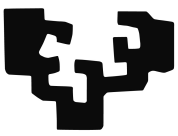
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

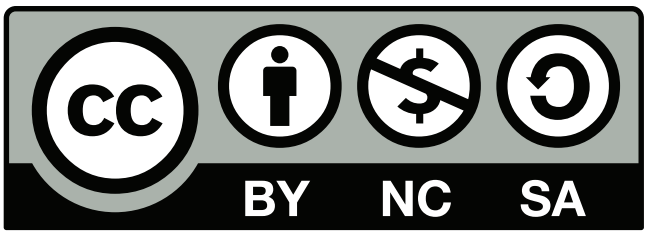
Proyecto de Fin de Grado

MAYORDOMO
Controla tu smarthome a través de la voz

Autor

Aitor Iturrioz Rodríguez

2017



Agradecimientos

Antes de nada, me gustaría agradecer a mis tutores, Rodrigo Agerri y Kepa Sarasola, el apoyo mostrado desde un primer momento cuando les propuse la idea de trabajar en el proyecto y por guiarme en el desarrollo del mismo.

A Tknika, por la flexibilidad ofrecida y por darme la oportunidad de crecer tanto a nivel profesional como personal. Sois un equipo fantástico.

Gracias a mis compañeros de clase y en especial a Markel, por la compañía y por los buenos momentos vividos.

Gracias a mi familia, por haberme dado la oportunidad de formarme, por sus ánimos y por no dejar de creer en mí en ningún momento.

Y finalmente, gracias a ti, Tamara, por tu paciencia, por estar a mi lado cuando más te necesitaba y por ayudarme incondicionalmente. Nada de esto habría sido posible sin ti, te quiero.

Resumen

La conectividad y la electrónica de bajo coste están posibilitando que cada vez más elementos de nuestra casa se vuelvan inteligentes y puedan ser controlados remotamente. Hasta ahora, la única forma de gestionar estos dispositivos era a través de aplicaciones móviles, pero últimamente están proliferando cada vez más sistemas basados en la voz.

Mediante este proyecto se pretende construir una plataforma de código abierto que permita interactuar con los elementos domóticos del hogar de forma sencilla. Además, tendrá que ser modular para que su uso pueda extenderse a otros ámbitos y tener la capacidad de procesar comandos en diferentes idiomas, convirtiéndose así en la primera plataforma en poder ser usada íntegramente en euskera.

De forma trasversal, el proyecto requerirá trabajar con tecnologías lingüísticas y permitirá aprender y profundizar en las técnicas empleadas en el procesamiento de lenguaje natural.

Palabras clave: Home Automation, NLP, TTS, STT, Freeling, IXA-PIPES, WordNet.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

1.1. Motivación

En los últimos años, debido a la disminución del precio de la electrónica y al auge de los chips de radio de bajo consumo, el mercado del IoT (Internet of Things) ha crecido de forma exponencial acercando a los usuarios finales productos capaces de conectarse a su red local y de ser controlados de forma remota. Sin embargo, este "boom" de diferentes productos y tecnologías ha creado un problema de interoperabilidad entre dispositivos. El usuario final se ve obligado a ceñirse a una única tecnología o a resignarse a usar sus dispositivos desde diferentes aplicaciones, debido a la nula interoperabilidad antes mencionada. Afortunadamente, nuevas plataformas open source como openHAB, Home Assistant, Domoticz,... han surgido para intentar solventar este problema creando interfaces de usuario unificadas que permitan a los usuarios controlar todos sus productos desde una única aplicación, normalmente desde dispositivos móviles como smartphones o tablets.

De forma paralela, una de las ramas de las ciencias de la computación que más ha evolucionado es la relacionada con el procesamiento del lenguaje natural o PLN. Tradicionalmente, los lingüistas y los informáticos han dividido el estudio del lenguaje en 6 niveles: fonética, fonología, morfología, sintaxis, semántica y pragmática [1]. Tras muchos años de investigación y el desarrollo de nuevos algoritmos, los analizadores morfosintácticos actuales son capaces de etiquetar correctamente las palabras de una oración con una precisión superior al 90 % [2]. El gran rendimiento de estos analizadores ha posibilitado la

creación de programas informáticos que, centrándose en el nivel de semántica, pueden interactuar con los usuarios de forma más natural e intuitiva mediante el lenguaje natural.

Combinando ambos desarrollos, la idea principal de este proyecto consiste en desarrollar un sistema de interacción humano-computador basado en lenguaje natural y aplicado al hogar. Los sistemas comerciales existentes (Amazon Echo, HomeKit, Google Home,...) tienen la limitación de ser sistemas propietarios, cerrados y no ampliables, por lo que el sistema que se pretende construir debe ser una plataforma modular, open source y abierta a futuras mejoras.

1.2. Contenido del documento

El presente documento está organizado de la siguiente forma:

1. El capítulo 2 muestra la gestión del proyecto desde el punto de vista administrativo: su alcance, análisis DAFO, EDT y la gestión de recursos.
2. El capítulo 3 realiza un pequeño análisis del estado del arte de las tecnologías involucradas en el proyecto.
3. Los capítulos 4, 5 y 6 describen las características y la implementación de la solución propuesta.
4. El capítulo 7 analiza las herramientas relacionadas con el procesamiento de lenguaje natural empleadas en el trabajo.
5. El capítulo 8 ofrece una guía para ampliar la plataforma, tanto para añadir nuevos idiomas a la misma como para desarrollar nuevos plugins.
6. El capítulo 9 presenta los resultados obtenidos del experimento realizado con el programa.
7. El capítulo 10 resume el trabajo realizado, las conclusiones extraídas y las posibles mejoras a realizar en la plataforma.

2. CAPÍTULO

Gestión del proyecto

En este capítulo se va a describir la gestión del proyecto, comenzando por el alcance, siguiendo con el análisis DAFO (debilidades, amenazas, fortalezas y oportunidades), continuando con el EDT y acabando con un análisis de los recursos invertidos en el mismo.

2.1. Alcance

El alcance del proyecto viene dado por las siguientes características de la plataforma:

- Open source: la plataforma debe tener una licencia libre y el código se subirá a GitHub para que cualquiera pueda colaborar en el desarrollo del programa.
- Comandos básicos para el control de los elementos del hogar: el software deberá incluir un módulo que sea capaz de actuar y recibir información del hogar a través de unos comandos básicos.
- Facilidad para añadir comandos nuevos al sistema: la plataforma será lo suficientemente flexible para que un desarrollador pueda implementar nuevos comandos de forma sencilla.
- Contexto lingüístico: se entiende el contexto lingüístico como la capacidad del programa para recordar comandos previos del usuario.

- Capacidad para requerir información adicional al usuario en caso de ambigüedad: Mayordomo deberá ser capaz de interactuar con el usuario cuando no sea capaz de interpretar un comando en su totalidad.
- Multilenguaje: el software deberá ser capaz de procesar comandos en diferentes idiomas para ser lo más universal posible.

2.2. Análisis DAFO

En esta sección se presentan los puntos fuertes y débiles del proyecto a través del análisis DAFO.

2.2.1. Debilidades

Como principales debilidades del proyecto se pueden destacar las siguiente:

- La complejidad del proyecto, ya que se deben aunar en una misma plataforma sistemas de conversión de voz a texto, texto a voz, lematización de oraciones, etc. y que todo funcione en un entorno real.
- La ambición del alcance, donde se puede destacar principalmente el hecho de que el sistema deba ser capaz de procesar comandos en diferentes idiomas.
- La ambigüedad inherente a cualquier sistema relacionado con el procesamiento del lenguaje natural.

2.2.2. Amenazas

Compatibilizar el proyecto con el trabajo a jornada completa es la mayor amenaza del mismo. Gestionar los horarios, el cansancio y, sobretodo, mantener un nivel de motivación constante serán muy importantes para poder llevar a cabo el proyecto.

2.2.3. Fortalezas

El proyecto presenta varias fortalezas clave que favorecen su éxito:

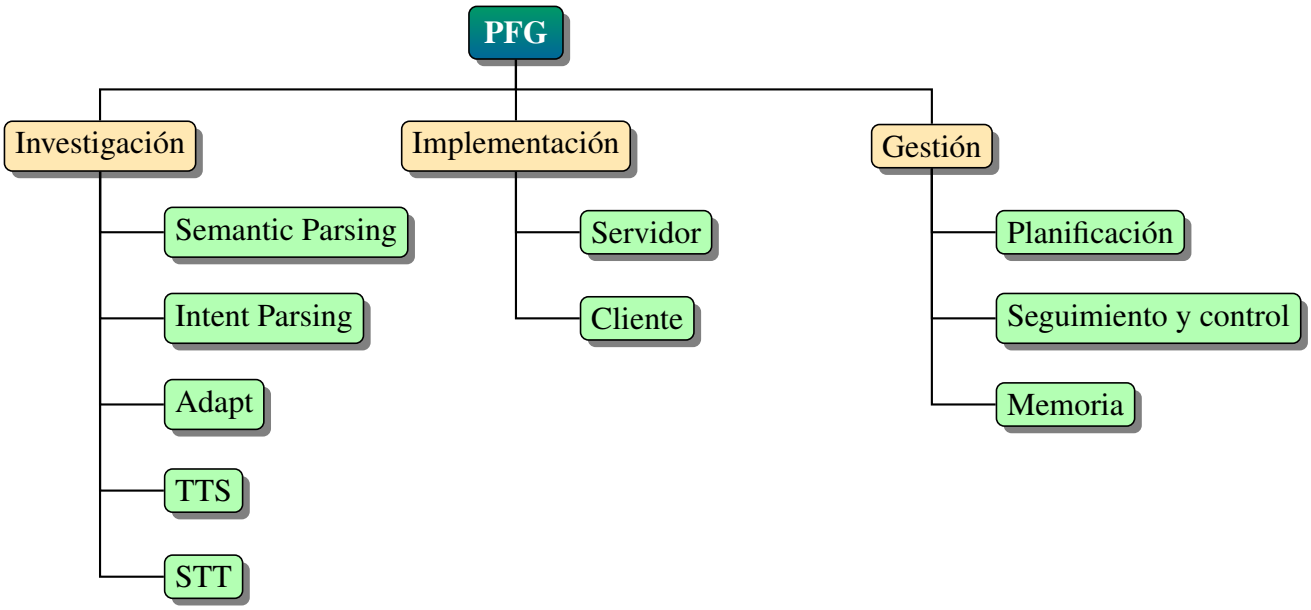
- El tema y la idea del proyecto fueron propuestos por mí, lo que posibilita una mayor implicación personal.
- El hecho de que la versión final del software permita controlar elementos del mundo real (luces, enchufes, temperaturas...) supone un aliciente extra.
- Los asistentes controlados por la voz están muy de moda hoy en día.

