérprete de PHP delega la gestión al paquete *longman telegram-bot*. Una vez en este punto, el paquete trata el mensaje y busca la presencia de algún comando o utiliza los servicios de *Microsoft Cognitive Services* y de *api.ai* para extraer la información relevante. En este paso se necesita usar una herramienta de manipulación de audio y vídeo llamada FFmpeg con el fin de evitar un problema de compatibilidad con los archivos de audio.

Microsoft Cognitive Services es utilizado en caso de recibir una nota de voz. Su propósito es transformar la nota de voz en texto. Cuando el *bot* recibe texto, se utiliza el servicio de *api.ai* para extraer una intención (*intent*) y las palabras clave necesarias para procesar la petición.

Finalmente, cuando la la biblioteca termina de procesar el mensaje, genera una respuesta utilizando la biblioteca de PHP *Image Magick* que devuelve al servidor de Telegram y éste a su vez, envía al usuario que dió comienzo a todo el proceso.

La utilización de FFmpeg para solucionar el problema de compatibilidad, así como el uso de *ImageMagick* para la muestra de resultados, son posteriores al planteamiento inicial ya que surgieron por una necesidad o como una idea durante el desarrollo. Aun así, se incluyen en este apartado por la suma importancia que tienen en el proyecto y se discutirá su inclusión más adelante.

Web crawler de Scrapy

El funcionamiento del comando */ask* requiere de una base de datos con información actualizada de los horarios de Renfe. La obtención de dicha información se consigue mediante un *web crawler*, un *bot* diseñado para rastrear la página o páginas web indicadas y obtener de estas la información deseada.

La figura 2.2 muestra el esquema del funcionamiento del *bot* de Scrapy.

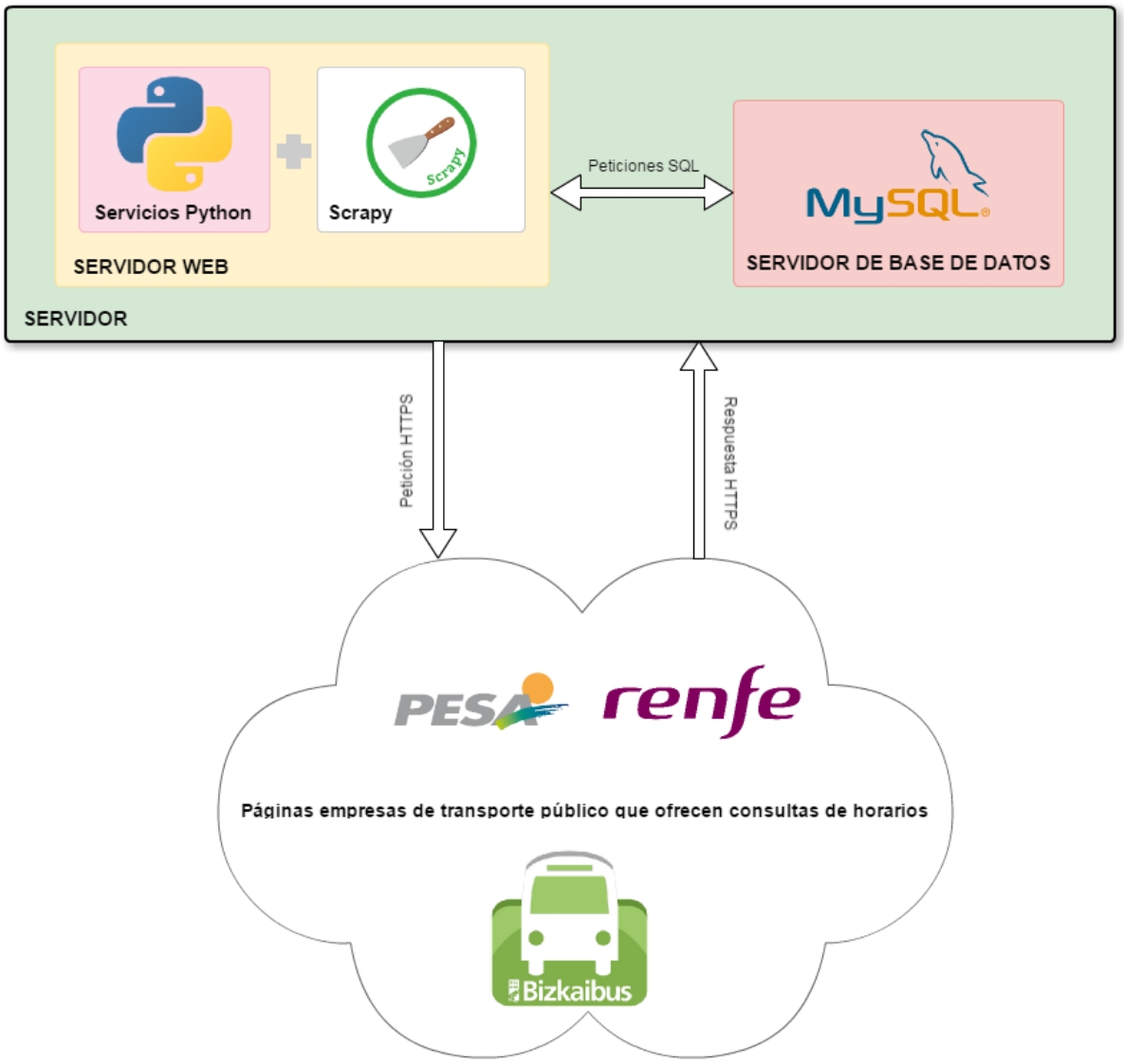

FIGURA 2.2: Esquema de la arquitectura del *web crawler*.

Scrapy ofrece un sistema de creación de arañas bastante sencillo, basta con instalar Scrapy y crear un archivo *.py* en la carpeta *spiders*. Acto seguido se puede crear una clase que extiende a una clase nativa de Scrapy: la clase *Spider*. Este archivo se encargará de descargar las páginas deseadas y de procesarlas utilizando el método *parse* que habrá que definir.

Scrapy funciona con una pila que inicialmente contendrá las páginas que se quieran procesar primero. A su vez, estas direcciones iniciales pueden ser exploradas para buscar nuevas direcciones que podrán ser añadidas a la pila. El método *parse* que se defina simplemente descargará las páginas que se hayan añadido a la pila y las procesará utilizando expresiones *XPath* para extraer la información relevante de forma rápida y precisa.

2.4. Alcance

El alcance del proyecto define la totalidad del trabajo necesario para terminar el proyecto, en esta sección se han separado los distintos requisitos encontrados y han sido enumerados.

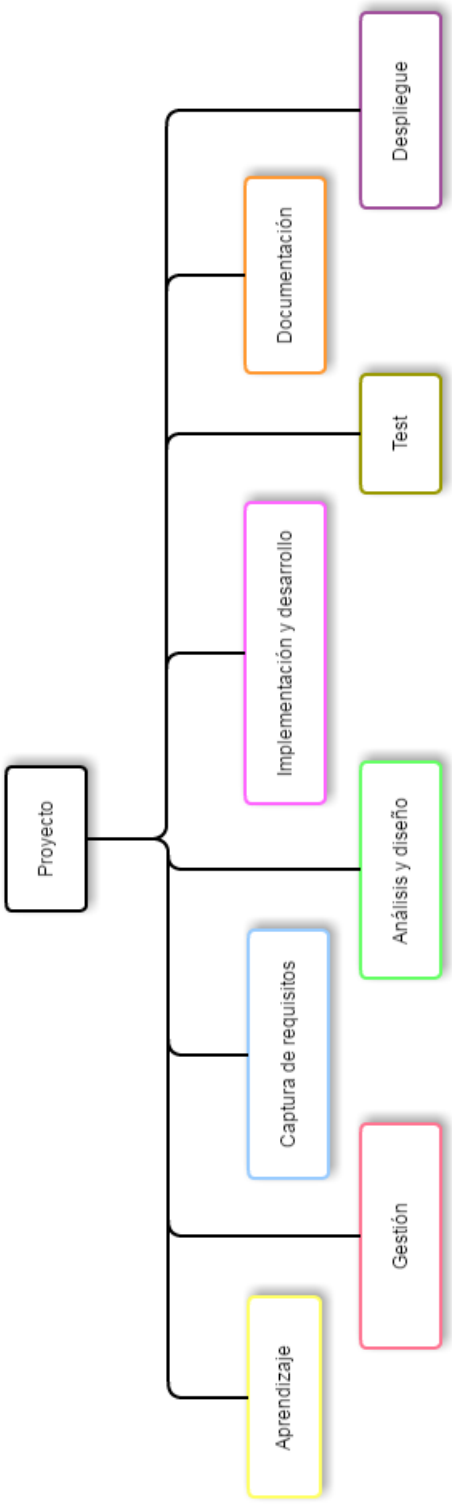

FIGURA 2.3: Diagrama EDT del proyecto.

A excepción del aprendizaje, no hay una secuencia concreta en la que se hayan realizado las tareas.

2.4.1. Aprendizaje

La tarea del aprendizaje contempla los intervalos de estudio por los que se ha tenido que pasar para poder manejar de forma eficaz las herramientas utilizadas en el desarrollo del proyecto.

A continuación se muestra un diagrama EDT con los puntos que componen la tarea del aprendizaje en este proyecto.

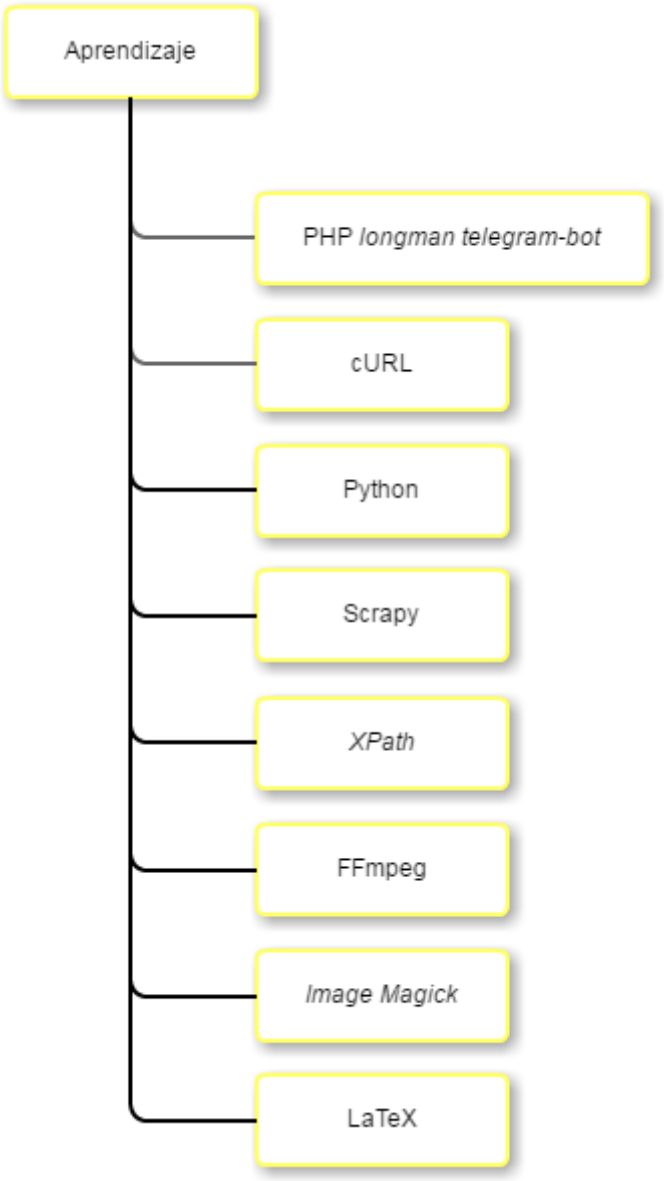

FIGURA 2.4: Diagrama EDT de la tarea del aprendizaje.

Junto al diagrama de la figura 2.4, estas tablas desglosan cada apartado del aprendizaje para obtener una visión más completa.

Aprendizaje del paquete PHP longman telegram-bot	
Duración estimada	20 horas
Descripción	Aprender a utilizar los métodos y el funcionamiento del paquete para poder desarrollar comandos personalizados.
Recursos necesarios	PHP, biblioteca *longman telegram-bot*, Atom, Navegador Web.
Aprender a hacer consultas usando cURL	
Duración estimada	10 horas
Descripción	Aprender a realizar peticiones HTTP utilizando la biblioteca cURL integrada con PHP.
Recursos necesarios	Navegador Web, Atom, PHP.
Aprendizaje del lenguaje Python	
Duración estimada	30 horas
Descripción	Aprender a programar en Python, adquirir nociones de estructuración, uso de algoritmos básicos y funciones nativas del lenguaje.
Recursos necesarios	Navegador Web, Atom, Python.
Aprendizaje del uso de Scrapy	
Duración estimada	20 horas
Descripción	Aprender a crear *bots* de Scrapy, a rastrear páginas y a tratarlas para obtener datos útiles.
Recursos necesarios	Navegador Web, Atom, Python, Scrapy.
Aprendizaje del uso de las expresiónes *XPath*	
Duración estimada	10 horas
Descripción	Aprender a utilizar las expresiones *XPath* para obtener los objetos deseados dentro de un documento HTML.
Recursos necesarios	Navegador Google Chrome y su consola JavaScript.

Aprendizaje del uso de la biblioteca FFmpeg	
Duración estimada	2 horas
Descripción	Aprender a utilizar las funciones de la biblioteca FFmpeg con el fin de arreglar un problema de compatibilidad durante la ejecución del comando */ask*.
Recursos necesarios	Navegador Web, PHP, FFmpeg.
Aprendizaje del uso del paquete de PHP *Image Magick*	
Duración estimada	10 horas
Descripción	Aprender a utilizar las funciones de la biblioteca *Image Magick* con el fin de implementar la generación de imágenes con el *bot* de Telegram.
Recursos necesarios	Navegador Web, PHP, *Image Magick*.
Aprendizaje del lenguaje LaTeX	
Duración estimada	15 horas
Descripción	Aprender a crear documentos bien estructurados usando el lenguaje LaTeX.
Recursos necesarios	Navegador Web y cuenta en Sharelatex para hacer pruebas.

2.4.2. Gestión

La organización es el proceso de ordenado y secuenciación de las partes del proyecto, con el objetivo de que se realicen de forma eficaz.

El diagrama EDT de la figura 2.5 muestra los puntos que componen la tarea de la organización.

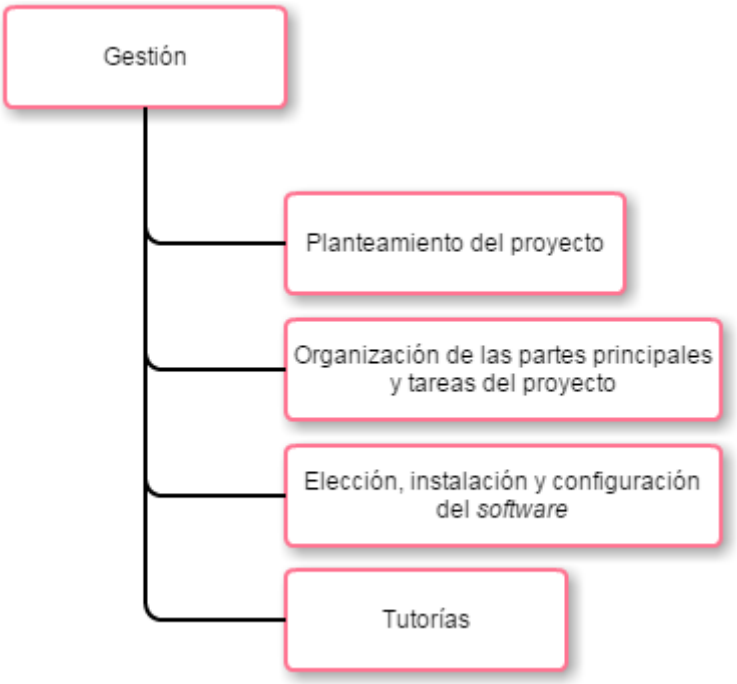

FIGURA 2.5: Diagrama EDT de la tarea de la gestión.

A continuación, las tablas que desglosan los apartados de la gestión.

Planteamiento del proyecto	
Duración estimada	5 horas
Descripción	Plantear un esquema general del proyecto, valorando la viabilidad del desarrollo planteado basándose en las tecnologías disponibles.
Recursos necesarios	Navegaro Web, papel y lápiz.

Organización de las partes principales y tareas del proyecto	
Duración estimada	5 horas
Descripción	Realizar una esbozo base de las distintas partes que componen el proyecto y de las tareas que habrá que llevar a cabo para desarrollarlas.
Recursos necesarios	Papel y lápiz.

Elección, instalación y configuración del *software*	
Duración estimada	7 horas
Descripción	Elección del *software* apropiado para el desarrollo y la instalación y configuración de dicho *software*.
Recursos necesarios	Navegador Web, *software* varios (los elegidos).
Tutorías	
Duración estimada	20 horas
Descripción	Reunirse con el tutor para establecer las bases del proyecto, para realizar consultas informativas y pedir consejos de cualquier tipo.
Recursos necesarios	Papel y lápiz.

2.4.3. Captura de requisitos

Este apartado recoge la captura de los requisitos que se habrán que cumplir utilizando las herramientas elegidas durante el desarrollo del proyecto.

El diagrama EDT muestra los puntos que componen la tarea de la captura de requisitos.