2.2.4. Oportunidades

Como oportunidades más importantes del proyecto pueden ser destacadas:

- Trabajar y profundizar en tecnologías relacionadas con el NLP. Entre ellas, la conversión de texto a voz (TTS), voz a texto (STT) y la tokenización y lematización de oraciones.
- Liberar el código del proyecto y subirlo a una plataforma como GitHub permite devolver a la comunidad open source una parte de lo recibido durante estos años.
- La posibilidad de implementar la versión final de mayordomo en el centro de trabajo también supone una gran oportunidad.

2.3. EDT



2.4. Gestión de recursos

A continuación se presentan varias tablas que resumen el número de horas dedicadas al proyecto en cada categoría:

Tarea	Dedicación (horas)
Semantic parsing	5
Slot filling	5
Intent parsing	20
Adapt	20
Herramientas TTS	10
Herramientas STT	10

Tabla 2.1: Tiempos de dedicación a la investigación.

Tarea	Dedicación (horas)
Arquitectura del sistema	15
Módulo MQTT	10
Módulo Adapt	20
Módulo traductor	15
Módulos NLP (ixa-pipes y freeling)	20
Módulos TTS	15
Módulo openHAB	20

Tabla 2.2: Tiempos de dedicación a la implementación del servidor.

Tarea	Dedicación (horas)
Arquitectura del sistema	10
Módulo MQTT	10
Módulo STT	15
Módulo Snowboy	15

Tabla 2.3: Tiempos de dedicación a la implementación del cliente.

Finalmente, la tabla 2.5 muestra el número de horas de cada categoría y el número de horas totales dedicadas al proyecto.

Tarea	Dedicación (horas)
Planificación	20
Seguimiento y control	10
Redacción de la memoria	60

Tabla 2.4: Tiempos de dedicación a la gestión.

Categoría	Dedicación (horas)
Investigación	70
Implementación Servidor	115
Implementación Cliente	50
Gestion	90
Total	325

Tabla 2.5: Tiempos de dedicación total.

3. CAPÍTULO

Estado del arte

A través de este capítulo se pretende hacer un pequeño resumen de los papers publicados alrededor de las tecnologías empleadas hoy en día para procesar comandos relacionados con la automatización del hogar.

3.1. General

En el artículo "Learning Semantic Parsers for If-This-Then-That Recipes" [3], un grupo de investigadores de Microsoft desarrolla un método para convertir frases expresadas en lenguaje natural en reglas para la plataforma IFTTT. IFTTT (if-this-then-that) es un servicio web en el que los usuarios pueden crear reglas de automatización, donde se ofrecen múltiples condiciones y acciones para las reglas. Las condiciones pueden ser: subir un fichero a Drive, publicar un mensaje en Facebook, detectar movimiento en una casa, recibir una llamada en el móvil, conectarse a una red WiFi,... mientras que las acciones pueden ser: enviar un sms, encender una luz de casa, subir una foto a Facebook... Nuevas condiciones y acciones pueden ser añadidas mediante plugins. El trabajo realizado por los investigadores ha consistido en recoger las más de 100.000 reglas presentes en la web (las reglas incluyen una breve descripción de su funcionalidad) y analizar las descripciones presentes para enlazar ambos conceptos. Las técnicas empleadas consisten en convertir las descripciones en árboles sintácticos abstractos, aplicar clasificadores basados en regresiones logísticas (unigramas, bigramas y trigramas como características) y entrenar el clasificador de forma iterativa.

El artículo “Learning to Transform Natural to Formal Languages” [4] presenta un método para convertir expresiones en lenguaje natural en un lenguaje formal. Describir las expresiones en un lenguaje formal permite realizar consultas (“queries”) de forma más sencilla. Para la conversión, se emplean dos métodos diferentes. El primero emplea cadenas de caracteres como patrones y los empareja directamente con frases en lenguaje natural. En el segundo, en cambio, los patrones son árboles (con palabras y marcadores sintácticos) y se emparejan con árboles sintácticos de las frases en lenguaje natural. Finalmente, los desarrolladores prueban el algoritmo desarrollado sobre CLANG, un lenguaje formal empleado en competiciones de fútbol entre robots.

Por último, en el paper “Semantic Validation of Uttered Commands in Voice-activated Home Automation” [5] se describe un sistema de reconocimiento de voz para la automatización del hogar. El trabajo realizado puede dividirse en dos partes: el sistema de captura y procesamiento del audio y la parte del programa que determina la validez del comando en función de la información del dominio disponible. Para la primera parte, se emplea la técnica MFCC (Mel-frequency cepstral coefficients), que convierte el espectro lineal de la voz en una escala no-lineal llamada Mel. Usando modelos ocultos de Markov se convierte la entrada en secuencias de fonemas y la probabilidad final de cada palabra se calcula con un “modelo de lenguaje” previamente establecido (corpus de frases en el que cada frase se ha clasificado en trigramas). El modelo ha sido entrenado con 51 horas de audio y más de 500.000 de palabras. Para la segunda parte, debido al limitado vocabulario a reconocer, se crea una ontología a mano con los dispositivos a controlar y la relación entre los dispositivos y sus características. Esta ontología es la que se usa para validar los comandos reconocidos por el sistema de reconocimiento de voz.

Como se ha podido ver, la mayoría de las técnicas actuales se basan en la disponibilidad de un gran corpus con frases de ejemplo al que se le aplica un tratamientos estadístico, con el fin de extraer las características principales que definen el sistema. Pero para el tema aquí tratado, debido a la limitación de los comandos esperados y a la falta de un corpus establecido, se ha optado por resolver los comandos a través de un algoritmo de búsqueda de palabras clave.

3.2. Intent parsing

Antes de comenzar a describir qué servicios o librarías existen hoy en día para determinar la acción a ejecutar por parte del usuario, conviene definir una serie de conceptos a los

que se hará referencia a lo largo del presente documento.

3.2.1. Vocabulario

- Utterance: se define un "utterance" como la transcripción a texto de la frase expresada por el usuario. Por ejemplo: "¿puedes encender la luz de la cocina?".
- Intent: un "intent" es la acción (solicitar información o controlar un elemento) que subyace del utterance. Siguiendo con el ejemplo previo, el intent asociado al utterance dado se podría definir como "set_light_state".
- Entity: los "entities" son las palabras clave (entidades) a buscar en el utterance y desempeñarán el papel de parámetros para el intent. En el ejemplo anterior se podrían identificar dos entities: "light_state", cuyo valor sería "encender", y "home_room", con el valor "cocina".

3.2.2. API.AI

Figura 3.1: Logo de *API.AI*.

API.AI [6] es una plataforma de creación de bots (o "Agentes" según su propia terminología) conversacionales recientemente adquirida por Google (en Septiembre de 2016) y que sigue los mismos principios descritos en el punto anterior. Se describen a continuación sus principales características:

- Configuración del sistema mediante una interfaz web.
- Creación de intents y entities personales, así como la utilización de intents y entities predefinidos para diferentes ámbitos.
- REST API bien documentada, lo que permite hacer uso del sistema desde cualquier lenguaje de programación.

- Posibilidad de exportar los agentes creados a diferentes plataformas: Slack, Facebook Messenger, Google Assistant...
- Gratuito, siempre que los “Agentes” sean públicos.

Entre las desventajas más notables destacan:

- Como en cualquier servicio web, se depende de la disponibilidad y fiabilidad de un proveedor externo.
- No se tiene acceso al código fuente del servicio, por lo que no es posible implementar nuevas funcionalidades.
- A la hora de crear un “Agente” es necesario definir de antemano su idioma, por lo que no es posible crear “Agentes” multilenguaje.
- A pesar de que por ahora es gratuito, las condiciones del servicio pueden cambiar en cualquier momento y pasar a ser una plataforma de pago.

3.2.3. WIT.AI

Figura 3.2: Logo de *WIT.AI*.

WIT.AI [7], de forma similiar a API.AI, es una plataforma web orientada a la creación de bots conversacionales, aunque en este caso la dueña del servicio es Facebook (desde Enero de 2015).