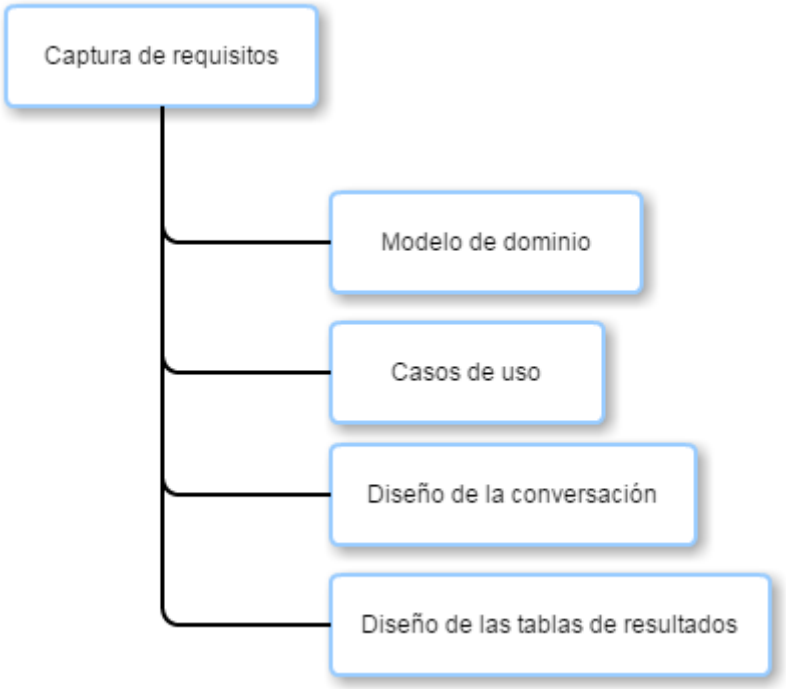

FIGURA 2.6: Diagrama EDT de la tarea de la captura de requisitos.

A continuación, las tablas que desglosan los apartados de la captura de requisitos.

Casos de uso	
Duración estimada	2 horas
Descripción	Enumerar los casos de uso para los procesos que se lleven al cabo al utilizar la aplicación del proyecto.
Recursos necesarios	Papel y lápiz.
Diseño de la conversación	
Duración estimada	8 horas
Descripción	Realizar un esbozo del flujo de la conversación que hay que realizar para la ejecución del comando */ask*.
Recursos necesarios	Papel y lápiz.
Diseño de las tablas de resultados	
Duración estimada	1 hora
Descripción	Diseñar una forma de mostrar los horarios lo más clara y sencillamente posible.
Recursos necesarios	Papel y lápiz.
Modelo de dominio	
Duración estimada	5 horas
Descripción	Diseñar el modelo de dominio que mejor se adapte al proyecto.
Recursos necesarios	Navegador Web, papel y lápiz.

2.4.4. Análisis y diseño

En este apartado, se detalla la planificación a la hora de analizar el proyecto y diseñar los distintos diagramas que se requieren.

El diagrama EDT muestra los puntos que componen la tarea del análisis y diseño.

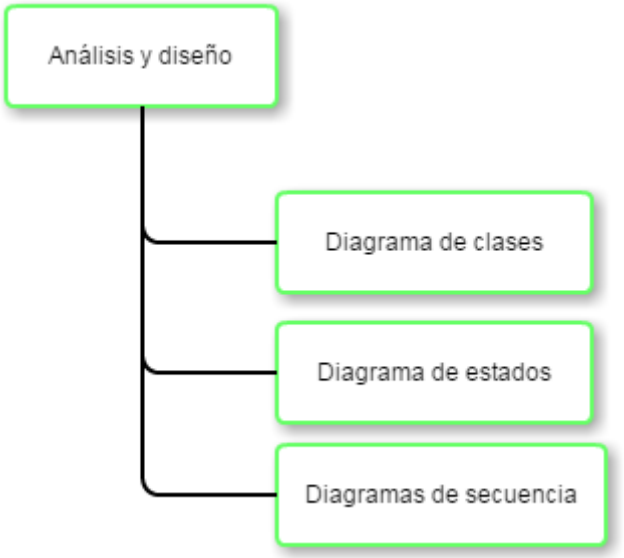

FIGURA 2.7: Diagrama EDT de la tarea del análisis y diseño.

A continuación, las tablas que desglosan el apartado.

Diagrama de estados	
Duración estimada	5 horas
Descripción	Diseño y construcción del diagrama de estados que atañe al proyecto.
Recursos necesarios	Papel y lápiz.

Diagrama de clases	
Duración estimada	5 horas
Descripción	Diseño y construcción del diagrama de clases que atañe al proyecto.
Recursos necesarios	Papel y lápiz.

Diagramas de secuencia	
Duración estimada	8 horas
Descripción	Diseño y construcción de los diagramas de secuencia que desglosan la secuencia de ejecución del *bot*.
Recursos necesarios	Papel y lápiz.

2.4.5. Implementación y desarrollo

Este apartado contiene los detalles sobre la implementación y el desarrollo del proyecto.

El diagrama EDT muestra los puntos que componen la tarea de la implementación y desarrollo.

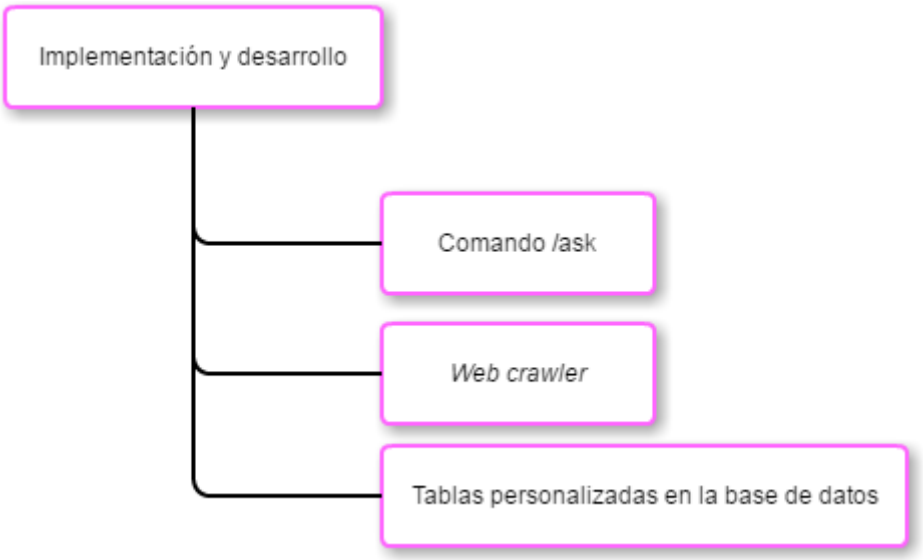

FIGURA 2.8: Diagrama EDT de la tarea de la implementación y desarrollo.

Se han dividido las tablas en 2 sub-apartados, a saber, la implementación y desarrollo del *bot* de Telegram y la implementación y desarrollo del *web crawler*.

Implementación y desarrollo del *bot* de Telegram

A continuación, las tablas que desglosan el desarrollo e implementación del *bot* de Telegram.

Comando */ask*	
Duración estimada	60 horas
Descripción	Implementación y desarrollo del comando */ask* que engloba el uso de gran parte de las tecnologías utilizadas en el proyecto.
Recursos necesarios	PHP, biblioteca *longman telegram-bot*, base de datos con información útil, cuentas en las páginas de los proveedores de los servicios de *Microsoft Cognitive Services* y *api.ai*, FFmpeg, *Image Magick*, MySQL, Atom.

Implementación y desarrollo del *web crawler*

A continuación, las tablas que desglosan el desarrollo e implementación del *web crawler*.

Tablas personalizadas en la base de datos	
Duración estimada	2 horas
Descripción	Implementación de las tablas personalizadas en la base de datos.
Recursos necesarios	MySQL.
Web crawler	
Duración estimada	30 horas
Descripción	Implementación y desarrollo del *web crawler*, verificando que las páginas rastreadas no cambian con frecuencia y que las expresiones *XPath* son robustas.
Recursos necesarios	Python, Scrapy, MySQL.

2.4.6. Test

Este apartado contiene los detalles sobre los test realizados con las distintas partes del proyecto para verificar su funcionamiento y robustez.

El diagrama EDT muestra los puntos que componen la tarea de la realización de test.

FIGURA 2.9: Diagrama EDT de la tarea del testeo.

A continuación, las tablas que desglosan los apartados de la fase de testeo.

Test con el *bot* de Telegram	
Duración estimada	20 horas
Descripción	Test realizados al *bot* de Telegram durante el desarrollo para asegurar el funcionamiento correcto y detectar posibles errores de cualquier tipo.
Recursos necesarios	Aplicación de Telegram de escritorio y de *Smartphone*, PHP, biblioteca *longman telegram-bot*, base de datos con información útil, cuentas en las páginas de los proveedores de los servicios de *Microsoft Cognitive Services* y *api.ai*, FFmpeg, *Image Magick*, MySQL, Atom.

Test con la consola de *tg-cli* y el *wrapper Python-tg*	
Duración estimada	10 horas
Descripción	Preparación y prueba de una serie de casos de uso, que mediante *wrapper Python-tg* y *tg-cli* se comunican con el *bot* automáticamente y comprueban que se comporta de la forma esperada.
Recursos necesarios	Python, *tg-cli*, *wrapper Python-tg*, Aplicación de Telegram de escritorio y de *Smartphone*, PHP, biblioteca *longman telegram-bot*, base de datos con información útil, cuentas en las páginas de los proveedores de los servicios de *Microsoft Cognitive Services* y *api.ai*, FFmpeg, *Image Magick*, MySQL, Atom.
Test con el *web crawler*	
Duración estimada	10 horas
Descripción	Test realizados al *web crawler* para asegurar el funcionamiento correcto y detectar posibles errores de cualquier tipo.
Recursos necesarios	Python, Scrapy, MySQL.

2.4.7. Documentación

Aquí de desglosan las tareas de documentación del proyecto.

El diagrama EDT muestra los puntos que componen la tarea de la documentación.

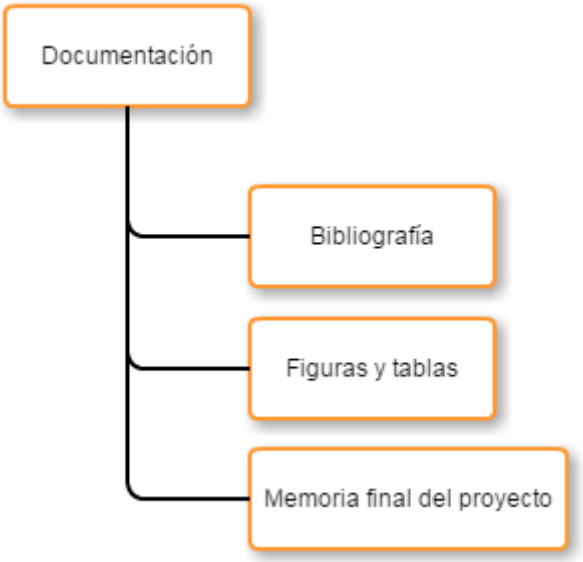

FIGURA 2.10: Diagrama EDT de la tarea de la documentación.

A continuación, las tablas que desglosan los apartados de la fase de documentación.

Bibliografía	
Duración estimada	5 horas
Descripción	Búsqueda de referencias al material utilizado a la hora de redactar la memoria final del proyecto.
Recursos necesarios	Navegador web, editor de LaTeX.

Figuras y tablas	
Duración estimada	15 horas
Descripción	Creación de las figuras y tablas utilizadas en la memoria final del proyecto.
Recursos necesarios	Navegador web, editor de LaTeX, Cacoo.

Memoria final de proyecto	
Duración estimada	32 horas
Descripción	Redacción de la memoria final del proyecto. Se trata de la organización y cristalización de las ideas y desarrollos realizados en un documento final.
Recursos necesarios	Navegador web, editor de LaTeX.

2.4.8. Despliegue

Aquí de desglosan las tareas de despliegue del proyecto.

El diagrama EDT muestra los puntos que componen la tarea del despliegue.

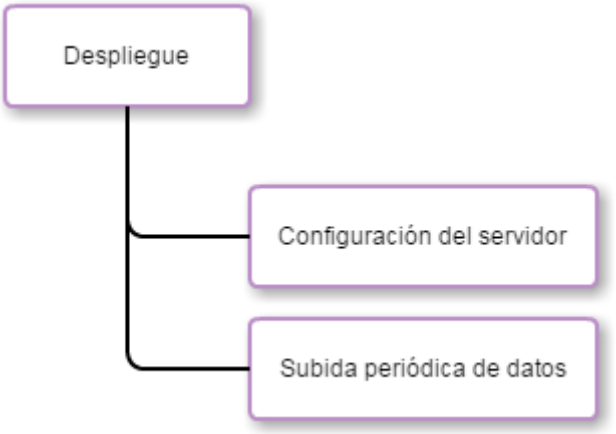

FIGURA 2.11: Diagrama EDT de la tarea del despliegue.

A continuación, las tablas que desglosan los apartados de la fase de despliegue.

Configuración del servidor de producción	
Duración estimada	5 horas
Descripción	Instalación y configuración del *software* utilizado en el servidor de producción.
Recursos necesarios	Conexión a *bots.ikasten.io* (servidor).
Subida periódica de datos	
Duración estimada	2 horas
Descripción	Subida periódica del código y base de datos al servidor de producción.
Recursos necesarios	Conexión a *bots.ikasten.io* (servidor).

2.5. Planificación temporal

En la figura 2.12 se muestran las dependencias y duraciones de las tareas previamente mencionadas, que más tarde servirán para la creación de un diagrama Gantt que muestre de forma clara la planificación del proyecto.

ID	Tarea	Predecesor	Tiempo est.
	Aprendizaje		**117**
1	Paquete *PHP longman telegram-bot*	-	20
2	Consultas con cURL	-	10
3	Python	-	30
4	Scrapy	3	20
5	Expresiones *XPath*	-	10
6	FFmpeg	-	2
7	*Image Magick*	-	10
8	LaTeX	-	15
	Gestión		**37**
9	Planteamiento del proyecto	-	5
10	Organización de las partes principales tareas del proye	9	5
11	Elección, instalación y configuración del *software*	10	7
12	Tutorías	-	20
	Captura de requisitos		**16**
13	Casos de uso	11	2
14	Diseño de la conversación	13	8
15	Diseño de las tablas de resultados	14	1
16	Modelo de dominio	14	5
	Análisis y diseño		**18**
17	Diagrama de estados	14	5
18	Diagrama de clases	16	5
19	Diagrama de secuencia	17	8
	Implementacióny desarrollo		**92**
20	Comando /ask	1,2	60
21	Tablas personalizadas en la BD	18	2
22	*Web crawler*	3,4,5,21	30
	Test		**40**
23	Test con el bot de Telegram	20	20
24	Test con la consola de *tg-cli* y el *wrapper Python-tg*	20,23	10
25	Test con el *web crawler*	22	10
	Documentación		**52**
26	Bibliografía	-	5
27	Figuras y tablas	20,21,22	15
28	Memoria final del proyecto	23,24,25,26,27	32
	Despliegue		**7**
29	Configuración del servidor de producción	11	5
30	Subida periódica de datos	-	2
	TOTAL		**379**

FIGURA 2.12: Resumem del alcance temporal.

Una vez que se tiene una visión general de las tareas del proyecto agrupadas en divisiones de fácil distinción, se procederá a la construcción de un diagrama de Gantt para la distribución de dichas tareas en el tiempo, dentro de los límites de inicio y final de la realización del proyecto.

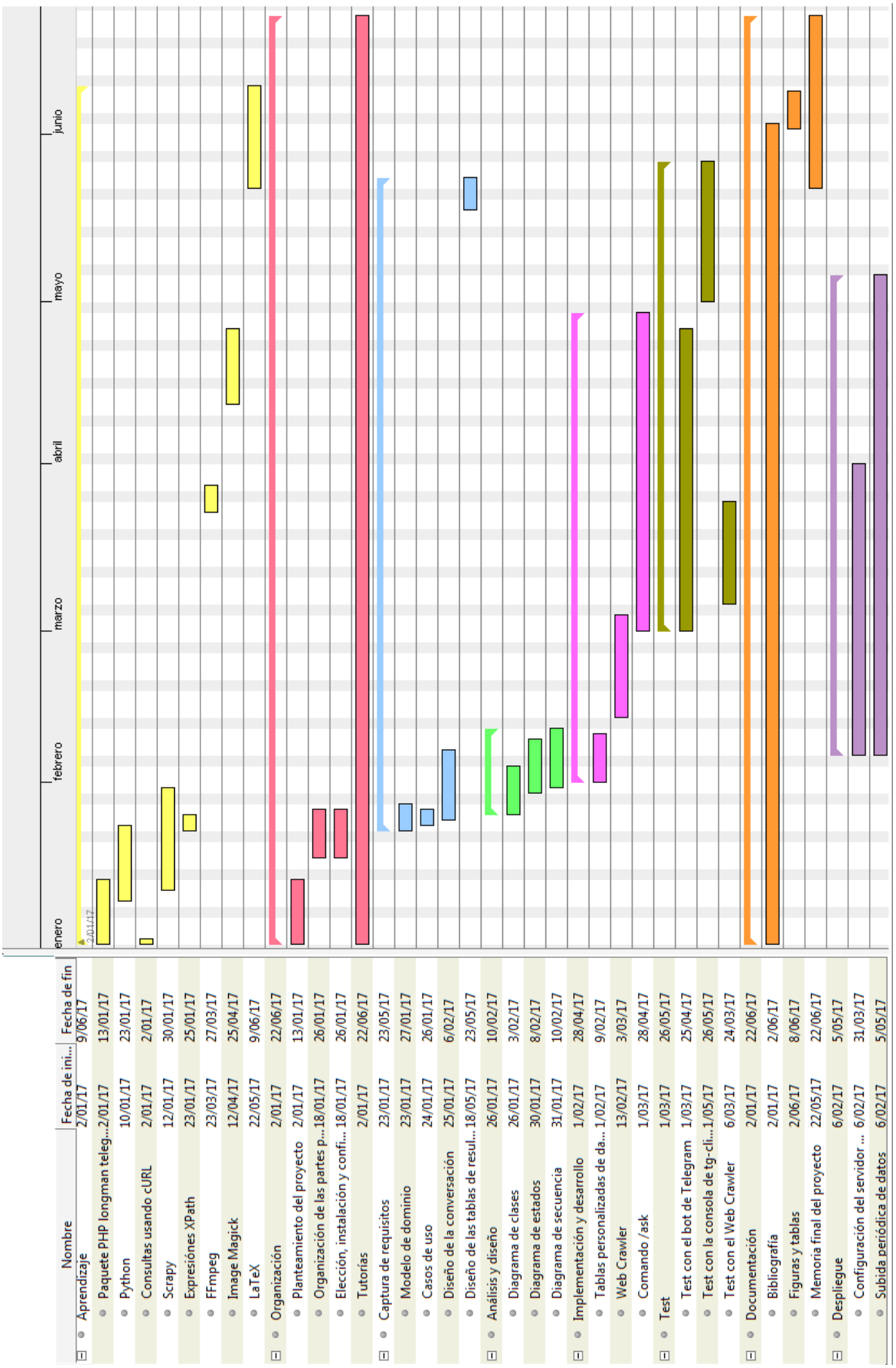

FIGURA 2.13: Diagrama de Gantt.