Comparte la mayoría de pros y contras con API.AI, aunque se diferencian en:

- En WIT.AI todas las apps (el equivalente a los “Agentes” de API.AI), tanto públicas como privadas, son gratuitas.
- Soporta más idiomas para los entities e intents predefinidos, pero no sigue siendo posible crear apps en varios idiomas simultáneamente.

- No ofrece la opción de exportar nuestras apps a plataformas como Slack, Facebook messenger, Google Assistant, etc.
- A pesar de que el código fuente de la plataforma no está disponible, el equipo de desarrollo ha liberado la herramienta que usan para procesar expresiones temporales en varios idiomas: duckling.

3.2.4. SEMPRE

SEMPRE [8], tal y como se indica en su página web, es un conjunto de herramientas para entrenar analizadores semánticos, los cuales, haciendo uso de expresiones lógicas, relacionan los "utterances" en lenguaje natural con respuestas.

La herramienta incluye una copia completa de Freebase (una base de conocimiento colaborativa) con 41 millones de "entities" y 596 millones de relaciones entre "entities". Esto hace que SEMPRE sea una herramienta ideal para crear sistemas de pregunta-respuesta de carácter general, ya que Freebase contiene una gran cantidad de información acerca de instituciones, capitales, poblaciones, áreas, distancias, alturas, etc. del mundo.

3.2.5. ADAPT

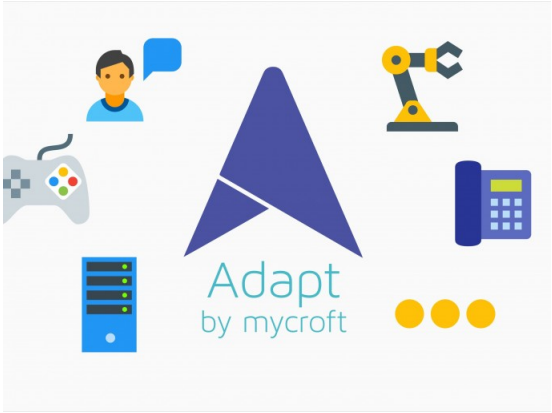


Figura 3.3: Logo de *Adapt*.

ADAPT [9] es una librería open source desarrollada en Python cuya función principal es transformar "uterances" de lenguaje natural en estructuras JSON con los "entities" detectados. Desarrollada por la empresa Mycroft, es una librería liviana pensada para ser

ejecutada en dispositivos con recursos limitados y permite definir “entities” e “intents” de forma sencilla.

Su flexibilidad y licencia open source, además del hecho de estar escrita en Python -que es el lenguaje de programación en el que estará escrito Mayordomo-, hacen de esta librería el candidato ideal sobre el que basar el núcleo del programa.

4. CAPÍTULO

Mayordomo plataforma

Después de establecer el alcance del proyecto, los requisitos del sistema y del análisis del estado del arte presentado en la sección anterior, el presente capítulo pretende establecer una introducción a la plataforma, su arquitectura general y el protocolo de comunicación empleado.

4.1. Introducción

"Mayordomo" es una plataforma de software libre, escrita en Python 2.7, que permite a cualquier usuario controlar los elementos de su sistema domótico de forma sencilla mediante comandos de voz. Además de poder controlar luces, enchufes o radiadores, y de consultar temperaturas, humedades o potencias, permite extender su funcionalidad mediante "plugins", a través de los cuales se pueden implementar nuevos comandos.

Debido a la ubicuidad de los sistemas informáticos actuales, en la que el usuario tiene a su disposición diferentes dispositivos electrónicos tales como ordenadores portátiles, teléfonos móviles, relojes inteligentes o tablets, el programa presenta una arquitectura modular siguiendo el patrón cliente-servidor.

4.2. Arquitectura

Tal y como se acaba de mencionar en el punto anterior, la plataforma "Mayordomo" está dividida en dos secciones: un programa principal, al que llamaremos servidor, y uno o varios programas auxiliares, a los que llamaremos clientes.

La idea principal es que haya un único servidor para cada casa y que éste se ejecute en un dispositivo embebido de bajo consumo, como una Raspberry Pi. La Raspberry Pi es un ordenador de placa reducida (o SBC -single board computer- por sus siglas en inglés) desarrollado por la fundación Raspberry Pi, cuyo objetivo inicial era fomentar la enseñanza de programación en las escuelas de Gran Bretaña. Gracias a su versatilidad (es capaz de ejecutar diferentes distribuciones de Linux) y bajo precio (~40€) su popularidad creció rápidamente y hoy en día es todo un referente en la comunidad maker. En la red podemos encontrar numerosas webs donde se presentan proyectos relacionados con este pequeño ordenador: desde emuladores de máquinas arcade hasta clusters de más de 80 dispositivos, pasando por estaciones meteorológicas o impresoras 3D. Por todo ello, este mini PC es el candidato ideal para albergar la parte servidor de nuestro programa.

Los clientes, en cambio, deben poder instalarse en la mayor cantidad de dispositivos posibles, para que el usuario tenga acceso a las funcionalidades ofrecidas por la plataforma en cualquier momento. Para esta primera versión del programa, que es la que abarca este proyecto, se ha desarrollado un cliente de escritorio en Python compatible con cualquier dispositivo con sistema operativo Linux. Para la segunda versión se prevé desarrollar una aplicación nativa para Android escrita en Java.

El protocolo elegido para comunicar los clientes con el servidor es MQTT, debido a su velocidad, versatilidad y ligereza. En la siguiente sección se puede encontrar más información acerca del protocolo.

La imagen 4.1 que se muestra a continuación resume la arquitectura del sistema.

4.3. MQTT

MQTT [10] (Message Queue Telemetry Transport por sus siglas en inglés), es un protocolo de intercambio de mensajes que sigue el patrón publicar-suscribir (publish-subscribe [11]) sobre el protocolo TCP/IP. Es el estándar de facto en el mundo del IoT para crear

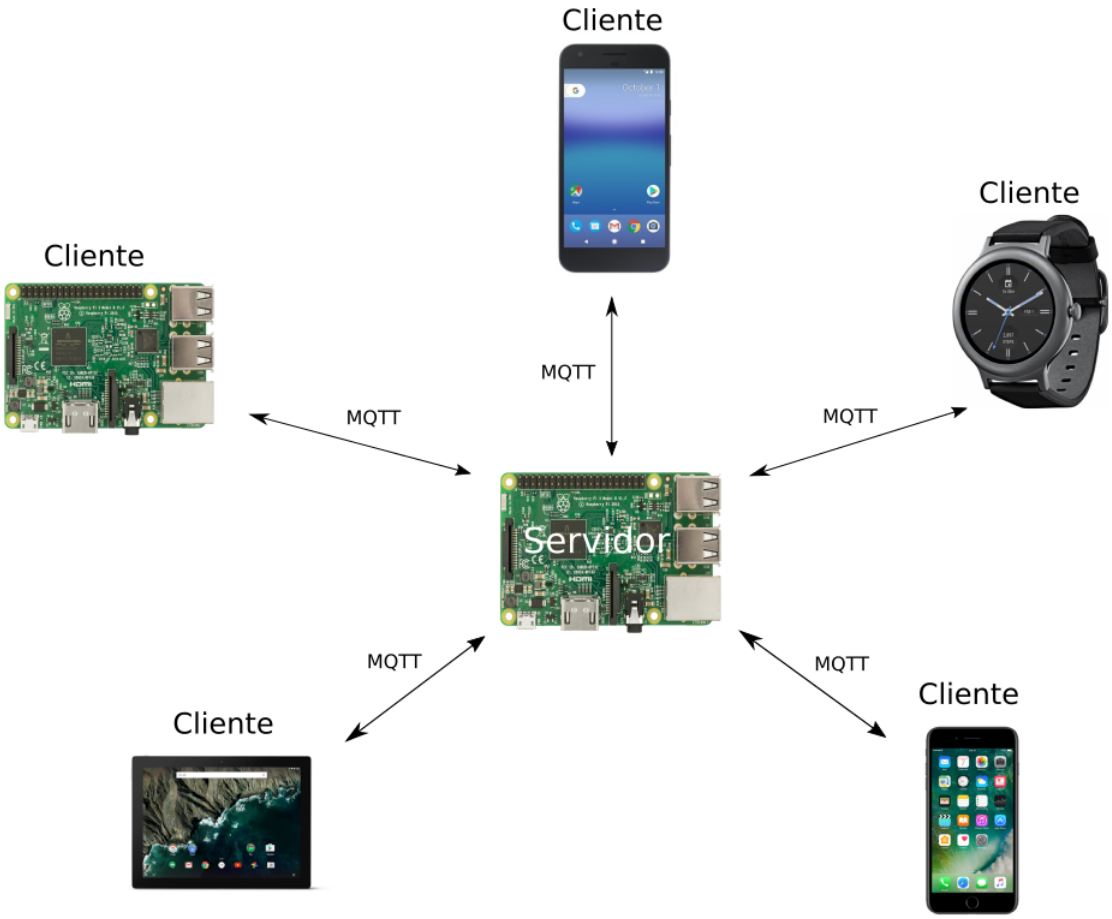


Figura 4.1: Arquitectura de la plataforma.

Figura 4.2: Logo de *MQTT*.

redes de sensores y actuadores en redes locales o a través de internet, gracias a su escalabilidad y a que consume muy poco ancho de banda.

4.3.1. Broker y clientes

MQTT es un protocolo centralizado, es decir, en toda red existe un servidor central que coordina las comunicaciones y diferentes clientes que se conectan a él para intercambiar mensajes. Se podría decir, por tanto, que su arquitectura sigue una topología de estrella.

En la jerga del protocolo, a este nodo central se le denomina "Broker", y existen numerosas soluciones open source disponibles en el mercado. Una de las más conocidas, y la empleada en este proyecto, se denomina "Mosquitto" [12].

Un cliente en una red MQTT puede ser cualquier dispositivo con conectividad IP, ya sea cableada o inalámbrica. En el mercado se pueden encontrar extensiones para navegadores, aplicaciones nativas para Android y iOS, clientes de escritorio o terminal, etc. que actúan como tales. Además, existen librerías para cualquier lenguaje de programación (C, Javascript, Java, Python,...) que facilitan la creación de aplicaciones basadas en un cliente MQTT. Para el desarrollo de la plataforma Mayordomo, tanto en la parte servidor como en el cliente, se ha hecho uso de la librería Eclipse Paho [13] en su versión para Python.

4.3.2. Suscribirse y publicar

Como se ha mencionado previamente, la forma en la que los clientes intercambian mensajes sigue el patrón publicar-suscribir. En una red MQTT los clientes no se comunican punto a punto, es más, un cliente no puede saber quiénes van a ser los destinatarios de su mensaje. Esto se debe a que los clientes deben enviar (publicar) sus mensajes a una especie de dirección, y los clientes que estén interesados en recibir dicha información deberán notificárselo al broker a través de una suscripción. El broker es el encargado de recibir el mensaje publicado y de redirigirlo a los clientes suscritos a esa dirección.

4.3.3. Topics y payloads

Las direcciones mencionadas en la sección anterior se denominan "Topics" en el mundo de MQTT. Estos topics no son más que cadenas de caracteres, con la particularidad de que pueden ser jerarquizados siguiendo una sintaxis muy similar a la de los directorios

del sistema de ficheros de Linux (Unix). La única diferencia con el sistema de carpetas de Unix es que en MQTT no es necesario que un topic empiece en el símbolo barra (“/”). Esta jerarquización permite que los clientes no tengan que suscribirse a todos los topics de forma individual, sino que puedan suscribirse a un topic de un nivel superior y recibir todos los mensajes que cuelgan de él.

Si un topic es la dirección a la que se envía un mensaje, el “Payload” es la información que contiene el propio mensaje. MQTT es completamente agnóstico respecto al tipo de información que se envía en el payload, ya sea desde el punto de vista del contenido (un dato de temperatura, de humedad, un comando,...) como de su formato (texto plano, json, imágenes en binario...).

4.3.4. Ejemplo

A fin de que los conceptos introducidos en esta sección queden más claros, a continuación se presentan una serie de ejemplos que pretenden clarificar la arquitectura de un sistema MQTT.

Suponiendo que en la red MQTT se describe una red de sensores (temperatura, humedad, luminosidad...) de una casa habitual (con una cocina, un comedor, un baño y dos dormitorios), la jerarquía de topics podría ser descrita de la siguiente forma:

- “casa” sería el topic raíz.
- “casa/cocina”, “casa/comedor”, “casa/baño”, “casa/dormitorio_1” y “casa/dormitorio_2” serían los topics que albergarían la información referida a cada estancia de la casa.
- Para cada uno de los topics anteriores, se añadiría un nivel adicional con cada una de las magnitudes medidas: temperatura, humedad, luminosidad, presencia, ruido... Así pues, la información sobre la temperatura de la cocina será publicada por el sensor en el topic “casa/cocina/temperatura”, y el nivel de ruido en el dormitorio 1 podrá ser consultado en “casa/dormitorio_1/ruido”.

De esta forma, un cliente MQTT podría recibir toda la información del sistema suscribiéndose simplemente al topic “casa/#”, ya que el símbolo “#” indica que se está interesado en todos los subniveles por debajo de ese topic. De forma complementaria, el símbolo “+”

sirve de comodín para un único subnivel. Un cliente al que sólo le interesasen los datos de temperatura de todas las habitaciones se suscribiría al topic “casa/+/temperatura”.

En los siguientes capítulos, cuando se analicen en profundidad cada una de las partes de la arquitectura, se describirán los topics definidos para el intercambio de mensajes entre el servidor mayordomo y los clientes.

5. CAPÍTULO

Mayordomo servidor

El programa “Mayordomo Servidor” [14] es el elemento principal de la plataforma desarrollada, ya que es el encargado de procesar todas las peticiones que le lleguen de los distintos clientes que se hayan instalado. Entre sus características se pueden destacar:

- Permite recibir y procesar comandos que lleguen de los clientes a través de MQTT.
- Interactua con un sistema domótico basado en openHAB para el control de los dispositivos del hogar.
- Mantiene un registro de los últimos comandos procesados que facilita autorellenar futuros comandos.
- Ofrece, a través de MQTT, un mecanismo para convertir texto a voz en varios idiomas.
- Puede funcionar en varios idiomas y permite conmutar entre los idiomas en tiempo real.
- Facilita la creación de nuevos plugins de forma sencilla.

Su desarrollo se ha basado en pequeños módulos que interactuan entre sí para dotar de mayor flexibilidad a la plataforma y para que el código estuviera lo más ordenado posible. Se ha elegido Python como lenguaje de programación debido, sobretodo, a la simplicidad de su sintaxis, lo que permite desarrollar programas en poco tiempo; la gran cantidad

de librerías externas que posee y al hecho de que puede ser ejecutado en la mayoría de sistemas operativos.

5.1. Arquitectura

La figura 5.1 muestra la arquitectura del servidor a través de sus módulos y la relación que hay entre ellos.

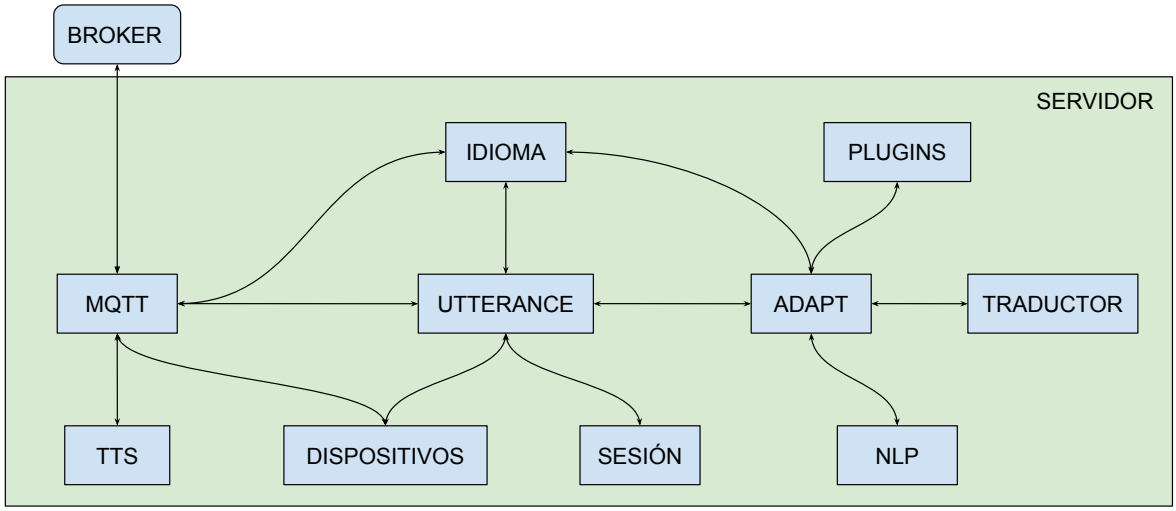


Figura 5.1: Arquitectura del servidor.

La funcionalidad y particularidades de cada módulo serán descritas a continuación.

5.2. Módulo MQTT

Este módulo (mqtt_handler.py) es el encargado de establecer la comunicación con el broker y de permitir al resto de módulos del sistema registrar sus propias suscripciones y publicar sus mensajes. Él no se suscribe a ningún topic, es una simple pasarela para que el sistema disponga de una única conexión con el broker y así el resto de módulos no tienen que gestionar sus propias conexiones.

5.3. Módulo Dispositivos

El módulo dispositivos (devices_handler.py) es el responsable de registrar los diferentes clientes de la red Mayordomo. Como se puede ver en el diagrama, el módulo hace uso del módulo MQTT, ya que se suscribe al topic "mayordomo/devices/register" a la espera de recibir mensajes de registro por parte de los clientes. Cuando un cliente Mayordomo desea registrarse, lo único que tiene que hacer es publicar un mensaje al topic "mayordomo/devices/register" con un objeto JSON que contenga los siguientes pares nombre/valor:

- "name": es el nombre del dispositivo y será usado como identificador.
- "description": breve descripción del dispositivo, su uso está orientado a ofrecer información adicional sobre el cliente Mayordomo.
- "owner": dueño o usuario habitual del dispositivo, este campo puede ser usado a la hora de dotar de contexto al sistema. Para clientes con usuarios múltiples, se recomienda usar el valor "unknown".
- "location": estancia del hogar en el que se va a situar el dispositivo y que se usará en el módulo de contexto. Para dispositivos móviles, se recomienda el valor "unknown".

El mensaje JSON que se muestra a continuación refleja el ejemplo de un registro válido:

```
{
    "name": "cliente_sala",
    "description" : "Raspberry Pi con el cliente Mayordomo",
    "owner" : "aitor",
    "location": "sala"
}
```

Cuando un cliente se registra en el sistema, debe suscribirse al topic "mayordomo/devices/<name>/answer", ya que el framework usará ese topic para devolver el resultado de los comando procesados.