Como puede apreciarse en la Figura 2.12, la suma total de horas de trabajo requeridas para completar este proyecto es de 379 horas. La fecha de inicio del proyecto está marcada a 1 de Enero de 2017 y la de finalización a 22 de Junio del mismo año, por lo que suponiendo una jornada laboral de 20 horas semanales:

$$Meses * Semanas * Horas\ semanales = 6{,}6 * 4 * 20 = 528\ horas \tag{2.1}$$

A las que restando una estimación de 50 horas en fines de semana, festivos y demás posibles acontecimientos:

$$528 - 50 = 478\ horas \tag{2.2}$$

Si se compara el resultado con la suma total de horas obtenida en la tabla del alcance temporal, se obtiene que hay tiempo suficiente para completar el proyecto en el tiempo estipulado.

Capítulo 3

Captura de requisitos

En este capítulo se detallan los requisitos que se deben cumplir con respecto al *bot* de Telegram y al *web crawler* con el objetivo de conseguir la máxima satisfacción por parte del usuario proporcionando facilidad de uso y simpleza a la hora de mostrar resultados.

3.1. Jerarquía de actores

El *bot* de este proyecto tiene como objetivo ser un *bot* de ámbito público, por lo que en la jerarquía de actores se podrá distinguir un único tipo de usuario. Sin embargo, el paquete de *longman telegram-bot* distingue dos tipos de usuarios, el usuario normal y el administrador, por lo que la jerarquía de actores de este proyecto contemplará a ambos tipos de usuarios.

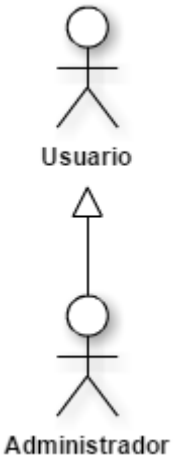

FIGURA 3.1: Jerarquía de actores.

3.2. Casos de uso

Los casos de uso de la figura 3.2 muestran las distintas acciones que puede realizar un usuario. Se distinguirán dos comandos: el comando */ask* y el comando */cancel*.

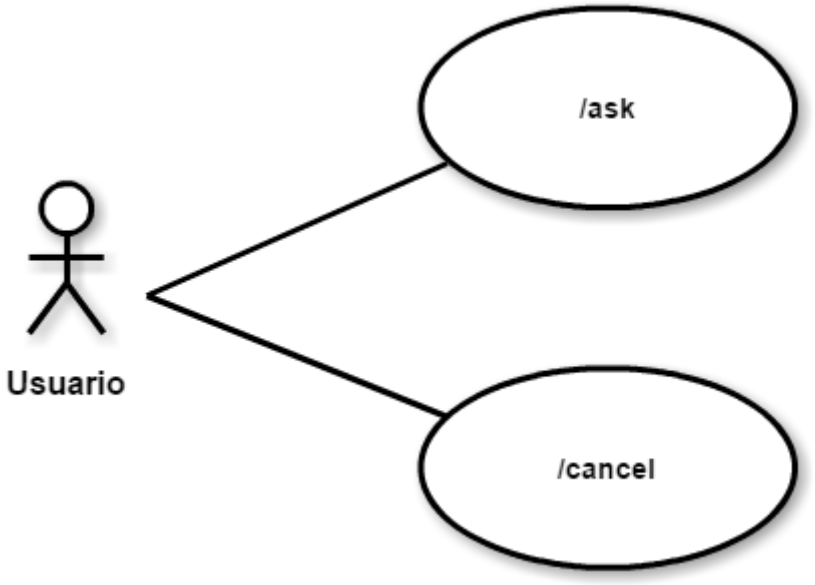

FIGURA 3.2: Casos de uso.

- ***/ask*:** Al ejecutar este comando se iniciará una conversación con el *bot*, el cual tratará de obtener los datos necesarios para realizar una consulta de horarios. El *bot* comenzará pidiendo al usuario el origen y destino de su consulta, sugiriéndole que puede tanto escribirlos como comunicarlos por medio de una nota de voz. La interacción con este *bot* pretende utilizar un lenguaje natural en lugar de rellenar un formulario. Esto facilita el uso del *bot* a los usuarios de cualquier edad y también proporciona velocidad en caso de no estar en un buen momento para rellenar campos de texto en un formulario.

- ***/cancel*:** La ejecución de este comando implica la cancelación de cualquier otro comando que pudiera estar en curso. Aunque el comando */ask* va a tener pocos estados en la mayoría de los casos, la existencia de una comando para cancelar toda actividad es siempre útil.

3.3. Diseño del *web crawler*

En lo referente al *web crawler*, se ha diseñado para que la información de los horarios se haga de forma poco agresiva para las páginas web objetivo, configurando un retardo de 1 segundo entre cada petición realizada. Las expresiónes *XPath* se han diseñado dándole importancia a la robustez en la medida de lo posible, para que en caso de que haya modificaciones en el código fuente de las páginas, se minimice el impacto en los datos recogidos (aunque si hay cambios en la estructura general o los cambios son muy grandes, es inevitable tener que reajustar los *spider*).

3.4. Diseño de los resultados

Cuando el *bot* haya conseguido todos los datos necesarios para una consulta y ésta sea una consulta válida, éste enviará una imagen generada mediante la biblioteca de PHP *Image Magick* (o *Imagick*) con una disposición de los datos en forma de tabla. El diseño intenta parecerse en lo máximo a las tablas de horarios de la mayoría de empresas de transporte público. El diseño inicial ha sido realizado a bolígrafo en una hoja y más tarde implementado y sometido a varios ajustes para una mayor claridad.

Existen dos tipos de tablas: tablas de viajes lineales y tablas de viajes con transbordo. El diseño realizado en papel se corresponde únicamente con el primer tipo, mientras que el segundo ha surgido añadiendo más campos de datos y un campo de cabecera al mismo.

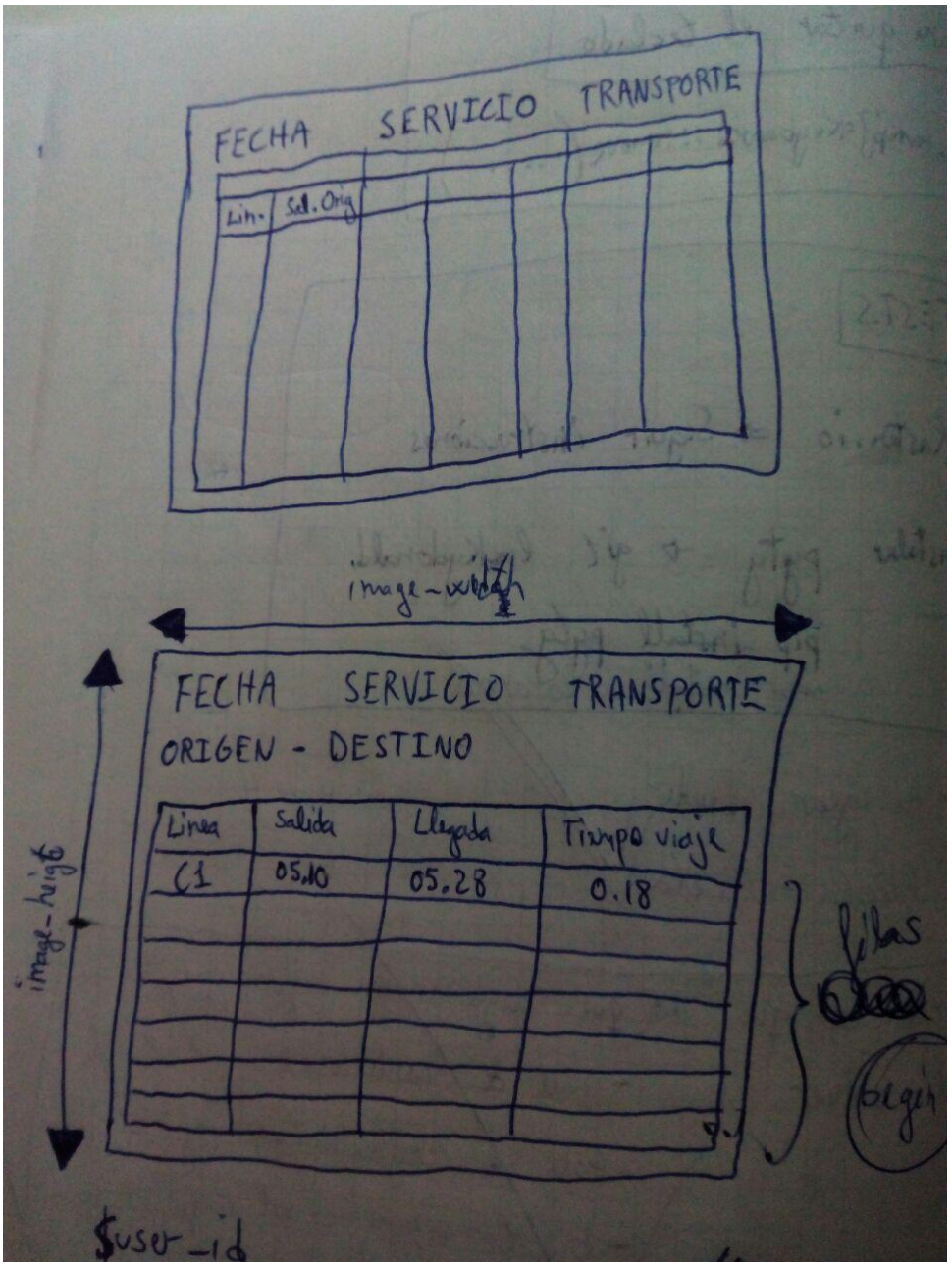

FIGURA 3.3: Prototipo sencillo de la tabla de resultados sin transbordo.

3.5. Modelo de dominio

El modelo de dominio se utiliza para la representación conceptual de las entidades de un proyecto con sus atributos, y mostrando las relaciones entre éstas. En la figura 3.4 se muestra el modelo de dominio del proyecto.

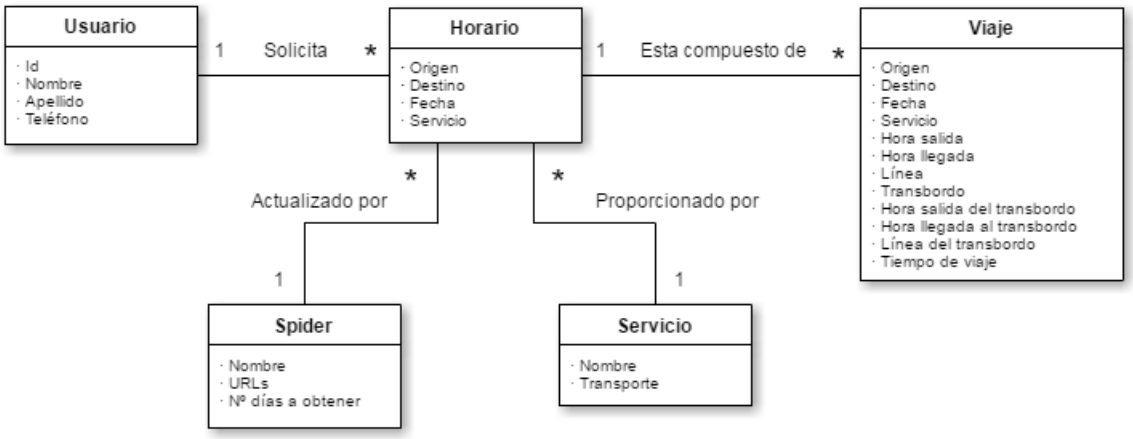

FIGURA 3.4: Modelo de dominio del proyecto.

Ahora se describirán las entidades del modelo de dominio para una mayor comprensión del esquema:

Usuario

Todo aquel cliente de Telegram que utiliza el *bot*. Los comandos que puede usar le permiten solicitar horarios de transporte público y contiene información relativa a su cuenta de Telegram, tales como el nombre, apellido, numero de teléfono y un identificador único que lo diferencia de los demás usuarios.

Horario

Una agrupación de datos referentes a lo que se asocia conceptualmente como un horario. La información que contiene lo describe y debe tener un origen, un destino y una fecha. Se ha añadido una dato llamado servicio que hace referencia a la compañía que proporciona dicho horario, la cual estará asociada a un medio de transporte público.

Servicio

Cada horario está asociado a una compañía de tansporte público de la cual se habrá obtenido. La entidad servicio contiene el nombre de la compañía y el medio de transporte que utiliza para sus viajes.

Spider

La entidad *spider* hace referencia a un *web crawler* que estará al cargo de mantener actualizada la información de los horarios de un servicio en la base de datos. Cada *spider* tiene un nombre y unas URLs asociadas a la págian web de la empresa de la cual obtiene la información. Actualmente hay un *spider* encargado de rastrear la información de los horarios de tren de Renfe Cercanías en Bilbao.

Viaje

Un horario está compuesto de varias entidades viaje. La entidad viaje describe cada una de las filas de un horario, así como uno de los viajes que se muestra en un horario con un origen y un destino, en una fecha determinada. Cada entidad viaje engloba información relativa a dicho viaje, así como el origen, el destino, la fecha, el servicio, la hora de salida, la hora de llegada, la línea, el transbordo (si lo hay) y el tiempo de viaje. En caso de haber transbordo, también existirá la hora de llegada y de salida y la línea de la parte del viaje tras realizar dicho transbordo.

Capítulo 4

Análisis de antecedentes

Para la consulta de horarios de transporte público, existen hoy en día una serie de aplicaciones web que son desarrolladas por las empresas responsables de ofrecer el servicio de transporte. El problema es que su acceso desde dispositivos móviles puede ser lento y engorroso por el hecho de acceder a ellas a través de un navegador web y por que a menudo, requieren completar formularios que en una situación de mucha prisa no son nada convenientes.

FIGURA 4.1: Aplicación web de Renfe Cercanías en Bilbao.

Por otra parte, existen aplicaciónes para dispositivos móviles pero cada empresa tiene su propia aplicación. Esto significa que si se quiere viajar en autobús por Bizkaia, ha de instalarse alguna de las aplicaciones que ofrezca el servicio, sin embargo, si se va a realizar un viaje que cruce varias regiones, es posible que haya que instalarse varias aplicaciones.

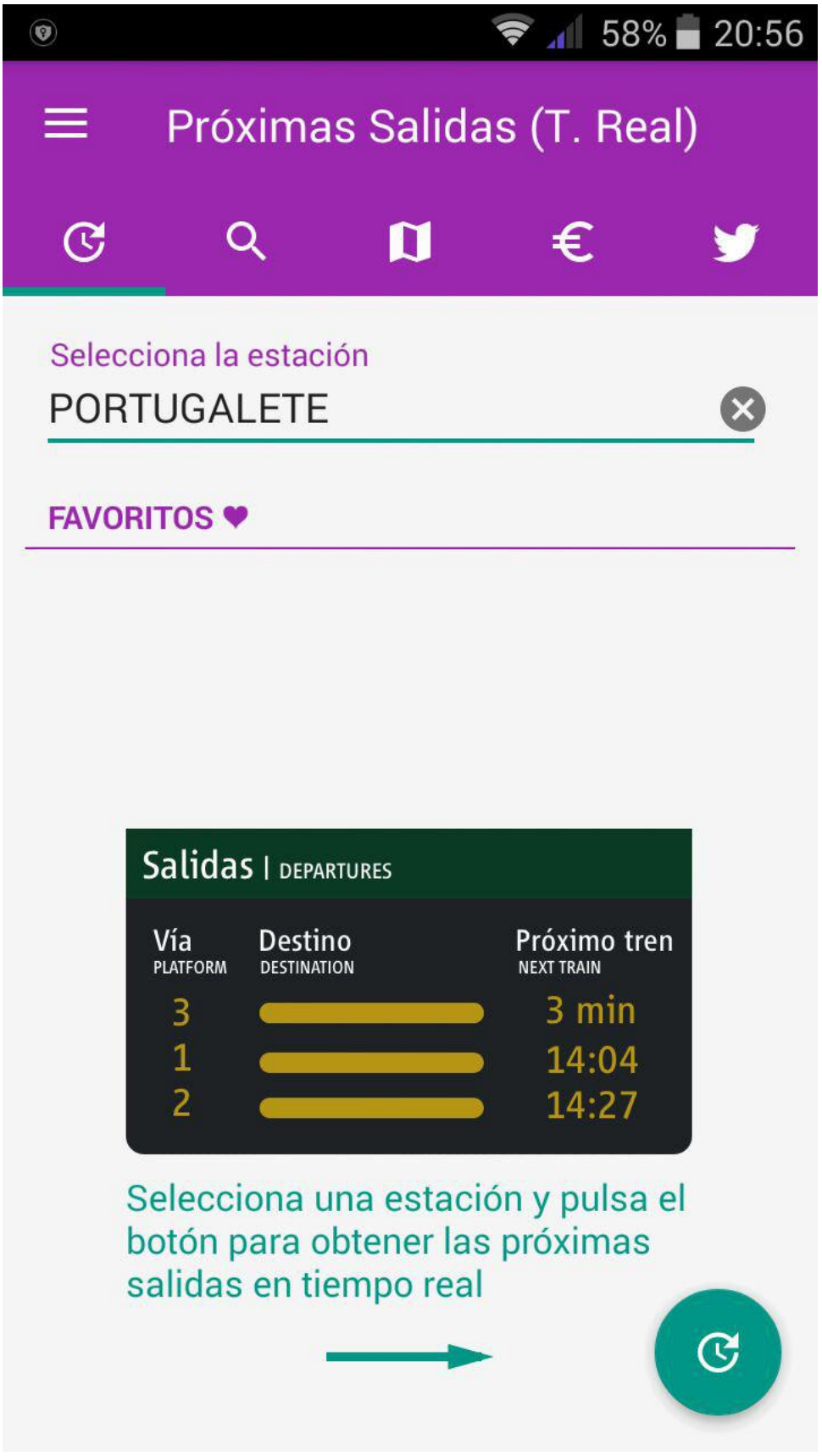

FIGURA 4.2: Aplicación para dispositivos móviles de Renfe Cercanías en Bilbao.