5.4. Módulo Idioma

Este módulo (language_handler), como su nombre indica, se encarga de gestionar en todo momento en qué idioma está configurado el servidor. El idioma viene representado por la clase Language, que contiene los atributos "language", "region" y "code", y su representación está basado en el identificador BCP-47 [15]. Bajo este estándar, el castellano se identifica como "es-ES", el euskera como "eu-ES" y el inglés americano como "en-US". En cualquier caso, la clase Language provee un método llamado "get_language_code_3()" que devuelve el idioma del sistema bajo la normal ISO 639-3 [16].

Tal y como sucede en el módulo anterior, el módulo descrito en esta sección también hace uso del handler mqtt para modificar el idioma del sistema a través de MQTT. Cualquier cliente podrá conocer el idioma del sistema suscribiéndose al topic "mayordomo/configuration/language", topic al que le llegarán mensajes con el payload:

```
{
    "code": "es-ES",
    "language" : "es",
    "region" : "ES"
}
```

Asimismo, todo cliente será capaz de cambiar el idioma enviando un mensaje al topic "mayordomo/configuration/language/set" incluyendo un identificar BCP-47 válido en el payload del mismo. El framework notificará el cambio a todos los clientes mediante el topic "mayordomo/configuration/language" y volverá a cargar los módulos correspondientes.

5.5. Módulo Utterance

El módulo Utterance (utterance_handler) es el responsable de recibir y procesar los utterance, que son la transcripción a texto plano de los comandos dados por los usuarios. Para ello, nada más iniciarse, se suscribe al topic "mayordomo/utterance/message" haciendo uso del módulo MQTT y define una función callback que será llamada cuando alguno de los clientes publique un mensaje en dicho topic. Estos mensajes deberán incluir un objeto JSON en su payload con los pares:

- “device_name”: nombre del dispositivo, que previamente ha tenido que ser registrado, desde el que se envía el utterance.
- “user_name”: nombre del usuario, si el cliente es capaz de identificarlo, que envía el comando.
- “message”: transcripción a texto plano de la frase pronunciada por el usuario.

El siguiente objeto JSON es un ejemplo válido:

```
{
    "device_name": "cliente_sala",
    "user_name" : "aitor",
    "message" : "apaga la luz de la sala, por favor"
}
```

Estos son los pasos que sigue el módulo una vez ha recibido un mensaje JSON válido a través de MQTT:

1. Se comprueba si el dispositivo desde el que se ha enviado el mensaje, la clave “device_name” del JSON, es un dispositivo registrado.
2. Se comprueba si el usuario que ha enviado el mensaje, clave “user_name” del JSON, tiene una sesión activa en el sistema y, en caso afirmativo, se obtiene su contexto.
3. Se obtienen los lemas de cada una de las palabras del utterance, clave “message” del JSON, mediante la herramienta activa en el módulo NLP.
4. El utterance lematizado y el contexto previo del usuario son enviados al módulo Adapt para ser procesados y obtener el intent que mejor se ajuste a las palabras del usuario.
5. Sea cual sea el resultado devuelto por Adapt (error, incompleto o completo) se genera una notificación que se envía inmediatamente al dispositivo para que éste le devuelva la información al usuario.
6. Salvo que el resultado sea erróneo, se actualiza el contexto del usuario con los entities detectados en el utterance.

5.6. Módulo Adapt

El módulo Adapt (engine_handler) gestiona las comunicaciones con la librería del mismo nombre. Tal y como se ha descrito en el capítulo 3, Adapt es la librería usada para, dado un utterance, devolver el intent que más se aproxime a través de los entities reconocidos en la frase. Para ello, el sistema debe ser informado previamente sobre los entities que debe buscar y qué tipos de intents debe reconocer, pero el mecanismo para registrar esa información está reservada al módulo de los plugins, que será analizado más adelante.

Pese a que no se va a detallar el algoritmo usado por Adapt para resolver los intents, en esta sección sí que se van a describir de forma breve los tipos de entities que existen en la plataforma, el resultado devuelto por la función "determine_intent()" y las modificaciones realizadas a la librería para extender su funcionalidad.

En Adapt se pueden definir dos tipos de entities diferentes:

- "Entities estándar": son strings o palabras literales que construyen el vocabulario fijo de Adapt. Para que un entity estándar sea reconocido dentro de un utterance, debe aparecer tal y como se ha definido.
- "Entities regex": son entities definidos a través de una expresión regular y por tanto no están limitados a una única palabra. Muy útiles a la hora de detectar números o palabras que van precedidas o sucedidas de una palabra clave.

Cuando se quiera resolver el utterance dado por un usuario, el sistema hará uso de la función "determine_intent", la cual admite cuatro parámetros:

- "utterance": es el único de los parámetros que es obligatorio y, como indica su nombre, deberá contener el utterance a resolver.
- "num_results": parámetro opcional, determina el número máximo de resultados a devolver por la función. Su valor por defecto es 1.
- "include_tags": parámetro opcional, determina si la función debe devolver información adicional sobre las palabras reconocidas. Su valor por defecto es False.
- "contexto": parámetro opcional, contiene el contexto del usuario. Su valor por defecto es None.

Siguiendo con el ejemplo descrito en la sección anterior, el resultado devuelto por la función es un objeto JSON con la siguiente estructura:

```
{
    "entities": {"LightKeyword": "luz", "LightOnOffLabel": "sala",
                 "OnOffAction": "apagar"},
    "intent_type" : "set_light_state",
    "confidence" : "1.0",
    "missing_required_tag": "",
    "target": None
}
```

La información acerca de los entities reconocidos por Adapt puede ser completada con metadatos adicionales haciendo uso de la función “get_entity_metadata()” ofrecida por el módulo.

A pesar de que las funcionalidades ofrecidas por Adapt son excelentes, durante el desarrollo del proyecto se detectaron un par de carencias que fueron subsanadas modificando la librería. A continuación se describen los problemas detectados y de qué manera se solventaron.

5.6.1. Confidence = 0

En Adapt, a la hora de definir un intent, se debe informar al framework acerca de los entities que deben aparecer en el utterance para que el intent sea validado. Sin embargo, el sistema es tan rígido que en caso de que uno solo de los entities no sea detectado, el confidence final del intent es igual a 0. Un escenario más adecuado sería aquel en el que el sistema buscara el intent que mejor se ajustara al utterance, a pesar de no satisfacer todos los requisitos, y en caso de necesitar información adicional, interactuara con el usuario para recoger la información no disponible. Gracias a los cambios efectuados en la librería, la función “determine_intent()” devuelve ahora una pareja clave/valor adicional llamada “missing_requiered_tag” que puede ser usada para que el cliente Mayordomo lance una pregunta al usuario.

Suponiendo que un cliente publique un mensaje en el topic “mayordomo/utterance/message” con la orden “apaga la luz, por favor”, la función “determine_intent” devolverá ahora el objeto JSON:

```
{
    "entities": {"LightKeyword": "luz", "OnOffAction": "apagar"},
    "intent_type" : "set_light_state",
    "confidence" : "0.6",
    "missing_required_tag": ["LightOnOffLabel"],
    "target": None
}
```

A la hora de determinar cuál debe ser la pregunta a lanzar al usuario ante la ausencia de un entity, los metadatos del mismo juegan un papel fundamental.

5.6.2. Metadatos

A raíz de los visto en la sección 5.6.1 y como se verá más adelante cuando se analicen las capacidades multilenguaje de Mayordomo, se detectó que no era suficiente con registrar los entities como una pareja "categoria:valor" (por ejemplo: "HomeRoom: sala"). Era necesario incluir más información sobre los entities para que el sistema fuera capaz de lidiar con situaciones más complejas que las que habían previsto los autores de Adapt. Así pues, el sistema ahora es capaz de guardar más información sobre un entity y Mayordomo es capaz de consultar esa información gracias a la función "get_entity_metadata". De:

```
{
    "OnOffAction": "apagar"
}
```

se pasa ahora a:

```
{
    "OnOffAction": {"entity_original_value": "switch_off",
                    "entity_value": "apagar",
                    "missing_phrase": "¿Qué quieres hacer con el elemento?",
                    "language": "es-ES",
                    "entity_type": "OnOffAction"}
}
```

5.7. Módulo Sesión/Contexto

El módulo de Sesión/Contexto (session_handler) se encarga de crear, actualizar y eliminar las sesiones de los usuarios del sistema, así como de mantener actualizado el contexto asociado a cada usuario. La lista de usuarios que han utilizado el programa viene dada por los nombres de usuario recibidos a través de la clave "user_name" de los mensajes publicados en el topic "mayordomo/utterace/message". Ver la sección 5.5 para más información.

Por defecto, cada sesión tiene una duración de 5 minutos (aunque el valor es editable) y las sesiones son eliminadas si, tras ese tiempo, el usuario no ha envíado ningún utterance nuevo al programa.

5.8. Módulo Plugins

El módulo Plugins (plugins_handler) es uno de los más importantes del sistema, ya que es el encargado de leer y cargar los plugins instalados. Será responsabilidad de cada plugin definir tanto los entities como los intents en los que está interesado y, para cada intent definido, deberá implementar el método que será ejecutado cuando el utterance del usuario sea asociado con dicho intent. Todos los plugins deben estar escritos en inglés y ser instalados en una subcarpeta propia dentro de la carpeta plugins, aunque sobre las particularidades de cómo desarrollar un plugin se hablará en mayor profundidad en el capítulo 8.2.