Con el *bot* de Telegram de este proyecto, se pretende unificar los servicios de varias empresas, sin tener que instalarse ninguna aplicación particular (solo hay que tener instalada la aplicación de Telegram). Esto ofrece al usuario una comodidad y rapidez difíciles de alcanzar mediante el método tradicional de buscar las aplicaciones de cada empresa y aprender como se usa cada una.

A fecha de la redacción de la memoria del proyecto, el *bot* incluye soporte para los horarios de Renfe Cercanías en la zona de Bilbao, pero el código fuente está diseñado para que la integración de nuevos servicios requieran únicamente el desarrollo de nuevos *web crawlers*, lo cual hace que el proceso sea relativamente fácil.

Capítulo 5

Análisis y diseño

En esta sección se mostrarán tres diagramas fundamentales en todo proyecto. El diagrama de secuencia, el diagrama de clases y el diagrama de estados.

Para poder diseñar los mecanismos utilizados por un *bot* a la hora de mantener una conversación, se va a analizar lo que es una conversación y a explicar brevemente el concepto de máquina de estados.

5.1. Conversación

Una conversación es el diálogo entre dos o más sistemas que intervienen alternativamente intercambiando información. En el ámbito de los *chatbot* se pueden diferenciar tres estados distintos de una conversación, a saber, una conversación activa, una parada y una cancelada.

Una conversación se podría asimilar con el comando de un *chatbot*. En cuanto el usuario ejecuta el comando, se inicia un intercambio de información entre éste y el *bot* con el fin de obtener los resultados de dicho comando.

Conversación activa

Se trata de toda aquella conversación que se encuentra en mitad del proceso de diálogo, sin que ninguno de los participantes la haya dado por concluida.

Conversación parada

Cuando un usuario inicia una conversación mediante la ejecución de un comando y tras una serie de intercambios, el usuario obtiene los resultados deseados, se dará el caso en que o bien el propio *bot* dé por concluida la conversación, o que el propio usuario sea el que lo indique. Llegada a este punto, la conversación cambiará al estado de parada.

Conversación cancelada

Parecido al caso de la conversación parada, la diferencia radica en que en este caso, el usuario ha decidido cancelar la conversación antes de concluirla, por lo que la conversación, en lugar de concluir (pararse) se cancela.

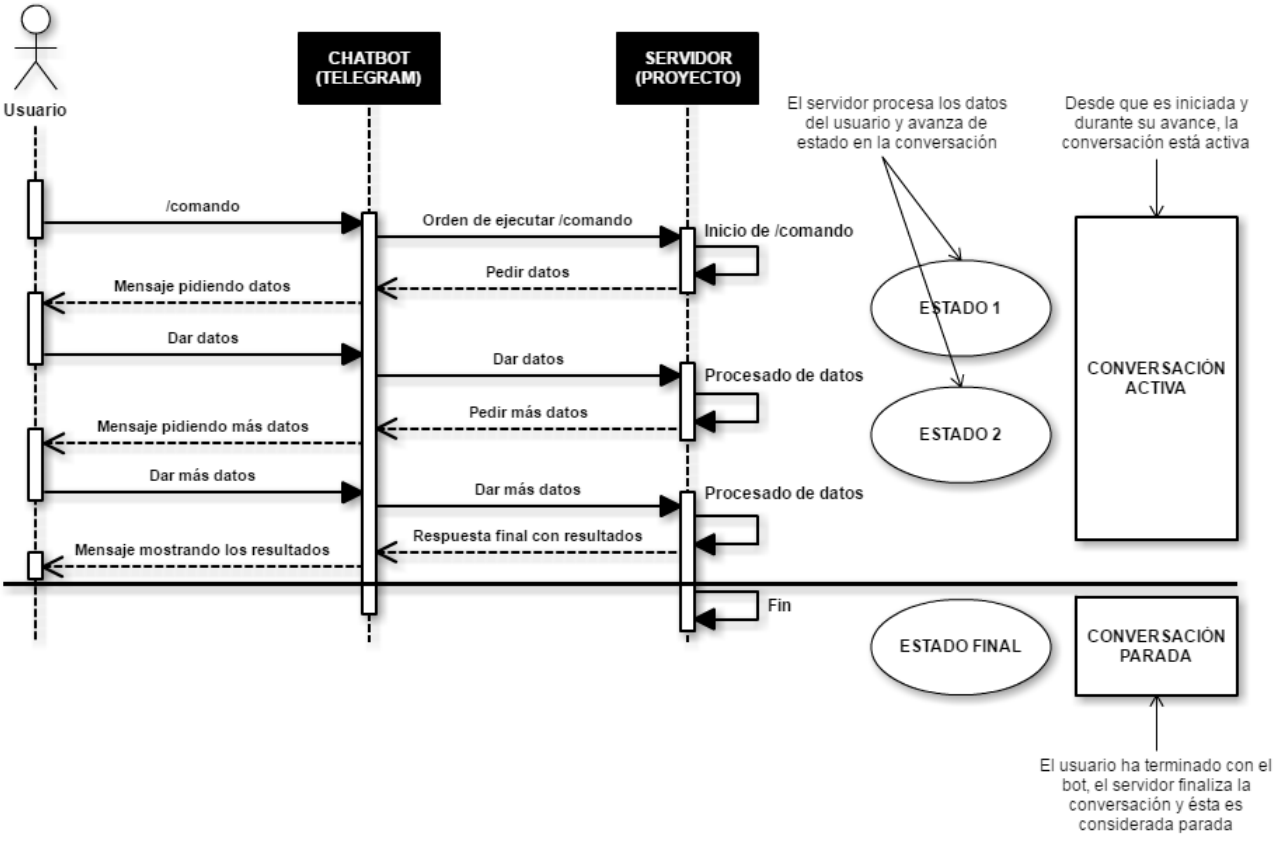

FIGURA 5.1: Muestra los estados de una conversación y las diferencias entre una conversación activa y parada.

En la figura 5.1 no se ha añadido el caso que muestra una conversación cancelada. Para ello simplemente habría que tomar el mismo caso y cambiar el punto en que el usuario envía datos por segunda vez por uno en el que el usuario envía la orden de ejecutar el comando */cancel*. El *chatbot* mandaría la orden al servidor y éste lo llevaría

a cabo, terminando la conversación en mitad de la ejecución y cambiando el estado de la conversación al de una conversación cancelada.

5.2. Máquina de estados

Una máquina de estados es un modelo de comportamiento de un sistema que funciona mediante entradas y salidas. Lo que lo diferencia de otros sistemas similares, es que las salidas no sólo dependen de las entradas actuales, sino que pueden depender también de entradas más antiguas.

La capacidad de poder guardar datos sobre el histórico de entradas para mantenerlos y utilizarlos durante la conversación es imprescindible en una máquina de este tipo. Para conseguirlo, el *bot* utilizará un objeto asociado a cada conversación llamado *notes* o las notas de la conversación.

Gracias al guardado de esta información, cada vez que se ejecuta el comando en cualquiera de los pasos intermedios, el *bot* puede procesar la información de estas notas para poder avanzar al siguiente paso de la conversación.

5.3. Diagrama de estados

El diagrama de estados define el flujo de estados del *bot* de Telegram durante la ejecución del comando */ask*. En el diagrama de estados de la figura 5.2, se puede apreciar que el objetivo del *chatbot* es obtener la información relativa a la consulta de horarios que el usuario quiere realizar.

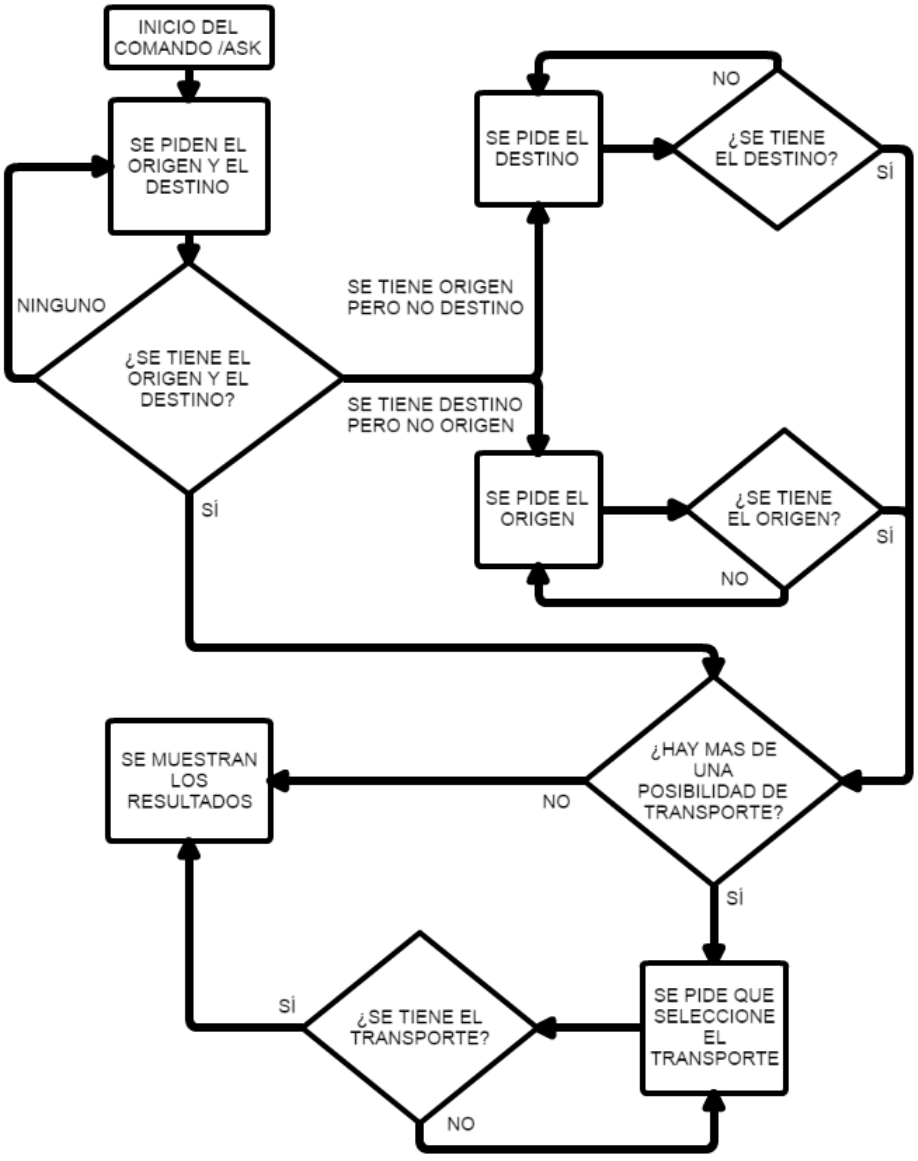

FIGURA 5.2: Diagrama de estados del *chatbot*.

Como se puede apreciar en el diagrama de estados, el *bot* puede llegar a pasar por un único estado antes de mostrar los resultados. En caso de no obtener el origen y el destino la primera vez que los pide o cuando haya más de una posibilidad de transporte, será cuando tenga que pasar por otros estados.

5.4. Diagrama de clases

Un diagrama de clases muestra la estructura de un sistema, mostrando las clases, sus métodos y atributos, y las relaciones entre estas en un diagrama.

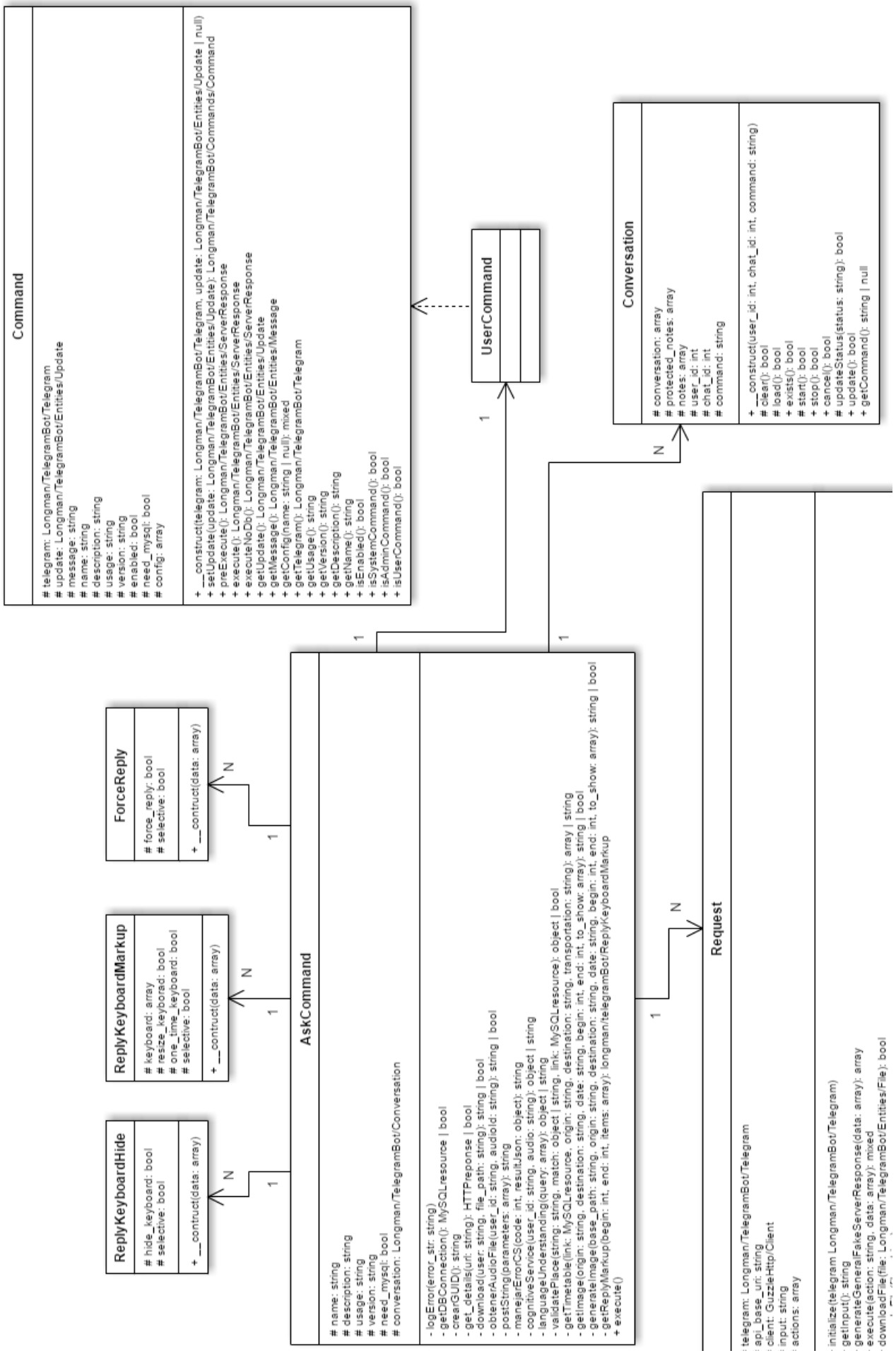

FIGURA 5.3: Diagrama de clases.