En su instalación básica, Mayordomo incluye tres plugins listos para ser usados, ChangeLanguagePlugin, GreetingsPlugin y openHAB2Plugin.

5.8.1. ChangeLanguagePlugin

El primero de ellos, ChangeLanguagePlugin, sirve para cambiar el idioma del sistema a través de un comando de voz. Este plugin define dos entities y un único intent.

Los entities son:

- "SpeakKeyword": cuyo único valor es "speak". El módulo traductor se encargará de traducir la palabra al idioma correspondiente.

- "Language": sus valores son "English", "Spanish", "Basque", "Italian", "German"... Como en el caso anterior, el módulo traductor se encargará de traducir las palabras al idioma configurado en el programa.

Y el intent es:

- "change_language", que hace uso de los entities arriba definidos.

5.8.2. GreetingsPlugin

Este plugin se creó inicialmente para validar el funcionamiento del programa y su único objetivo es saludar a una persona a través de un comando de voz. Al igual que en el plugin anterior, sólo se definen dos entities y un intent.

Entities:

- "Person": nombres de personas a las que se podrá saludar, tales como Ana, Ane, Iker, Aitor, Kepa, Maite, Mike, John, Emily... Estos nombres no se traducen.
- "GreetingsAction": los valores posibles son "hi", "hello" y "greet", palabras que el usuario deberá pronunciar cuando desee ejecutar la acción.

Intent:

- "greet_person", que define los entities "Person" y "GreetingsAction" como requeridos.

5.8.3. openHAB2Plugin

Por último, el plugin openHAB2Plugin es más complejo que los dos anteriores y es el encargado de controlar los elementos domóticos del hogar. Antes de pasar a detallar los entities e intents definidos, es conveniente realizar un breve repaso por las características principales del programa.

openHAB [17] es un programa open source escrito en Java que nace para combatir los problemas de interoperabilidad existentes en el mundo de la domótica y del IoT. Hoy en

día cada vez más elementos de la vida cotidiana son capaces de reportar información o ser controlados remotamente, pero generalmente cada fabricante implementa sus propios protocolos y desarrolla sus propias aplicaciones para interactuar con los dispositivos. Esto provoca que el usuario no sea capaz de gestionar todos sus dispositivos desde una única aplicación y de ahí surgen los problemas de interoperabilidad antes mencionados.

La forma en la que openHAB soluciona este problema es modelando los dispositivos según su funcionalidad en vez de por sus características físicas. Un sensor que sea capaz de medir la temperatura ambiente y la humedad relativa de una habitación no será representando en la aplicación como un único sensor, aunque físicamente lo sea, sino como dos elemento diferenciados: el valor de temperatura reportado por un lado y el valor de humedad por otro. Cada uno de estos valores recibe el nombre de "Item" dentro del sistema [18].

Los items en openHAB son por tanto los elementos básicos del sistema y describen, de forma individual, cada una de las funcionalidades de los sensores y actuadores. A la hora de definir un item se debe especificar su tipo (Switch, String, Number, Contact, Color, Dimmer, RollerShutter, Player, Datetime y Group), su nombre (que es el cadena de caracteres que actuará como su identificador) y un label (que contiene la descripción de lo que representa). Adicionalemente, los items pueden ser etiquetados con un "Tag" para dar un mayor carácter semántico a los mismos. La lista de tags soportada por openHAB es por ahora limitada, y se resume en: Lighting, Switchable, CurrentTemperature, CurrentHumidity y Thermostat. Estos tags permiten a openHAB ser compatible con otros asistentes de voz como Siri (de Apple) y Alexa (de Amazon), además de ser el sistema empleado también por Mayordomo.

El último punto a destacar de openHAB, y que tiene relevancia con el proyecto descrito en el presente documento, es que posee una API REST. Esta API permite que otros programas sean capaces de leer los items definidos en la aplicación a través de llamadas HTTP.

Esta breve introdución a openHAB permite comprender mejor cómo actúa este tercer plugin. El openHAB2Plugin define 16 entities, muchos de ellos con valores predefinidos (increase, decrease, switch_on, switch_off, temperature, humidity, light, color, red, blue,...), pero varios de ellos son dinámicos, ya que sus valores son introducidos en Adapt a través de la REST API de openHAB. Entre los entities dinámicos, se pueden destacar:

- "LightOnOffLabel": contiene el label de un item que modela una bombilla que se puede encender y apagar.

- “LightDimLabel”: contiene el label de un item que modela una bombilla cuya intensidad puede ser regulada (del 0 al 100 %).
- “LightColorLabel”: contiene el label de un item que modela una bombilla cuyo color puede ser cambiado.
- “TemperatureLabel”: contiene el label de un item que representa la temperatura de una estancia de la casa.
- “HumidityLabel”: contiene el label de un item que representa la humedad de una estancia de la casa.
- “BrightnessLabel”: contiene el label de un item que representa la luminosidad de una estacia de la casa.
- “SwitchableOnOffLabel”: contiene el label de un elemento que se puede encender y apagar (horno, microondas, ordenador, televisión, radiador,...).

Gracias a esta particularidad, Mayordomo se ajusta a la configuración personal de cada usuario, en vez de definir los nombres de los elementos de antemano.

Para acabar, la lista de intents registrados por el plugin es:

- “set_light_state”: para encender/apagar una luz.
- “set_light_level”: para fijar el brillo de una luz.
- “set_light_color”: para cambiar el color de una luz.
- “get_temperature”: para consultar la temperatura de una habitación.
- “get_humidity”: para consultar la humedad de una habitación.
- “get_brightness”: para consultar la luminosidad de una habitación.
- “set_device_state”: para encender/apagar un dispositivo.

5.9. Módulo Traductor

Este módulo llamado Traductor (translation_handler) ofrece al servidor la posibilidad de traducir palabras o frases de un idioma a otro, lo cual es fundamental a la hora de dotar al sistema de la capacidad para funcionar en varios idiomas.

El módulo ofrece diferentes mecanismos para realizar las traducciones, pero estos serán tratados en el capítulo 7.

5.10. Módulo NLP

El módulo NLP (nlp_providers_handler) se encarga de la tokenización y lematización de la frase pronunciada por el usuario (el utterance), tal y como ha sido se ha explicado en la parte final de la sección 5.5. Las dos herramientas usadas para tal fin, Freeling e IXA-PIPES, serán analizada con detalle en el capítulo 7.

5.11. Módulo TTS

El módulo denominado TTS (tts_handler) ofrece a los clientes Mayordomo un servicio de conversión de texto a voz para que puedan dar notificaciones habladas a los usuarios. Por defecto, Mayordomo incorpora dos servicios para convertir texto a voz: Ivona y AhoTTS.

Ivona [19] es un servicio web de conversión de texto a voz de gran calidad ofrecido por Amazon, tras la compra de la empresa con el mismo nombre. Ofrece más de 45 voces en 24 idiomas diferentes, entre los que se pueden destacar el inglés británico, el inglés americano, el italiano, el francés, el alemán, el español de España o el español latino. Desgraciadamente, el euskera no es una de las voces ofrecidas.

Para hacer uso del servicio es necesario crearse una cuenta de desarrollar en su página web, con el fin de obtener las credenciales necesarias. Estas credenciales son un "access_key" y un "secret_key" descargables desde el perfil del propio usuario. El servicio ofrece la posibilidad de realizar 50.000 llamadas mensuales a la API de forma gratuita, pero para evitar que un usuario de la plataforma Mayordomo agote esas llamadas, el módulo TTS cachea localmente los ficheros de audio descargados.

Para que los clientes Mayordomo sean capaces de ofrecer notificaciones en euskera, el

Figura 5.2: Logo de *Ivona*.

servicio AhoTTS [20] ha sido añadido a la plataforma. AhoTTS es un programa desarrollado por la Universidad del País Vasco (EHU/UPV) que permite convertir texto a voz en dos idiomas diferentes: euskera y castellano. A pesar de que la calidad de las voces es inferior a la conseguida por Ivona, la gran ventaja de este programa es que funciona en local, sin ninguna conexión a internet. Los audios generados por AhoTTS también son almacenados en el disco duro, tal y como sucedía con Ivona, para responder lo antes posible a los clientes Mayordomo cuando estos soliciten la conversión de un frase previamente procesada.

Como se puede ver en el diagrama de la sección 6.1, el módulo TTS también se apoya en el módulo MQTT para ofrecer sus servicios mediante dicho protocolo. Los clientes interesados en convertir un texto en audio, deberán publicar un mensaje en el topic "mayordomo/tts/get_audio". Dicho mensaje tendrá que incluir un objecto JSON con los siguientes pares atributo/valor en su payload:

- "topic": nombre del topic al que el módulo TTS deberá enviar el fichero de audio.
- "audio_data": objeto JSON con información sobre el texto a convertir.

El atributo "audio_data" tendrá que ser a su vez otro objecto JSON con los pares:

- "text": frase que se desea convertir en audio.
- "laguange": idioma (bajo la codificación BCP-47) en el que se desea el audio. Si no se especifica, por defecto es "en-US".
- "gender": tipo de voz a usar en el audio. Dos opciones: "male" o "female". El valor por defecto es "female".

El siguiente JSON es un ejemplo válido de lo que tendría que publicar un cliente Mayordomo en el topic "mayordomo/tts/get_audio":

```
{
    "topic": "mayordomo/tts/temp/asdljfljasdfljasdf",
    "audio_info" : {
                        "text": "Luz de la sala encendida",
                        "language": "es-ES",
                        "gender": "female"
                    }
}
```

6. CAPÍTULO

Mayordomo cliente

Los clientes Mayordomo son los elementos de la plataforma encargados de comunicarse con el usuario final, ya sea recogiendo sus peticiones a través de comandos de voz, como devolviéndoles el resultado de los comandos a través de notificaciones habladas.

Como se puede observar en la figura 4.1, cualquier dispositivo con conectividad, ya sea un teléfono, un reloj inteligente o una tablet, puede convertirse en un cliente Mayordomo. Los únicos requisitos con los que debe contar son:

- Conectividad IP para poder hacer uso del protocolo MQTT.
- Micrófono (o teclado) para que el usuario puede introducir el comando deseado.
- Seguir el convenio establecido en el capítulo 5 para la publicación de mensajes y suscripción de topics.

Con el fin de dotar a la plataforma de una solución completa, y debido a la nula experiencia programando aplicación en Java para Android, en este proyecto se ha desarrollado un cliente Mayordomo de escritorio [21]. La idea es que el cliente pueda ser instalado en cualquier ordenador capaz de ejecutar la versión 2.7 de Python, aunque el hardware ideal para este programa vuelve a ser una Raspberry Pi con un micrófono y unos altavoces con conexión USB. Su bajo consumo de energía, su precio y el poco espacio que ocupa vuelven a hacer de la Raspberry Pi el candidato perfecto para esta tarea.

Además de los requisitos previamente expuestos, a la hora de desarrollar el cliente se han tenido en cuenta dos condicionantes extra:

- El usuario debe ser capaz de hablarle al sistema sin tener que accionar ningún botón y a 2-3 metros de distancia.

- La aplicación tiene que ser capaz de transcribir el comando en diferentes idiomas.

En los siguientes capítulos se analiza la arquitectura del programa y se realiza una pequeña descripción de la funcionalidad de cada módulo.

6.1. Arquitectura

Gracias a que la parte de procesado del utterance la realiza el servidor, el diseño del cliente es significativamente más sencillo, con menos módulos y un menor número de líneas de código. Los clientes tienen que destacar en dos funciones:

- Activarse de una forma sencilla y natural.

- Convertir la voz del usuario a texto lo más rápido posible.

Los módulos desarrollados y la relación entre ellos pueden verse en la figura 6.1:

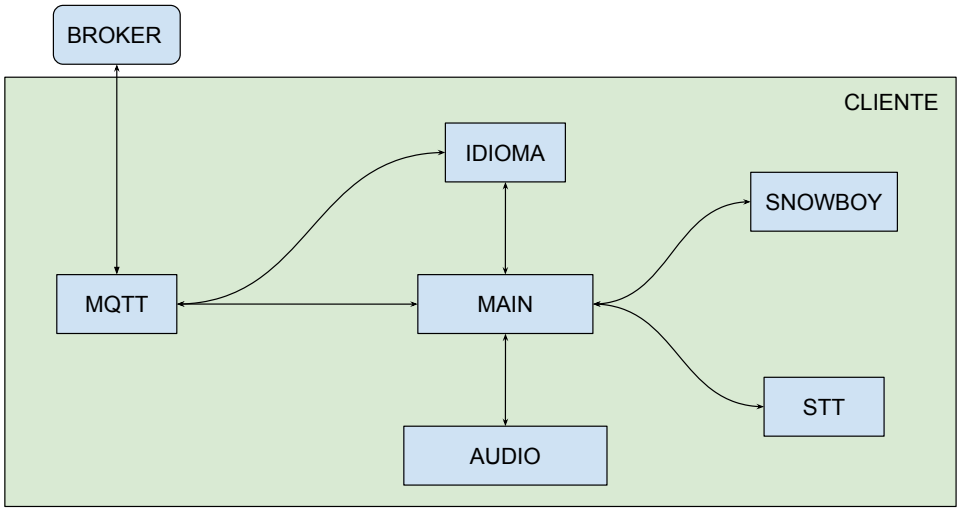


Figura 6.1: Arquitectura del cliente.

6.2. Módulo Main

El módulo Main se encarga de leer el fichero de configuración “configuration.yml” y de activar el resto de módulos del programa. Es el punto de entrada del programa, de ahí que su funcionalidad no sea compleja.

6.3. Módulo MQTT

Como ya sucedía en el módulo MQTT del servidor, este módulo sirve de interfaz para que otros módulos publiquen mensajes y se suscriban a topics MQTT a través de una única conexión. El módulo de por sí no se suscribe ni publica mensajes, su único cometido es ser importado por el resto de elemento del sistema con del comando “import mqtt_handler”.

6.4. Módulo Idioma

En el sección 6.4 se expone que el servidor informa a través del protocolo MQTT de cuál es el idioma del sistema publicando un mensaje en el topic “mayordomo/configuration/language”. La función de este módulo es suscribirse a ese topic para ser notificado cuando haya un cambio de idioma en la plataforma, de forma que todo el sistema esté sincronizado.

Nuevamente, el idioma es gestionado en el cliente haciendo uso de la clase Language, cuya representación interna está basada en la codificación BCP-47 [15].

6.5. Módulo Audio

El módulo Audio (audio_handler) gestiona todas las funciones relacionadas con la reproducción de notificaciones por parte del cliente.

Por un lado, se encarga de solicitar al servidor el audio cuando se quiere convertir texto a voz. El protocolo a seguir ha sido descrito en la sección 5.11. El payload del mensaje devuelto por el servidor incluye un objeto JSON en el que uno de los pares clave/valor contiene el audio en formato .wav codificado como un archivo binario en base64.

Por otro, se ocupa de reproducir el audio a través del sistema de sonido del dispositivo. Para esta labor, se optado por usar la librería “pyaudio” de Python, ya que facilita trabajar con este tipo de ficheros.

6.6. Módulo Snowboy

Una de las características más interesantes de esta implementación de un cliente Mayordomo es la posibilidad de activar el sistema sin pulsar ningún botón, sólo mediante la voz. Para que el sistema sea fácil de manejar por parte del usuario, el cliente debe estar preparado para recibir un comando en cualquier momento, a cualquier hora. Obligar al usuario a desplazarse hasta el dispositivo para activarlo no sería práctico. Sin embargo, convertir todo el audio registrado por el micrófono en texto tampoco es una opción viable por dos motivos. La primera es que emplearía demasiados recursos del sistema y el programa está pensado para ser ejecutado en dispositivos con poca potencia de cómputo. El segundo motivo es que los mejores conversores de voz a texto del mercado son ofrecidos como servicios online por empresas como Google [22], Microsoft [23] o IBM [24]. Los servicios ofrecidos por estas empresas suelen tener una limitación de peticiones mensuales, las cuales se agotarían rápidamente si procesáramos en tiempo real todo el audio captado por el micrófono.

Para hacer frente a este problema se ha incorporado al proyecto un servicio de detección de palabras clave (hotword detection) llamado Snowboy [25]. De esta manera, el cliente está a la espera de que el usuario pronuncie la palabra de activación y, a partir de ese momento, comienza el proceso de conversión de voz a texto.

Figura 6.2: Logo de *Snowboy*.

A continuación se describen las principales características del programa:

- Es gratuito para uso no-comercial.
- A pesar de estar escrito en C++, ofrece un wrapper para Python.
- La detección de la palabra clave se hace en local, no es necesario tener una conexión activa a internet.

Dentro de Snowboy existen dos tipos de modelos de palabras clave, los universales y los personales. Los primeros han sido entrenados en una red neuronal con muchas muestras de audio y por tanto pueden ser usados por cualquier persona. La extensión de estos ficheros es ".umdl". Los segundos, como su nombre indica, son personales, y para generarlos es necesario enviar tres muestras de audio a la web del programa. A partir de estas muestras, el servidor web de Snowboy genera un fichero con extensión ".pmdl" que puede ser descargado y usado por el programa. El cliente Mayordomo incorpora el fichero universal "snowboy.umdl", en el que la palabra de activación es "snowboy", y el modelo personal "mayordomo.pmdl" generado a partir de mi voz. Se recomienda que cada usuario del sistema genere su propio modelo personal para aumentar la fiabilidad del sistema.

6.7. Módulo STT

Por último, el módulo STT es el encargado del proceso de convertir la voz del usuario en una cadena de caracteres para su posterior análisis.

Las distintas opciones existentes en el mercado pueden dividirse en dos grupos: los módulos STT que funcionan de forma local y los que se ofrecen como un servicio en la nube. Los primeros tienen tres aspectos positivos a destacar: pueden funcionar sin conexión a internet, respetan la privacidad del usuario y en la mayoría de los casos son open source. Sin embargo, también tienen una serie de puntos negativos: su instalación es más compleja, generalmente sólo funcionan en inglés y el resultado obtenido no se puede comparar al ofrecido por los servicios cloud.

Debido a estas razones, y tras analizar las diferentes opciones disponibles, se ha decidido hacer uso de la plataforma Cloud Speech de Google.

La principal razón que marca la diferencia entre las plataformas de Google, Microsoft e IBM es que la primera es capaz de convertir voz a texto en más de 80 idiomas, incluyendo

Figura 6.3: Logo de *Google Cloud Speech*.

el euskera. Así, el mismo servicio es capaz de satisfacer las necesidades multilenguaje del cliente, sin tener que añadir módulos adicionales. Cabe recordar que para la conversión de texto a voz se tuvieron que añadir dos servicios diferentes: Ivona y AhoTTS.