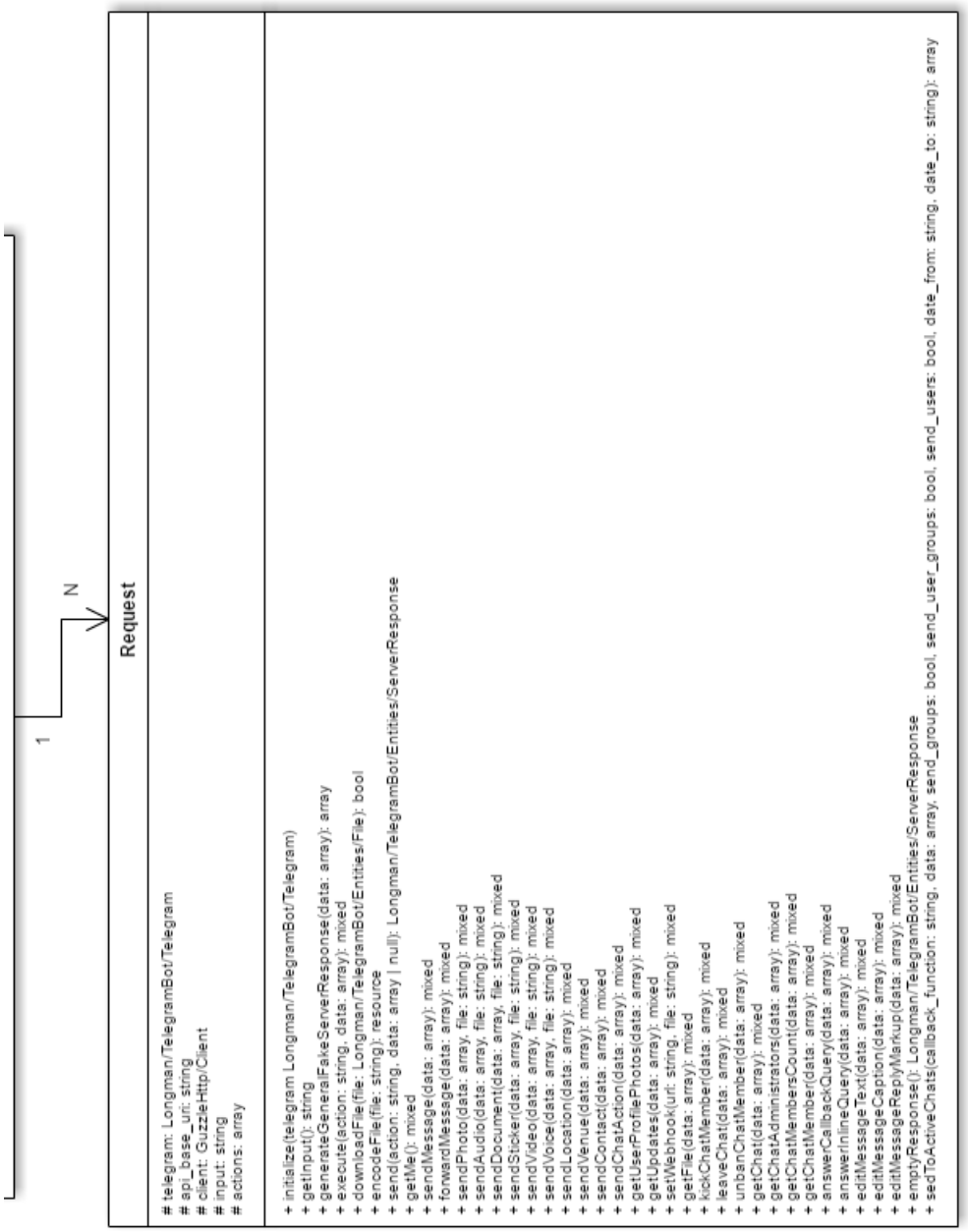

FIGURA 5.4: Diagrama de clases. (clase Request)

La clase principal del diagrama de las figuras 5.3 y 5.4 es la clase AskCommand, que es la que contiene todo el código desarrollado para el *bot* del proyecto. El resto de clases son clases de la biblioteca *longman telegram-bot* que sirven para varios propósitos, desde generar menús como parte de las respuestas al usuario hasta gestionar la propia conversación y almacenar las notas utilizadas en la estructura de la máquina de estados.

5.5. Diagramas de secuencia

Un diagrama de secuencia es un diagrama de comportamiento que modela la interacción a través del tiempo entre dos o más objetos durante la ejecución de un proceso.

Los diagramas de secuencia mostrados a continuación se corresponden a dos casos de uso distintos para mostrar los distintos estados por los que pasa el *bot* si se presentan distintos flujos de conversación.

En el primero (figuras 5.5 y 5.6), el usuario ejecuta el comando y cuando el *bot* le pide el origen y destino, éste envía al *bot* una nota de voz del tipo: "quiero viajar desde [ORIGEN]". El *bot* convierte la nota de voz a texto utilizando los servicios de *Microsoft Cognitive Services* y acto seguido, procesa el texto mediante el servicio de *api.ai*. Por medio de estas operaciones detecta un origen y pasa a validarlo con éxito. Como le falta un destino procede a pedírselo al usuario y éste envía el destino, esta vez en formato de texto.

El *bot* procesa el texto y valida el destino detectado también con éxito por lo que pasa a realizar un consulta con los datos obtenidos con el fin de detectar los tipos de transporte asociados a ese origen y destino. Como el *bot* detecta que sólo tienen el tren asociado, pasa directamente a analizar los datos de la consulta para determinar el rango de información que aparecerá en la imagen que enviará como resultado (la imagen contendrá un máximo de 16 filas y si el usuario quiere ver las filas siguientes, deberá hacerlo mediante los botones de navegación que aparecerán junto con la imagen). El usuario pulsa el botón de "Siguiente" y el *bot* procede a enviarle una nueva imagen con nuevas filas en ella. Finalmente el usuario pulsa el botón "FINALIZAR" para comunicarle al *bot* que ha terminado su consulta. El *bot* termina la conversación y se despide del usuario.

Los estados por los que pasa el *bot* son:

Estado 1

El *bot* ha iniciado la ejecución del comando */ask* y está a la espera de obtener el origen y el destino por parte del usuario por primera vez.

Estado 2

El *bot* tiene el origen pero no el destino por lo que cambia de estado al estado de espera hasta obtener el destino.

Estado 3

El *bot* ha mandado los resultados al usuario y está a la espera de que el usuario le pida pasar de página (si es que puede) o que le diga que ha terminado de consultar la información.

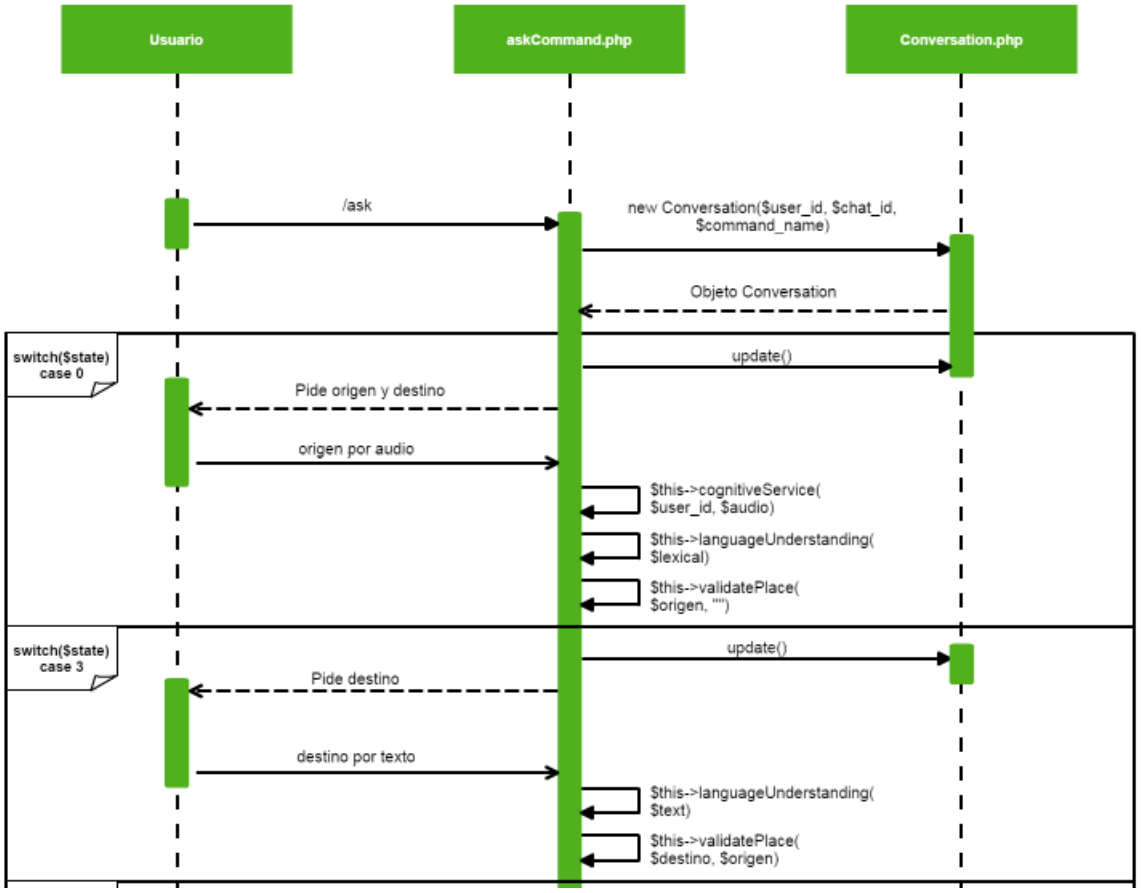

FIGURA 5.5: Diagrama de secuencia 1. El *bot* obtiene la información de la consulta en dos pasos antes de mostrar los resultados.

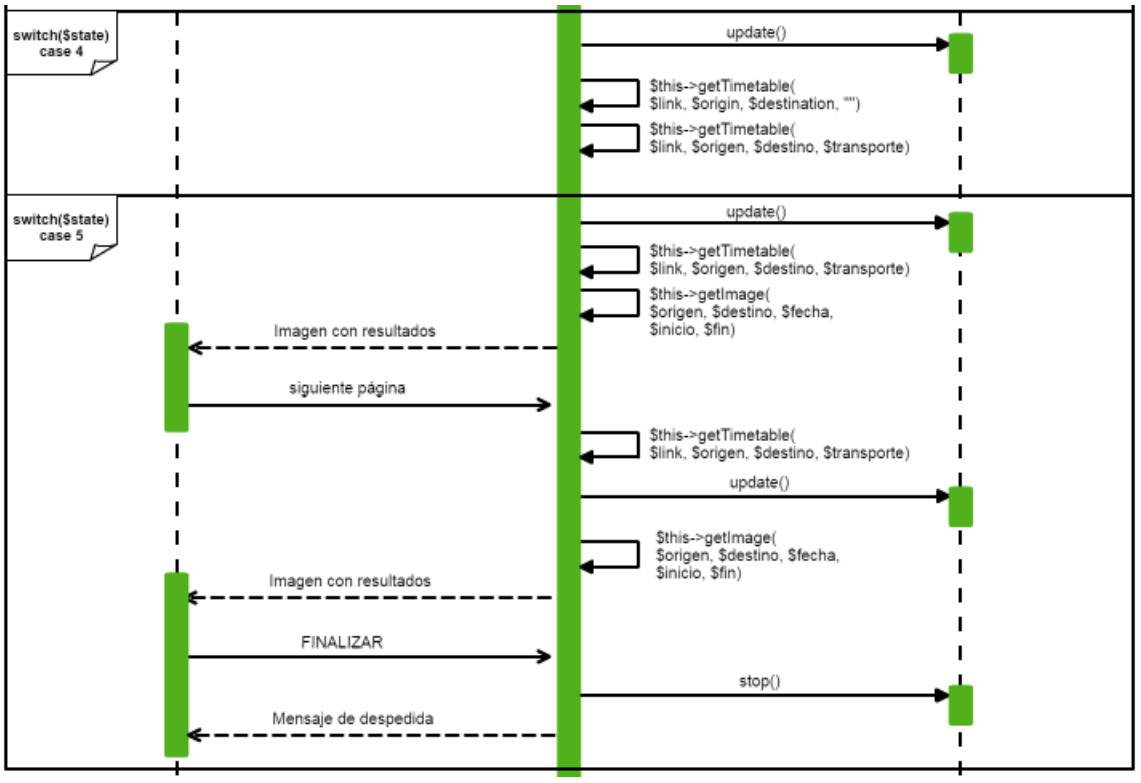

FIGURA 5.6: Diagrama de secuencia 1 (continuación). El usuario avanza una página en los resultados antes de finalizar la conversación.

En la figuras 5.7 y 5.8, la situación es similar al anterior caso de uso. La primera vez que manda información al *bot*, en lugar de mandar un audio con una frase, el usuario menciona el nombre del lugar y nada más. Esto provoca que el *bot* sea incapaz de detectar un origen o un destino pero no impide que sepa que se trata de un nombre que éste tiene en la base de datos, por lo que procede a preguntarle al usuario si el nombre mencionado es su origen o su destino. El usuario entonces elige la opción "Destino" y el *bot* procede a pedirle el origen tras validar dicho destino. El usuario comunica al *bot* el origen vía texto y tras validarlo y realizar la consulta a la base de datos, el *bot* detecta más de un transporte compatible con esa combinación de origen y destino. Se muestran al usuario las opciones de transporte disponibles y se le pide que seleccione una. Cuando el usuario selecciona "TREN" el *chatbot* procede a procesar el resultado de la consulta con la base de datos para obtener el rango de información del primer resultado, genera la imagen, y la envía. Una vez más, el usuario quiere pasar a la siguiente página y tras hacerlo termina la consulta de horarios.

Los estados por los que pasa el *bot* son:

Estado 1

El *bot* ha iniciado la ejecución del comando */ask* y está a la espera de obtener el origen y el destino por parte del usuario por primera vez.

Estado 2

El *bot* no sabe si lo que le ha dicho el usuario es un origen o un destino y pasa a la espera de su respuesta.

Estado 3

El *bot* tiene el destino pero no el origen por lo que cambia de estado al estado de espera hasta obtener el origen.

Estado 4

El *bot* detecta más de un transporte compatible con la consulta del usuario, por lo que le muestra las opciones y cambia de estado a la espera de que el usuario seleccione un transporte.

Estado 5

El *bot* ha mandado los resultados al usuario y está a la espera de que el usuario le pida pasar de página (si es que puede) o que le diga que ha terminado de consultar la información.

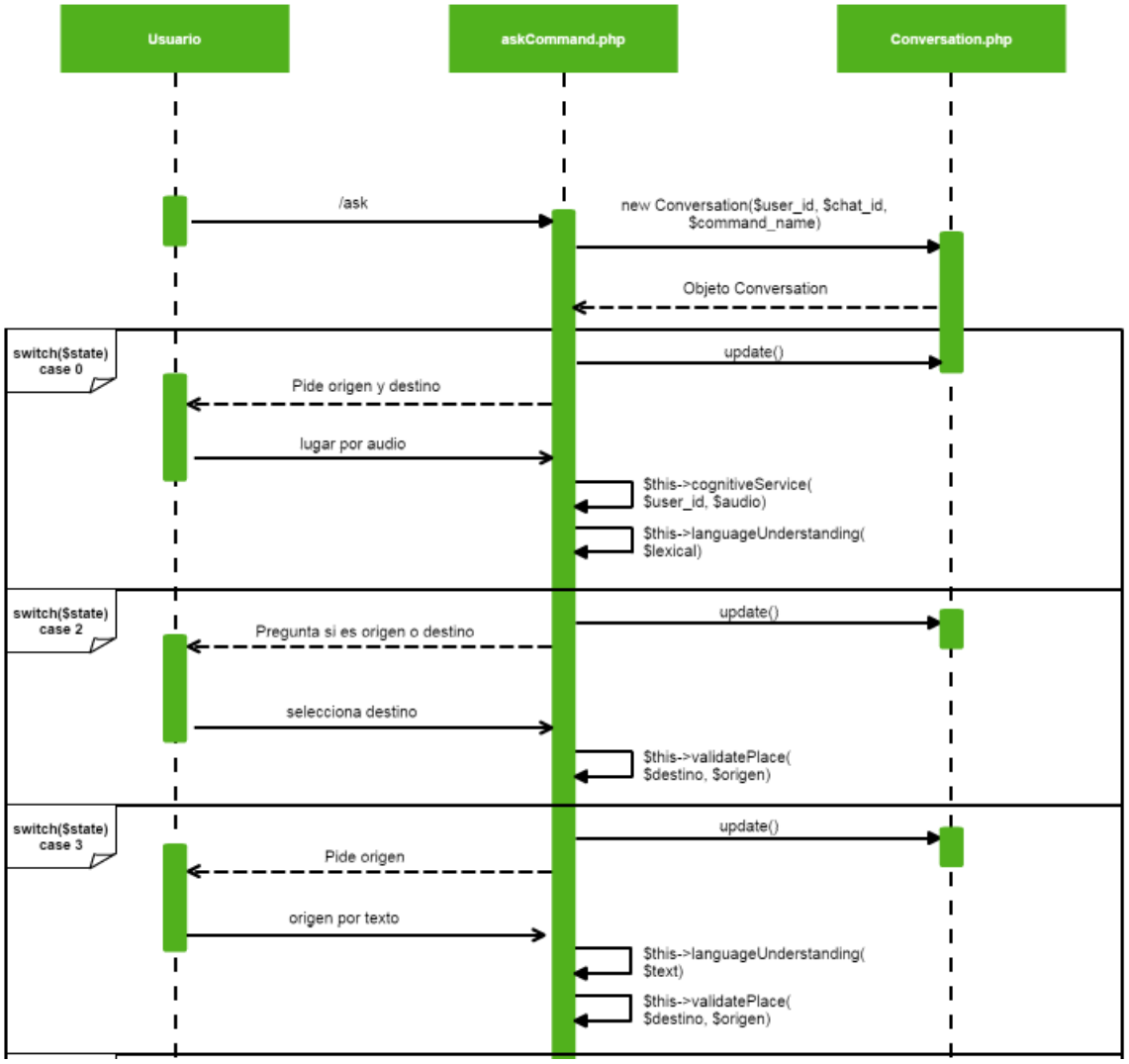

FIGURA 5.7: Diagrama de secuencia 2. El *bot* necesita tres pasos para obtener el origen y el destino.

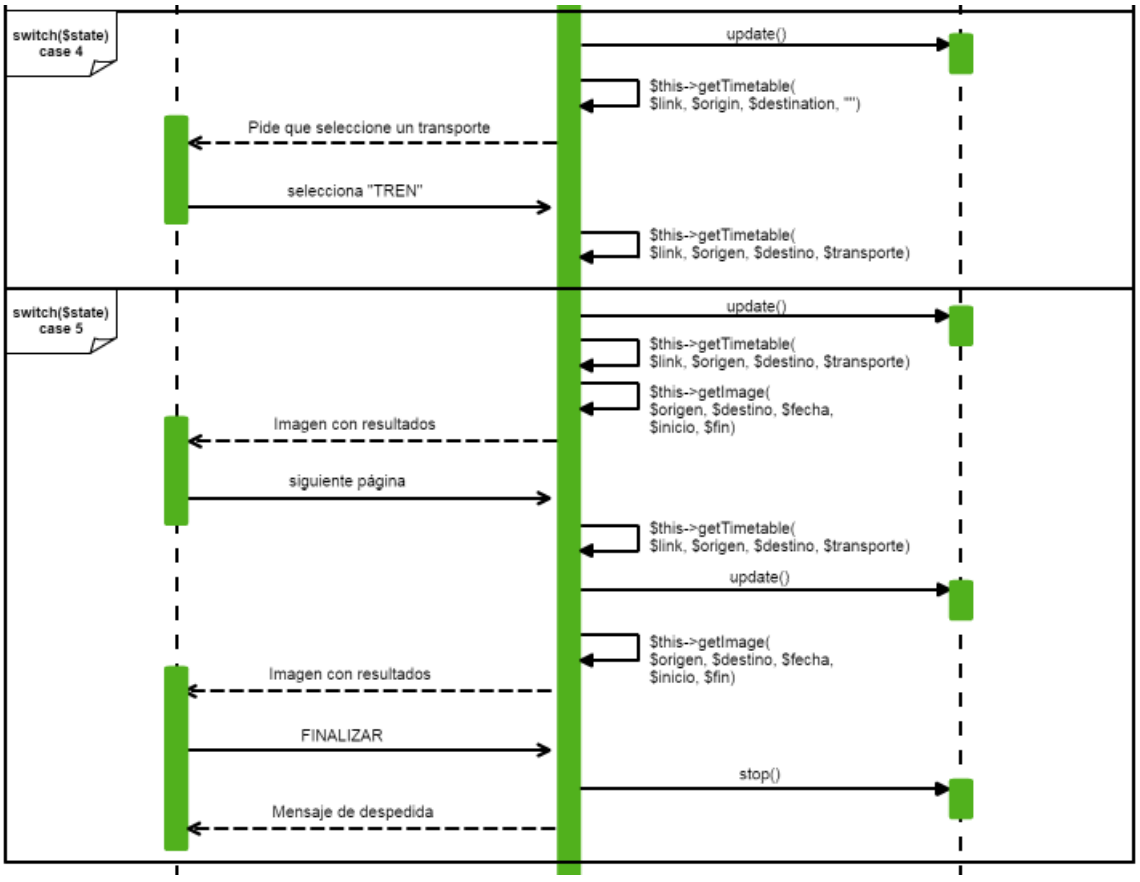

FIGURA 5.8: Diagrama de secuencia 2 (continuación). El usuario tiene que elegir un transporte antes de ver los resultados, avanza una página y finaliza la conversación.

Capítulo 6

Desarrollo

En esta sección se van a detallar los pasos seguidos a la hora de desarrollar la aplicación de este proyecto así como los percances encontrados y las pautas seguidas para resolverlos. El desarrollo estará separado en dos partes, siendo la primera el desarrollo del *web crawler* y la segunda el desarrollo del *chatbot* de Telegram.

En términos generales, puede decirse que primero se terminó de desarrollar el *web crawler* (a pesar de que más tarde sufriría algunas modificaciones) y después el *chatbot*. Sin embargo, antes de empezar a desarrollar el *web crawler* se exploraron los mecanismos con los que funciona el *chatbot* llegando a desarrollar un *bot* de prueba que se utilizaría de guía y sobre el que se desarrollaría el *chatbot* en su versión a día de escribir la memoria del proyecto.

6.1. Necesidades del cliente

Antes de explicar las dos partes del desarrollo, conviene tener una visión de los requerimientos mínimos que el *bot* debería cumplir, ya que fueron discutidos con el fin de desarrollar el proyecto poniendo énfasis en llevarlos a cabo.

Éstas necesidades son: la facilidad que debe tener el *bot* al ser utilizado y la ampliación progresiva de opciones de transporte con el fin de unificar poco a poco todos los horarios de las principales empresas de transporte público en una sola aplicación.

Facilidad de uso del *chatbot*

Una de las principales características de este proyecto es que se trata de un *bot* de uso público y lo que esto requiere es que su utilización sea lo más cómoda posible. Es por esta razón que se hayan elegido tecnologías para el tratamiento del lenguaje natural y diseñado un flujo de conversación simple.

Capacidad de integraciones futuras de transportes

El *chatbot* incluye actualmente los horarios de Renfe Cercanías de la zona de Bilbao pero en el futuro habrá que añadirle más opciones, por ello el *chatbot* debe de estar programado para manejar la conversación con la premisa de que existen varias opciones de transporte en la base de datos y que es posible que una combinación de origen y destino pueda coincidir en varios transportes públicos. Gracias a este sistema, en el futuro, el *bot* ofrecerá varias posibilidades al usuario.

6.2. Desarrollo de la base de datos

La biblioteca de PHP *longman telegram-bot* incluye una base de datos que es utilizada para el manejo y almacenamiento de los mensajes y chats en los que el *chatbot* participa. A pesar de la gran utilidad de esa base de datos, se requería añadir más tablas para guardar los horarios. La figura 6.1 muestra las tablas propias de la biblioteca *longman telegram-bot* y la figura 6.2 muestra las tablas creadas durante el desarrollo del proyecto.

Las tablas necesarias para manejar la información rastreada por los *web crawlers* son la tabla de servicios, la de lugares y la de viajes. A continuación se detallará brevemente la función de cada tabla.

Tabla servicios

En esta tabla se almacenan los identificadores de los servicios que vaya a utilizar el *bot* junto con el identificador del medio de transporte que ofrece dicho servicio. Cuando en el futuro se vaya a añadir un nuevo servicio mediante la creación de un nuevo *spider*, habrá que incluir la información del mismo en esta tabla.

Tabla lugares

Esta tabla almacena una lista de los lugares (origenes y destinos) que hay relacionados con un servicio. Se utiliza en el proceso de validación del origen y destino durante la conversación.

Tabla viajes

La tabla que esencialmente contiene los horarios. Cada fila de esta tabla contiene información referente a un viaje ofrecido por un servicio y con los datos de origen, destino, tiempo de viaje, etc, necesarios para formar una fila de la tabla de horarios que se mostrará al usuario como resultado.

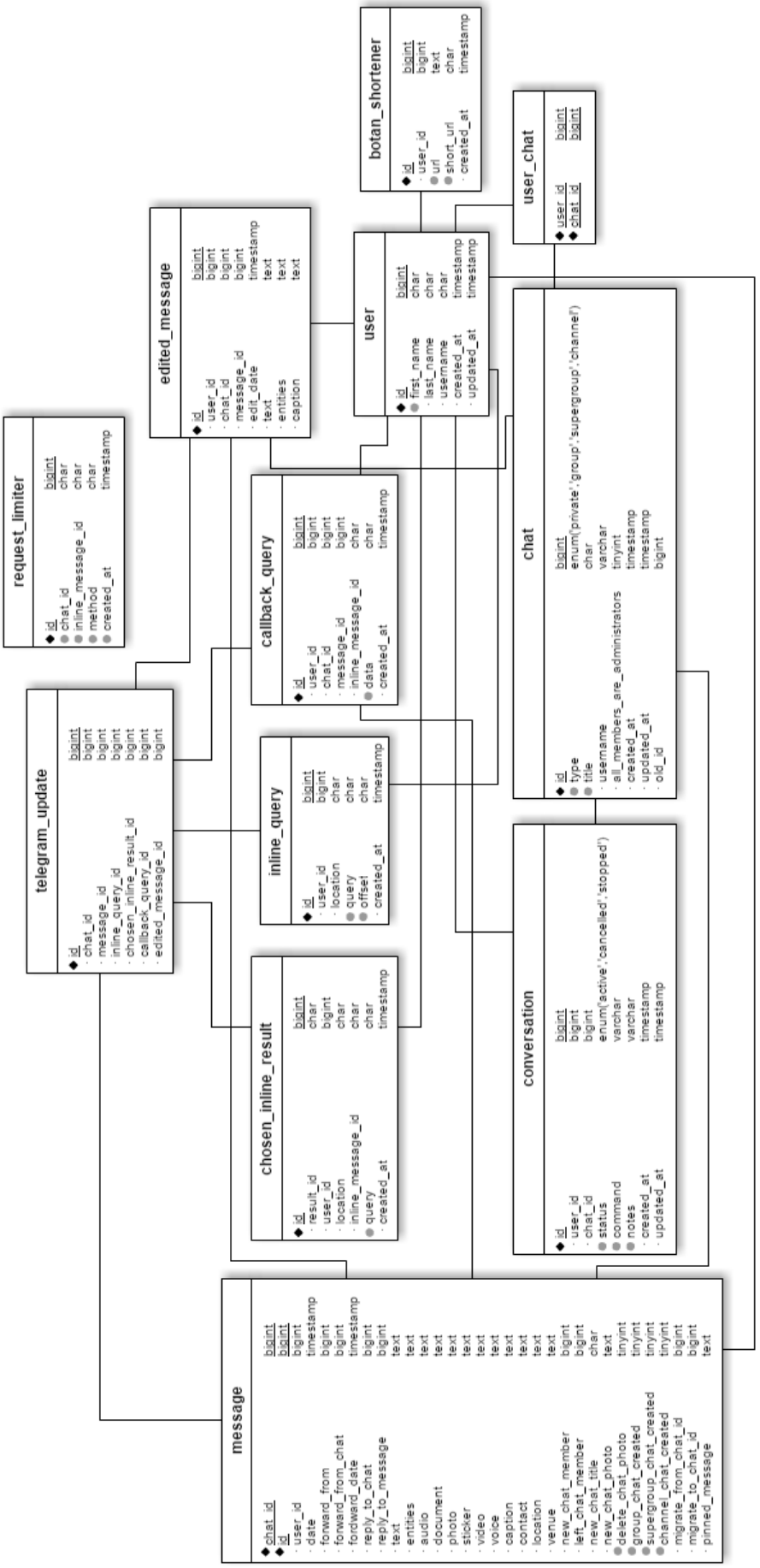

FIGURA 6.1: Diagrama entidad relación de las tablas de la base de datos correspondientes a la biblioteca *longman telegram-bot*.

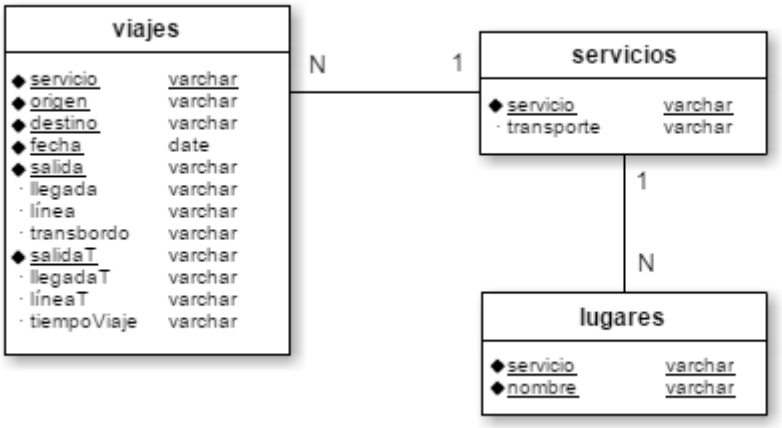

FIGURA 6.2: Diagrama entidad relación de las tablas nuevas creadas para almacenar los horarios.

6.3. Desarrollo del *web crawler*

Ahora se procederá a explicar los detalles de la implementación del *web crawler*. Lo primero que se hizo fue programar un *spider* sencillo que lo único que hiciese fuese guardar las páginas que se le ordenara descargar. Una vez entendida la mecánica básica con la que funcionaba, se procedió a buscar la página de alguna empresa de transporte público para intentar descargar los datos necesarios. Se eligió la página de Renfe Cercanías de Bilbao por haber encontrado en ella una dirección que mostraba exclusivamente el formulario para realizar consultas de horarios.

Cabe destacar que al empezar a hacer pruebas de peticiones a Renfe con el código, se detectó que el archivo *robots.txt* de la página no permitía el acceso al *bot*. Para solucionarlo simplemente se cambió la configuración del *spider* para que ignorase los archivos *robots.txt* de las páginas. Esta práctica no es recomendable, y únicamente se hizo con propósitos educativos.

Una vez examinada la web, se procedió a desarrollar un sistema que visitara la página y que generase una consulta con cada combinación de origen y destino disponible en los selectores de su formulario. El resultado fue que se puso en cola una cantidad enorme de páginas (puesto que cada vez que se envía el formulario con una combinación añade una dirección a la cola) y al ejecutar el *spider* comenzó a aparecer en la consola muchísima información. Esto generó la preocupación de que, al trabajar Scrapy tan deprisa y al

tener una cantidad tan grande de páginas que visitar en un mismo dominio, el programa pudiese interferir con el servicio ofrecido por las empresas a causa de una sobrecarga de peticiones. Esto podía resultar en un bloqueo por IP al *spider* por parte de la página, lo que sería un problema mayor y a tener en cuenta con las *webs* de los futuros servicios que se consultasen.

La solución fue simple, se añadió un retardo al *spider*, de manera que tuviese que esperar 3 segundos entre cada petición. Con todo se analizaron las páginas que tenía que procesar el *spider* para encontrar posibles diferencias en los datos desplegados en las distintas combinaciones de origen y destino. Al hacerlo, se detectaron dos variantes de tabla de resultados: las tablas en las que el viaje no incluía un transbordo y las que sí.

Al haber encontrado la opción del transbordo para los viajes, se tuvo que modificar la base de datos para que las entradas de la tabla de viajes contemplasen la posibilidad de incluir transbordos.

Con la base de datos modificada, se pudieron realizar las primeras pruebas. Primero hubo que estudiar el funcionamiento de las expresiones *XPath*, que son las utilizadas por Scrapy para obtener los objetos de la estructura de los documentos HTML. Utilizando estas expresiones, se desarrolló la función *parse_table*, que sería la encargada de procesar las páginas con tablas de horarios de la cola.

Cada vez que una tabla es procesada, la función *parse_table* genera un objeto *Viaje* por cada fila de la tabla y se la pasa a la clase *ViajePipeline* en el archivo *pipelines.py*. Finalmente, la clase *ViajePipeline* procesa cada *Viaje* para validarlo y meterlo en la base de datos.

Tras finalizar de desarrollar el código del *spider*, se procedió a realizar pruebas para detectar errores y determinar algunos aspectos de la ejecución del *web crawler*. Después de muchos cambios para arreglar errores y demás ajustes, al probar el *spider* en conjunto con el *chatbot* (una vez que el *chatbot* fue desarrollado), se optó por configurar el *spider* con un retardo de 1 segundo entre cada petición para reducir el tiempo de ejecución y también se eligió rastrear los horarios de un único día, con la condición de que el *spider* fuese ejecutado todos los días.

Cuando se iba a ejecutar el *web crawler* en el servidor de la universidad, surgieron algunos problemas con los permisos. Por una parte, se necesitó la ayuda del director para instalar el *software* necesario, ya que por motivos de seguridad, sólo él podía instalarlo.

Por otra parte, se decidió ejecutar el *crawler* de Scrapy todos los días a las dos de la mañana y para ello, se haría uso del administrador de procesos en segundo plano: *cron*. El problema era que en un principio no se ejecutaba el trabajo, por lo que se trató de cambiar el comando e incluso se creó un script para hacerlo. Finalmente, se dedujo que el problema estaba una vez más en los permisos y tras consultarlo con Juanan, el problema fue solucionado.

6.4. Desarrollo del *chatbot*

El desarrollo de la parte principal del proyecto, en esta sección se documentarán las pautas seguidas y los problemas encontrados, así como la resolución los mismos durante el desarrollo del *bot* de Telegram del TFG.

Como ya se ha comentado, lo primero que se hizo fue estudiar el funcionamiento de los *chatbots* de la biblioteca *longman telegram-bot* y desarrollar un pequeño comando copiando parte del código de uno de los comandos predeterminados. El comando realizaba una encuesta al usuario y al final de ella, mostraba los datos introducidos en la encuesta. Este comando resultó ser muy útil ya que manejaba distintos tipos de datos como imágenes y texto, así como distintos tipos de métodos de entrada, como la escritura directa en la barra del *chat* o la selección de uno de los botones mostrados en el teclado de Telegram.

Tras el desarrollo del *spider* de Scrapy, se procedió a mantener una reunión con el director del proyecto para determinar las bases del comportamiento del *chatbot* que se quería implementar. Con la estructura de la base de datos definida y el *web crawler* implementado, se procedió a desarrollar un comando simple para el *chatbot* que se asemejaría mucho al comando de la encuesta, cambiando los datos de la encuesta por los datos que se necesitarían a la hora de realizar la consulta final para solicitar un horario.

El comando (que fue denominado */ask*) tenía en este punto la estructura básica pero aún distaba de cubrir las necesidades del usuario definidas por el director. Se llegó a

la fase de desarrollo en la que el objetivo principal era procesar mensajes en lenguaje natural por parte del usuario, tratando primero de procesar la información relevante contenida en un texto (la conversión de audio a texto vendría después). Se exploraron varias tecnologías disponibles y al principio se optó por usar un servicio de Microsoft llamado *Language Understanding Intelligent Service* o LUIS.

Para utilizar LUIS había que crearse una cuenta en su página web y crear un "agente" que sería el programa de inteligencia artificial que habría de ser especializado en tratar frases del contexto deseado. Tras una investigación en el modo de trabajo de los agentes, se procedió a crear uno que detectara la acción de buscar un viaje en un texto y que detectara las palabras que definirían el origen y/o el destino.

Cuando tan solo hacía falta realizar las consultas HTTP a la dirección de contacto del agente, se descubrió que la clave del agente que ofrecía la cuenta de Microsoft era para uso experimental y aceptaba un máximo de mil peticiones por mes (las claves no experimentales eran de pago). Se debatió en si se debía buscar otro servicio de prestaciones similares y que no tuviese la limitación y finalmente se encontró uno que cumplía con ello: el servicio de *api.ai*.

Este servicio es muy parecido al servicio de LUIS, con una mecánica casi idéntica, pero pertenece a Google (fue adoptado por Google en 2016). Al igual que al utilizar LUIS, hace falta tener una cuenta con una clave y crear un agente que habrá que entrenar. En un principio se creó un agente que identificara la intención de viaje en una frase y dos entidades que podían ser encontradas en la intención: la entidad "deLugar" (origen) y "aLugar" (destino). El siguiente paso para que el agente funcionase de forma más precisa fue el de añadirle frases de referencia que formarían su conjunto de entrenamiento.

Un agente necesita un conjunto de datos que sea el conjunto de entrenamiento. Además, los agentes guardan un historial con los casos nuevos que se ha ido encontrando y los resultados obtenidos (si ha clasificado una intención, si ha encontrado alguna entidad, etc.) para que en cualquier momento, sea accedido y se puedan validar los resultados añadiéndolos al conjunto de entrenamiento. Este sistema de retroalimentación permite que a medida que se vaya usando, un agente se vuelva más y más fiable con sus predicciones.

La figura 6.3 muestra el menú del panel de control del agente en la página de *api.ai*.

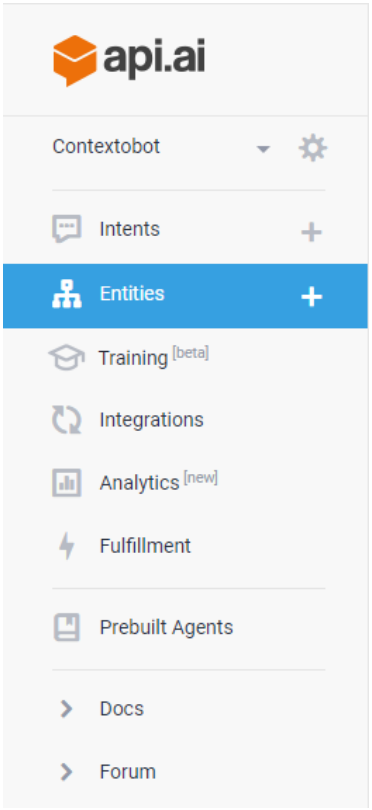

FIGURA 6.3: Interfaz de *api.ai*, panel de control.

Llegados a este punto, se descubrió un problema de detección de entidades, se vió que el agente solía detectar con relativa facilidad la intención de viaje pero no lograba captar ni el origen ni el destino y fue entonces, al investigarlo en la interfaz de *api.ai*, cuando se descubrió que se podía definir una lista de posibilidades para cada entidad. Esto hizo surgir una relación entre la base de datos y *api.ai*. Lo que hacía falta era generar una nueva tabla en la base de datos que guardase las opciones de viaje de los servicios y una forma de insertárselos al agente de *api.ai*.

Por suerte, *api.ai* tiene una API bien documentada y no se tardó en crear una tabla en la base de datos que contuviese la lista de lugares con los que se pudiese realizar consultas para, acto seguido, incluir código en el *spider* para que rellenase dicha tabla al ejecutarse y desarrollar un pequeño *script* en PHP que se conectase a la API de *api.ai* y subiese la lista de lugares apropiada a cada entidad.

Después de una seria de pruebas exhaustivas, desarrollando estructuras de control en el código de la conversación, creando las funciones necesarias, creando el código de las peticiones en cURL (tras investigar su funcionamiento) y analizando el contenido de las

respuestas de *api.ai*, el *chatbot* ya disponía de un sistema de reconocimiento del lenguaje natural.

El siguiente nivel hacia una interacción más natural con el usuario era el de la comunicación por voz, para el que ya se había encontrado un servicio viable: *Microsoft Cognitive Services*. Una vez más se decidió por un servicio de Microsoft que resultó no tener la limitación de LUIS.

El primer problema encontrado con este servicio fue que los audios debían ir en formato WAV y se detectó que los audios almacenados en los servidores de Telegram estaban en formato OGG. Para solucionarlo, se utilizó una herramienta de conversión de audios llamada FFmpeg, y que convertiría los audios al vuelo en cuanto se recibiesen de Telegram. Por lo tanto se tenía un sistema que si detectaba que el usuario enviaba un audio, al momento se descargaba el audio de los servidores de Telegram y se utilizaba FFmpeg para convertirlo a WAV.

Cuando se tuvo la información necesaria y el código de conexión con la API de *Cognitive Services* (una vez más en cURL) se detectó que cada vez que se intentaba obtener el resultado, se retornaba un error de los servidores de Microsoft. Tras unas tutorías con el director Juanan, se descubrió el problema: los datos no estaban correctamente introducidos en la petición POST que se enviaba a Microsoft. Hubo que cambiar ligeramente el código y finalmente se obtuvo el resultado deseado.

Se tenía finalmente un sistema para que el *chatbot* pudiera procesar consultas formadas con lenguaje natural en audio o texto. Al continuar desarrollando la conversación, se descubrió que si el usuario simplemente decía el nombre del origen o del destino, el servicio de *api.ai* no era capaz de detectar un origen o un destino, por lo que se tuvo que crear una nueva entidad para el agente: la entidad “Lugar” que no especificaba si era un origen o un destino pero sí, si se detectaba algún lugar de la base de datos.

La figura 6.4 muestra las entidades que debe tener la intención “viaje” en el panel de control de *api.ai*.

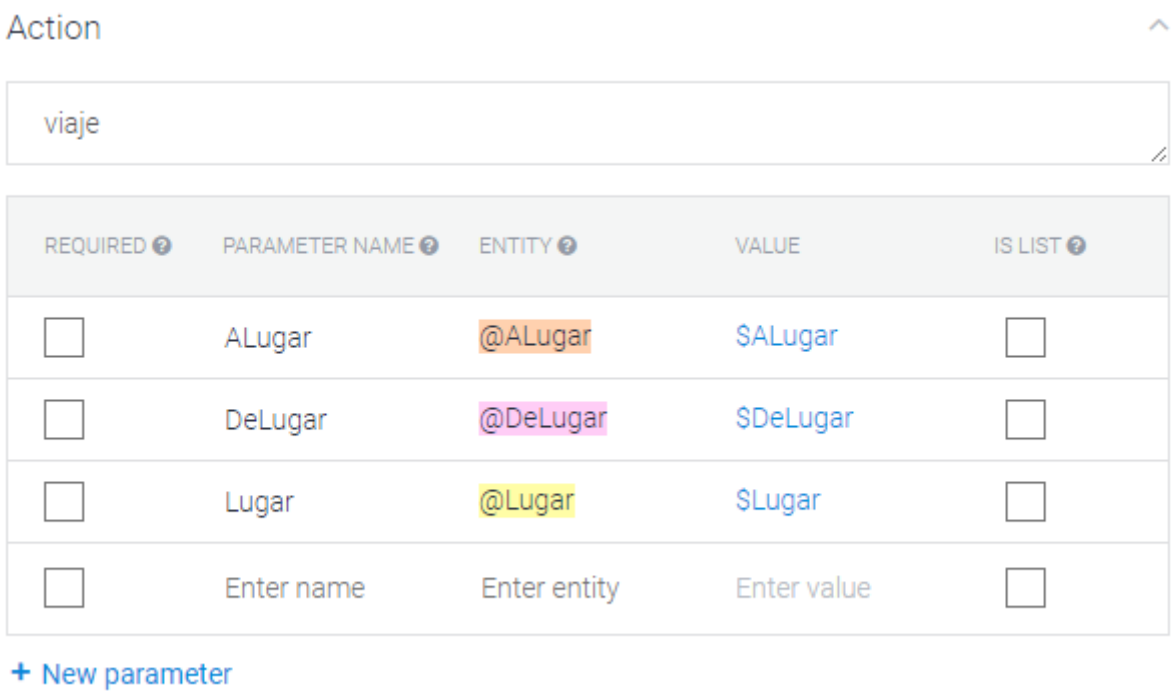

FIGURA 6.4: Entidades de la intención "viaje" de *api.ai*.

Se añadió el tratamiento para el caso de mencionar tan sólo el nombre del lugar y tras realizar varios cambios, controlar errores, y estructurar adecuadamente el flujo de estados del *chatbot* a lo largo de la conversación se consiguió un sistema que captase los datos necesarios y los validase de forma apropiada para realizar la consulta.

El siguiente paso era el de mostrar los resultados al usuario. Para ello, se optó por mandarle una imagen con los horarios solicitados, en lugar de generar una respuesta formada simplemente por texto. Para hacerlo, se utilizaría la biblioteca de PHP *ImageMagick* (o *Imagick*) que tiene un gran potencial hasta para generar GIFs y efectos complejos en una imagen desde cero.

Cuando se estaba diseñando un formato simple que contuviese la información necesaria se detectó que era posible que hubiese demasiadas filas en una tabla como para mostrarlas de manera cómoda en una sola imagen, por lo que se desarrolló un sistema de paginación con el que el usuario podría ver los horarios en varias imágenes y navegar por ellas usando el teclado con el que aparecerían las imágenes.

FIGURA 6.5: Imágen con resultados y menú del teclado para navegar.

Como recomendación por parte de Juanan, se decidió crear un segundo *bot* de Telegram, que sería el que funcionaría por medio de un *webhook* en el servidor de la universidad. El *chatbot* que se había estado desarrollando sería el *bot* de desarrollo y funcionaría de la forma en que había estado funcionando a lo largo de todo el desarrollo, mediante la ejecución de un archivo para procesar los mensajes enviados a Telegram. Cada vez que se quisiera que el *bot* respondiese a un mensaje, habría que ejecutarse el archivo *getUpdatesCLI.php* por lo que durante el desarrollo se utilizó un ciclo *while* en la consola de Ubuntu que ejecutara dicho *script* cada cuatro segundos.

Para manejar efectiva y cómodamente ambos *bots*, se subieron a un repositorio de Bit-Bucket cada uno, de forma que efectuar los cambios hechos en desarrollo a producción

fuese tarea sencilla mediante los comandos de Git. También se utilizó un sistema para los archivos de configuración utilizando un archivo que obtendría los datos correspondientes a cada *bot* mediante un fichero YAML.

Capítulo 7

Pruebas

Para realizar pruebas al *bot* y determinar su correcto funcionamiento, se utiliza una biblioteca de Python que se comunica con el cliente de línea de comandos de Telegram *tg* para mandar y recibir mensajes. Gracias a ello, puede actuar como intermediario entre el usuario y el *chatbot*, programándolo para que simule los casos de uso deseados y para que muestre los resultados de la ejecución en el momento.

Se utilizaron varios casos de uso distintos durante el testeo del *bot*, todos ellos en un único archivo escrito en lenguaje Python. En cada caso de uno se utiliza una función para enviar un mensaje al *bot*, se esperan unos segundos a que responda y se comprueba que ha respondido como se esperaba. En caso de responder correctamente aparece en la consola un "OK" en verde y en caso de no hacerlo un "KO" en rojo.

```
(py3env) will@will-VirtualBox:~/Desktop/contextobot_tests$ python3 ask_test.py
--Caso 1:

Comprobando mensaje 'Dime a dónde quieres ir':
OK
Comprobando que es una imagen:
OK
Comprobando mensaje 'Bye':
OK
--Caso 2:

Comprobando mensaje 'Dime a dónde quieres ir':
OK
Comprobando mensaje 'Dime o escríbeme el origen':
OK
Comprobando que es una imagen:
OK
Comprobando mensaje 'Bye':
OK
--Caso 3:

Comprobando mensaje 'Dime a dónde quieres ir':
OK
Comprobando mensaje '¿Ese es tu origen o tu destino?':
OK
Comprobando mensaje 'No he entendido lo que has dicho,':
OK
Comprobando mensaje 'Dime o escríbeme el destino':
OK
Comprobando que es una imagen:
OK
Comprobando mensaje 'Bye':
OK
(py3env) will@will-VirtualBox:~/Desktop/contextobot_tests$
```

FIGURA 7.1: Ejemplo de ejecución de tres casos de uso programados, todos han ido correctamente.

En todos los casos se inicia una conversación, se avanza siguiendo distintos caminos del diagrama de flujo y se finaliza por medio del botón "FINALIZAR" al recibir los resultados.

También existe un caso de uso para cuando no hay horarios disponibles en la base de datos.

El archivo de pruebas *ask_test.py* está en un repositorio y puede ser clonado utilizando esta dirección: https://willywilly@bitbucket.org/willywilly/contextobot_test.git

Capítulo 8

Conclusiones

En este capítulo se expondrán las conclusiones del proyecto, que constarán de la comparación entre la planificación previa al desarrollo del proyecto y lo que se ha logrado realmente en el trabajo, los planes de futuro y, finalmente, una conclusión personal.

8.1. Comparación entre la planificación previa al desarrollo y los objetivos logrados

En términos generales, el desarrollo del proyecto ha seguido la trayectoria establecida aunque igualmente, se han encontrado percances y no todos han podido resolverse.

8.1.1. Objetivos

Los objetivos propuestos durante la planificación del proyecto han sido cumplidos en su mayor medida. La razón de que no hayan sido cumplidos del todo es que durante el desarrollo del proyecto se detectó un percance con el servicio de conversión de audio a texto. El problema está en que el servicio está configurado para detectar el idioma castellano y alguno de los nombres de lugares son nombres en euskera. Esto se deriva a situaciones en las que el *bot* sea incapaz de detectar el origen y/o destino del usuario en cuanto intente pedirlos por medio de una nota de voz.

Actualmente, la única solución a ello está en que el usuario escriba la consulta en lugar de pedirla por medio de un audio.

8.1.2. Herramientas utilizadas

Se han tenido que utilizar más herramientas de las que inicialmente se pensaron, una de ellas para solucionar un problema de compatibilidad a la hora de cumplir con los requisitos de uno de los servicios utilizados (el de *Microsoft Cognitive Services*) y la otra, por causas de querer aumentar la eficacia de la muestra de resultados del *chatbot*.

FFmpeg

Una plataforma que ofrece un surtido completo de funcionalidades para conversión de archivos de audio y vídeo. Utilizada para convertir audios al formato deseado y así poder cumplir con los requisitos impuestos por los servicios que utiliza el proyecto.

PHP Image Magick

Un paquete de PHP que permite la creación desde cero de varios tipos de archivos de imagen. Permite crear efectos, aplicar filtros e incluso crear archivos GIF.

8.1.3. Planificación temporal

La duración del desarrollo del proyecto ha resultado ser un poco más larga que la estimación realizada inicialmente. La causa de esto es que se ha tenido que ampliar ligeramente la fase de aprendizaje para analizar las herramientas adicionales. En el resto de partes planificadas, se ha invertido un tiempo similar al establecido previamente.

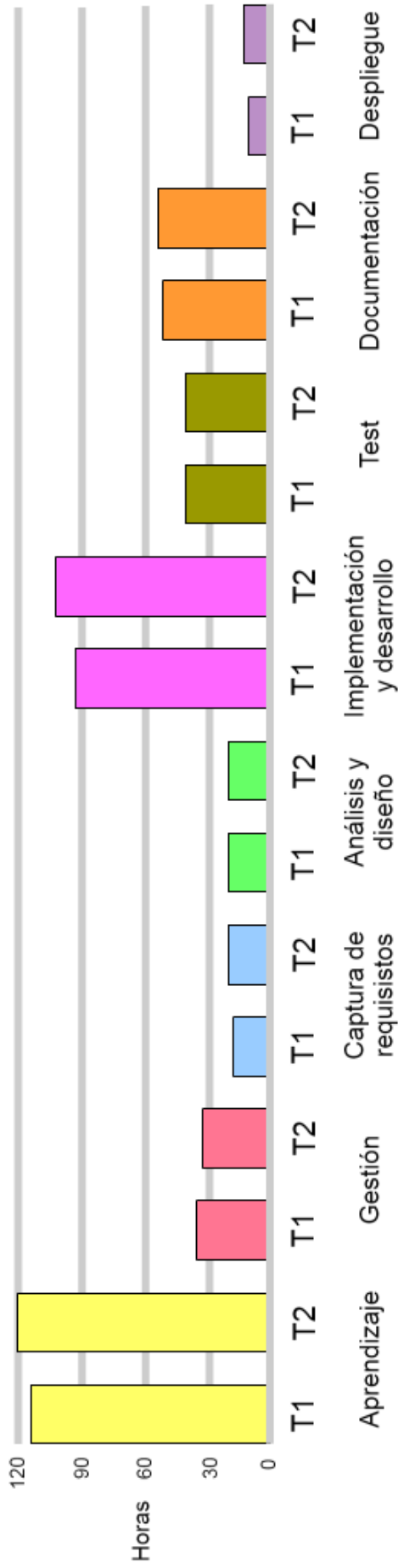

FIGURA 8.1: Gráfico que muestra la comparativa entre el tiempo a invertir calculado para cada tarea del proyecto en la planificación temporal y el tiempo real invertido.

8.2. Planes de futuro

Durante el desarrollo del proyecto, han surgido ideas de funcionalidades que vendrían muy bien a un *chatbot* como el que se ha desarrollado, así como la intención de ampliar el numero de servicios de transporte público, aumentando así el territorio en el que el *bot* pueda ser útil.

8.2.1. Consulta de saldo

Se ha creído oportuno que en el futuro, el *chatbot* tenga la capacidad de realizar una consulta de saldo a algúna de las tarjetas de transporte público que el usuario pueda tener, como el servicio Barik, Bat, Mugi, etc.

Esto aumentaría enormemente la utilidad del *chatbot* y podría incluso llegar a realizar recargas de saldo ya que Telegram ha añadido la funcionalidad de realizar pagos a los *bots*.

8.2.2. Ampliación de los horarios

Se ha dispuesto la estructura del proyecto para que añadir más servicios de transporte público (por ahora solo tiene uno) sea relativamente fácil. Se trataría de crear nuevos *spiders* que rastreasen distintas páginas de servicios de transporte público.

8.2.3. Detección de nombres en euskera

El inconveniente de utilizar tecnología de reconocimiento de voz está en que el idioma seleccionado es el Español y no contempla muchos de los nombres de ciudades vascas, por lo que tiene problemas para detectar algunas de ellas. Sería interesante trabajar en una solución para este problema, como podría ser un red neuronal entrenada para clasificar estas palabras.

8.2.4. Separar la ejecución del *crawler*

El *crawler* que hay implementado almacena la lista de lugares por una parte y la lista de horarios por otra. Sería algo bueno separar estas tareas para que las manejara una *spider* cada una por motivos de buena praxis.

8.3. Conclusión personal

Personalmente, opino que la realización de este TFG me ha aportado mucho y que me gustaría ver al *bot* con las funcionalidades extra que se han mencionado previamente.

Me ha interesado mucho aprender a utilizar las distintas tecnologías involucradas en el funcionamiento del proyecto. Me ha fascinado conocer los detalles del modo de trabajo que tienen los *spiders*, un tema del que había oído poco anteriormente y sobretodo, me alegro de ver que hoy en día es mucho más fácil de lo que creía tener acceso a servicios que incrementan enormemente el potencial de un programa como lo es un *bot*, hasta el punto de creer que estamos muy cerca de ver toda una generación de inteligencias artificiales que realicen gran parte de nuestras tareas por nosotros.

Por otro lado, agradezco mucho la ayuda de Juanan en los momento en los que la instalación de bibliotecas o la configuración de las mismas me han supuesto varios momentos con dolor de cabeza, sin su ayuda probablemente hubiese tardado mucho más en terminar el proyecto.

Apéndice A

Manual de instalación

En este capítulo se redactará una breve descripción de los requisitos para instalar el proyecto en un ordenador, así como un pequeño manual explicando los pasos a seguir para su correcto funcionamiento.

A.1. Requisitos previos

Para el correcto funcionamiento del proyecto, se requiere de la instalación de cierto *software* previa a la primera ejecución de cualquiera de las partes del proyecto (el *web crawler* o el *chatbot*).

PHP 7.0

La versión con la que se ha desarrollado el cógido del *chatbot* y los *scripts* para la actualización del agente. Es uno de los pilares básicos para gran parte de las aplicaciónes de un servidor.

Apache2

El otro de los pilares básicos es tener un servidor en el que resida la aplicación. Para ellos se utilizará apache2.

MySQL 5.7

La versión de MySQL que se ha utilizado para el desarrollo del proyecto. Si se tiene esta versión se asegura que MySQL no de problemas de compatibilidad.

cURL

La biblioteca cURL de PHP que se utiliza en el código del *chatbot* para contactar con los servicios externos como *Microsoft Cognitive Services* y *api.ai*.

Cuenta en Microsoft

Para disponer de una clave de acceso a la API de conversión de audio a texto de *Microsoft Cognitive Services*, se requiere de una cuenta.

Cuenta en Google

Se necesitará una cuenta de Google con la que acceder a *api.ai* y poder importar ahí el agente de reconocimiento del lenguaje natural.

FFmpeg

Esta biblioteca auxiliar es necesaria para que funcione el reconocimiento de voz del *chatbot* ya que transforma los archivos de audio enviados por el usuario a un formato que cumpla con las necesidades impuestas por el servicio de *Microsoft Cognitive Services*.

PHP Image Magick

Necesaria para mostrar los resultados del *chatbot* una vez que realice una consulta con éxito. Esta biblioteca genera las imágenes con los datos resultantes de la consulta.

Git

Es la herramienta a utilizar para clonar las partes del proyecto y disponer de todo el código. Se utilizará para clonar la parte de Scrapy y la parte de el *bot* de Telegram.

Telegram

La aplicación sobre la que funciona el proyecto, Telegram es necesario para crear la instancia del *bot* por medio de otro *bot* propio de Telegram llamado *botfather*. También se utilizará para interaccionar con el *chatbot* una vez esté todo en funcionamiento.

Python

Necesario para ejecutar Scrapy, ya que éste está desarrollado en este lenguaje de programación.

Scrapy

Es necesario tener Scrapy instalado para poder ejecutarlo y que así los *spiders* puedan hacer su trabajo obteniendo información de las páginas web de los servicios de transporte público.

A.2. Instalación de la base de datos

Una vez se hayan instalado los requisitos para el funcionamiento del proyecto, se procederá a crear una base de datos en MySQL sobre la que se volcará el esquema de la base de datos del proyecto.

Para hacerlo, primero habrá que descargar la parte del proyecto que contiene el *chatbot* y para hacerlo se podrá usar Git. Lo que se hará será clonar el repositorio en el que está almacenada esa parte del proyecto mendiante el siguiente comando:

```
git clone https://willywilly@bitbucket.org/willywilly/contextobot.git
```

Una vez hecho esto, se podrá acceder al esquema de la base de datos en la carpeta *database_schema*. Ya solo quedaría crear una base de datos con el nombre deseado y ejecutar el siguiente comando tras situarnos dentro de la carpeta *database_schema* mediante el comando *cd*:

```
mysql -u usuario -p nombredebasededatos < contextobot.sql
```

Tras ejecutar este comando, la base de datos recién creada tendrá la estructura de la base de datos que requiere el proyecto.

A.3. Importación del agente en *api.ai*

El siguiente paso será la importación del agente en *api.ai*. Habrá que iniciar sesión en la siguiente dirección:

```
https://console.api.ai/api-client/#/login
```

Tras hacerlo, habrá que crear un nuevo agente siguiendo los pasos que la página dará para hacerlo. En el apartado del idioma del agente, habrá que seleccionar el español y después habrá que pulsar el boton *SAVE* en la parte superior de la página.

Cuando se haya creado el agente habrá que dirigirse a la parte izquierda de la pantalla y clicar en el engranaje junto al nombre del agente que acaba de ser creado. A continuación, en el menú que habrá aparecido en el centro de la pantalla, habrá que pulsar en la pestaña *Export and Import* y hacer clic en *IMPORT FROM ZIP*. Finalmente habrá que seleccionar el archivo *Contextobot.zip* que fue descargado en la carpeta *api.ai_agent* que vino al clonar la parte del proyecto del *chatbot* y pulsar en *SAVE*.

Como paso adicional, conviene guardar la clave que aparecerá junto con el agente creado, ya que se utilizará en el archivo de configuración al finalizar el manual.

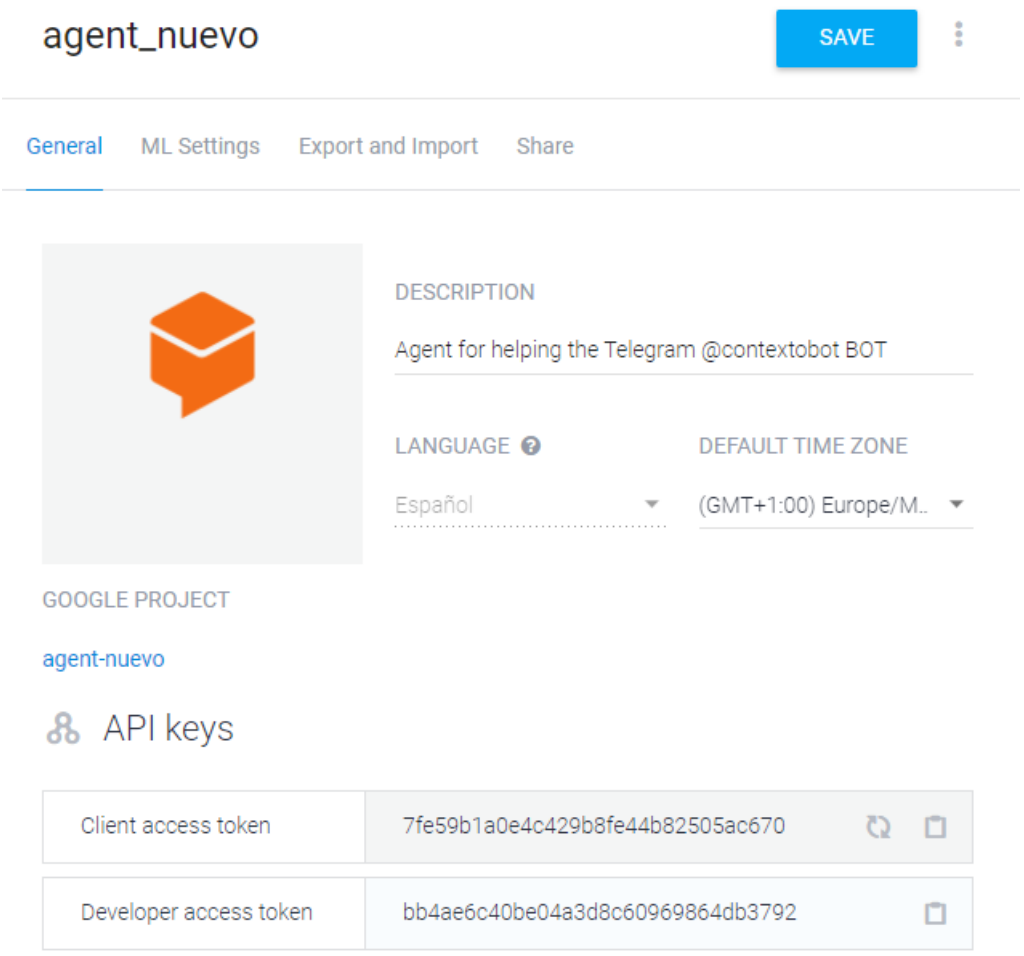

FIGURA A.1: Muestra el panel de control de *api.ai* mostrando las claves que serán generadas con el agente.

A.4. Creación del *bot* de Telegram

Una vez que se llega a este punto, se tendrá el código del *chatbot* en el repositorio del *chatbot* que fue clonado. También se dispondrá de un agente correctamente configurado en *api.ai* y de una base de datos en MySQL con la estructura deseada.

Para continuar, hará falta un *bot* en Telegram que se comunique con el usuario y para ello, se acudirá al *botfather*, un *bot* de Telegram encargado de generar nuevos *bots*. Para contactar con el *botfather* hace falta usar la aplicación de Telegram y hablar con él buscándolo a través de su alias "@BotFather".

Para solicitar que se genere un *bot* simplemente habrá que ejecutar el comando */newbot* y seguir los pasos que vaya indicando el *botfather*.

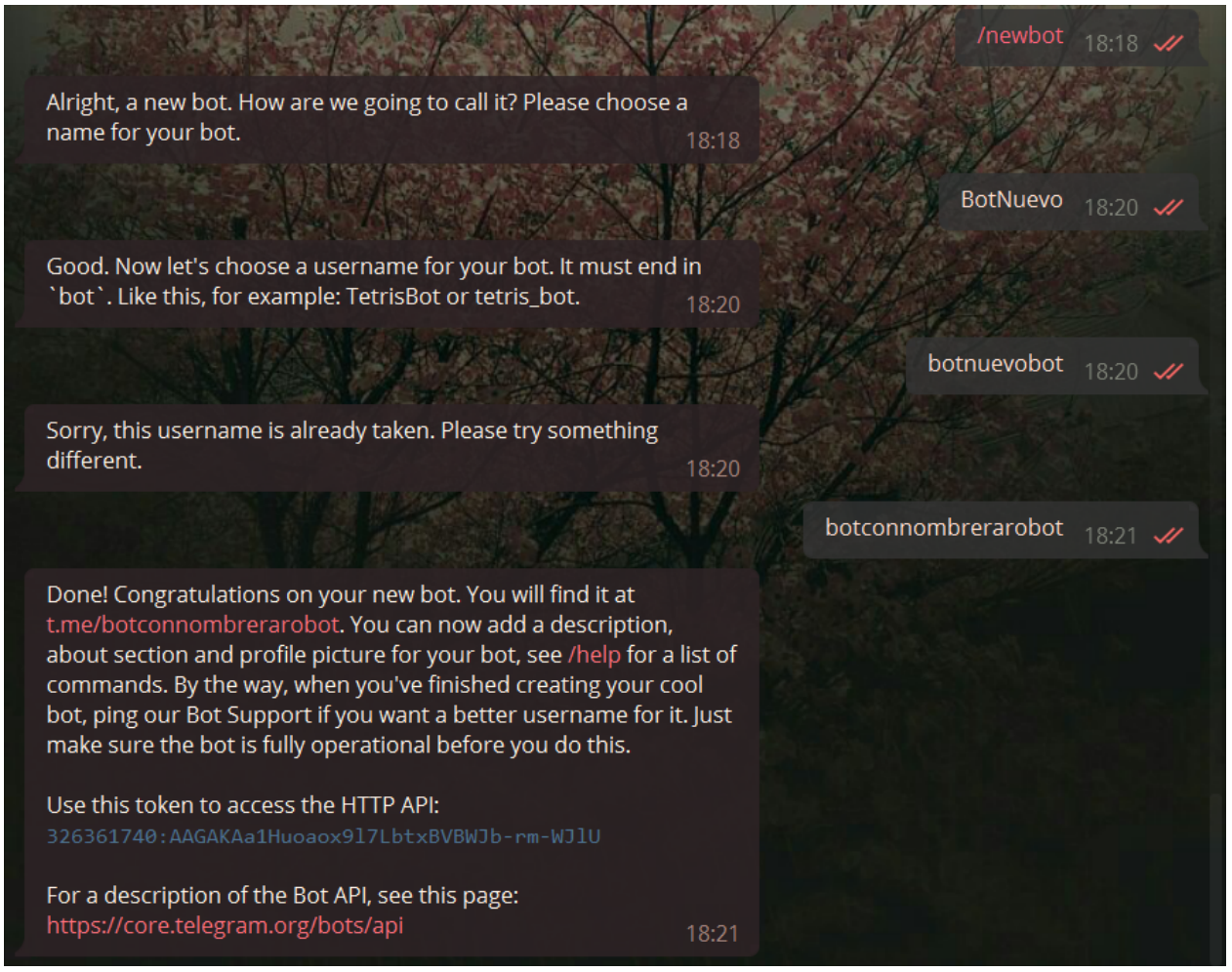

FIGURA A.2: Muestra cómo se crea un *bot* nuevo a través del *botfather* de Telegram.

Finalmente habrá que utilizar un edito de texto para configurar el archivo de configuración que reside en la carpeta del repositorio clonado. El archivo se llama *config.yml.tamplate* y habrá que introducir la información requerida en los campos sin dejar ninguno vacío. También habrá que quitarle la extensión *.template*.

A.5. Preparación de Scrapy

La parte del proyecto que concierne al *chatbot* ya está instalada y configurada, pero para que los datos de los horarios sean descargados y mantenidos al día, se requiere de la parte de los *web crawlers*. Para conseguirlo, se utilizará el mismo método que para descargar la parte del *chatbot*. Se usará Git para clonar esta parte del proyecto.

```
git clone https://willywilly@bitbucket.org/willywilly/contextobot_scrapy.git
```

Finalmente, hay dos archivos de configuración que habrá que rellenar, el primero se encuentra en la carpeta *naomiCRWL* bajo el nombre *data.py* y el segundo se encuentra en la carpeta *agent_updater* bajo el nombre *config.yml.template*.

Para rellenar algunos de los campos referentes a los identificadores de las entidades del agente de *api.ai* habrá que irse a la página del agente en *api.ai* e ir seleccionando las entidades para luego copiar el código de cada una que aparecerá en la url del navegador.

FIGURA A.3: Muestra cómo el identificador de la entidad seleccionada en el panel de control de *api.ai* se muestra en la URL.

A.6. Ejecución

Una vez se han completado todos los pasos anteriores, se puede probar el funcionamiento del *bot*. Lo primero que hay que hacer es ejecutar el *web crawler* ejecutando el script *Updater.php* que se encuentra en la carpeta *agent_updater* del sub-proyecto de Scrapy. Este script tambien se encargará de actualizar la lista de nombres para las entidades del agente en *api.ai*.

Hay dos maneras de hacer que el *bot* procese solicitudes provenientes del usuario y para usar cualquiera de las dos habrá que dirigirse a la carpeta raíz del sub-proyecto del *bot* de Telegram.

Archivo *getUpdatesCLI.php*

La forma de usar este archivo para procesar solicitudes es que se ejecute cada vez que se quiera que el *bot* procese las solicitudes que estén pendientes de procesar. Utilizar este archivo sólo es recomendable cuando se esta en fase de desarrollo ya que para su comportamiento normal, el *bot* debería procesar las solicitudes del usuario una a una al momento de recibirlas. Si se esta en fase de desarrollo se puede ejecutar manualmente el archivo *getUpdatesCLI.php* cada vez que se quiera procesar una o varias solicitudes de golpe, o bien, ejecutar el archivo mediante un ciclo *while*, por ejemplo, cada 4 segundos.

Uso de un *webhook*

Cuando el *bot* está listo pasa su uso en producción (con usuarios reales), es recomendable el uso de un *webhook* que asegura que el *bot* procese las solicitudes en cuanto las recibe. Para crear un *webhook* se ejecutará una vez el archivo *set.php*, que devolverá un mensaje comunicando el éxito de haber creado el *webhook*. Puede que la creación del *webhook* tarde unos minutos en hacerse efectiva pero a partir de entonces, el *bot* procesará al momento las solicitudes que se le envíen.

Apéndice B

Webs consultadas

Botnerds, *Types of Bots: An Overview*. Abril 2016.

URL http://botnerds.com/types-of-bots/

Wikipedia, *PHP*. Junio 2017.

URL https://es.wikipedia.org/wiki/PHP

PHP, *Introducción a cURL*. Junio 2017.

URL http://php.net/manual/es/intro.curl.php

Avtandil Kikabidze aka LONGMAN, *PHP telegram bot*. Abril 2017.

URL https://packagist.org/packages/longman/telegram-bot

@larknart, *How to launch getUpdatesCLI.php script in background*. Marzo 2017.

URL https://github.com/php-telegram-bot/core/wiki/%D0%9Dow-to-launch-getUpdatesCLI.php-script-in-background-(getUpdate-installation)%3F

W3schools, *XPath tutorial*. Junio 2017.

URL https://www.w3schools.com/xml/xpath_intro.asp

Bibliografía

[1] Fabrizio Romano. *Learning Python.* Packt Publishing, 2015.

[2] Dimitrios Kouzis-Loukas. *Learning Scrapy.* Packt Publishing, 2016.