El mayor defecto de la plataforma de Google se encuentra en el número de minutos en los que se puede hacer uso del servicio de manera gratuita. El tiempo está limitado a 60 minutos al mes, lo que da una media de 2 minutos diarios. Considerando que las órdenes que se le van a dar al sistema son muy breves (2-3 segundos), el límite impuesto por Google puede ser suficiente para un uso medio del programa (10-20 comandos al día).

7. CAPÍTULO

Herramientas NLP

En este capítulo se describirán las herramientas empleadas en el proyecto que tienen relación con el procesamiento del lenguaje natural y con las capacidades multilenguaje del programa. Al final del capítulo, se describirá el algoritmo usado para hacer que Mayordomo sea compatible con varios idiomas.

7.1. Wordnet

Wordnet [26] es una base de datos léxica en inglés desarrollada por la universidad de Princeton. La base de datos contiene más de 150.00 palabras [27], entre las que se pueden encontrar sustantivos, verbos, adjetivos y adverbios, y la forma en la que las palabras están relacionadas unas con otras es mediante "synsets". Los synsets son grupos de palabras que tienen la misma categoría léxica y que pueden ser considerados sinónimos.

De forma complementaria a Wordnet, otras base de datos léxicas han sido desarrolladas para otros idiomas en los últimos años. Sin embargo, muchas de éstas han seguido el esquema del Wordnet original para su desarrollo y por tanto, han podido surgir iniciativas como Open Multilingual Wordnet [28]. El proyecto Open Multilingual Wordnet se encarga de recopilar y normalizar diferentes bases de datos para que el uso de Wordnet en varios idiomas sea más accesible. Además de la base de datos original en inglés, existen versiones de Wordnet en castellano y en euskera [29] [30].

Para que Mayordomo tenga acceso a las bases de datos disponibles en Open Multilingual

Wordet, se ha añadido la libería NLTK [31] al sistema. El paquete NLTK ofrece una API intuitiva, fácil de usar y que no requiere de conexión a internet para buscar, relacionar y traducir palabras entre idiomas desde Python.

Gracias a Wordnet y a NLTK, Mayordomo es capaz de traducir los entities definidos por los plugins -ya sean sustantivos, verbos, adjetivos o adverbios- del idioma en el que fueron definidos al idioma del sistema.

7.2. Google Translate

Para que el framework fuera capaz de responder al usuario en varios idiomas, ya sea para dar una respuesta a una pregunta, informar sobre la ejecución de un comando o lanzar una pregunta para obtener más información, ha sido necesario introducir un programa que fuera capaz de traducir frases completas de un idioma o otra. Wordnet no es una herramienta apropiada para resolver este problema porque solo es capaz de traducir palabras.

Tras analizar las distintas opciones disponibles, finalmente se ha añadido la librería pygoogletrans [32] al sistema. Esta librería, escrita en Python y de código abierto, hace uso de la API de Google Translate para ofrecer traducciones entre más de 100 idiomas. Con el fin de reducir el número de peticiones a la API de Google y para disminuir el tiempo de ejecución, Mayordomo cachea las traducciones en un fichero de texto.

7.3. i18n

A pesar de disponer de Wordnet y Google Translate para automatizar el proceso de traducción, tanto el sistema como el usuario pueden encontrarse ante situaciones en las que ninguno de los dos servicios ofrezca una traducción satisfactoria. Para resolver este problema, Mayordomo incorpora la posibilidad de que el usuario -o el desarrollador del plugin- traduzca manualmente las palabras o frases que considere erróneas.

Las traducciones manuales se realizan mediante ficheros con extensión ".po", siguiendo el estándar de localización e internacionalización (i18n) "gettext" [33]. Los ficheros se guardan en una subcarpeta llamada "locale" dentro de la carpeta de cada plugin, y a su vez, la traducción de cada idioma se almacena en su propia carpeta. Así, la traducción en castellano de plugin GreetingsPlugin se encuentra en: "`GreetingsPlugin\locale\`

es_ES\LC_MESSAGES\GreetingsPlugin.po". Estos ficheros son generados automáticamente para cada plugin cuando se cambia el idioma del sistema y pueden ser editados con cualquier editor de textos. A continuación se muestra un extracto del fichero OpenHAB2Plugin.po:

```
#: OpenHAB2Plugin.py:43
msgid "Which light?"
msgstr ""

#: OpenHAB2Plugin.py:46
msgid "switch_on"
msgstr ""

#: OpenHAB2Plugin.py:46
msgid "switch_off"
msgstr ""

#: OpenHAB2Plugin.py:47
msgid "What do you want to do with the light?"
msgstr ""
```

El fichero ".po" muestra el archivo del que se han extraído las palabras, las líneas en las que aparecen y el identificador en el idioma original. Para realizar una traducción, el usuario sólo tiene que sustituir el valor del parámetro "msgstr". No es necesario traducir el fichero al completo, Mayordomo permite traducir sólo los identificadores que se consideren oportunos.

7.4. Servicio de traducción

Esta sección describe el proceso que sigue Mayordomo para introducir los entities en Adapt en el idioma configurado. Cada palabra o frase que se desea traducir se debe definir llamando a una de la siguiente funciones:

- self._(«entity en inglés»), si es una palabra (lema).
- self._sentence(«frase en inglés»), si es una frase.

La primera función dispone de tres parámetros opcionales. “pos”, en el que se le puede “ayudar” a Wordnet especificando si el entity es un noun, verb, adj o adv. “original_lang”, donde se puede especificar en qué idioma está el entity (por defecto en inglés). “translation_lang”, donde se puede definir a qué idioma queremos traducir la palabra (por defecto al idioma en el que esté configurado el sistema en ese momento).

La segunda función también presenta los parámetros “original_lang” y “translation_lang” de la función anterior.

El orden de ejecución que sigue la función “self._()” es:

1. Se comprueba si existe el fichero “.po” y, si el entity ha sido traducido, se coge el valor introducido por el usuario para añadirlo a Adapt.
2. Si no hay traducción manual, se buscan todos los synsets de esa palabra con Wordnet y se cogen todos los lemmas correspondientes a los synsets en el idioma a traducir.
3. Si aún así el sistema no encuentra nada, se usa Google Translate para obtener la traducción del entity e introducirlo en Adapt.

Para “self._sentence()” se tiene:

1. Se comprueba si existe el fichero “.po” y, si la frase ha sido traducida, se coge el valor introducido por el usuario para añadirlo a Adapt.
2. Si no hay traducción manual, se usa Google Translate directamente para conseguir la traducción de la frase.

7.5. Freeling

En la sección 5.5 del capítulo dedicado al servidor se explica que, cuando se recibe un mensaje en el topic “mayordomo/utterance/message”, el sistema obtiene los lemas de

cada una de las palabras presenten en la orden del usuario. Para ello, el programa dispone de dos herramientas: Freeling e IXA-PIPES.

Freeling [34] es una librería que ofrece multitud de funcionalidades para el análisis lingüístico (tokenización, análisis morfológico, detección de entidades, etiquetado gramatical o PoS-tagging, desambiguación,...) en una gran cantidad de idiomas, entre los que se pueden destacar: inglés, castellano, francés, alemán, portugués, italiano,... La librería ha sido desarrollada por la Universitat Politècnica de Catalunya y está escrita en C++, aunque se puede usar en Python a través de un wrapper.

Entre todas las opciones ofrecidas por Freeling, Mayordomo sólo hace uso de la tokenización y del etiquetado gramatical para la lematización de la oración. La función de la lematización es la normalización de las palabaras con el fin de facilitar la búsqueda de los entities mediante Adapt. En algunos idiomas, entre los que se puede destacar el euskera, las declinaciones (absolutivo, ergativo, dativo, adlativo, ablativo,...) se añaden como sufijos en la palabra, por lo que es muy importante obtener el lema (raíz) de la palabra para poder determinar correctamente el intent de la oración.

7.6. IXA-PIPES

IXA-PIPES [2] es un conjunto de herramientas open source especializadas en el procesamiento del lenguaje natural en varios idiomas. Los módulos están desarrollados por el grupo IXA NLP de la Universidad del País Vasco (EHU) y están escritos en Java. Estos son los módulos principales del sistema:

- ixa-pipe-tok: Tokenizador de oraciones y textos para varios idiomas.
- ixa-pipe-pos: Etiquetador gramatical estadístico y lematizador en euskera, holandés, inglés, francés, gallego, alemán, inglés y castellano.
- ixa-pipe-nerc: Identificador de entidades en euskera, castellano, inglés, alemán, holandés e italiano.
- ixa-pipe-chunk: Chunker probabilístico en euskera y castellano.
- ixa-pipe-parse: Parser probabilístico en castellano e inglés.

La arquitectura IXA-PIPES es muy modular ya que cada herramienta se encarga de una parte del proceso y por tanto se puede elegir qué módulos instalar y cuáles no. El servidor

mayordomo hace uso de los módulos ixa-pipe-tok e ixa-pipe-pos para obtener los lemas del utterance enviado por el cliente.

Al estar escritos en Java, los módulos no pueden importarse al proyecto como si fueran librerías de Python. Para hacer uso de ixa-pipe-tok, Mayordomo llama al fichero ".jar" mediante una llamada de la consola de comandos. En el caso de ixa-pipe-pos, en cambio, se ejecuta el etiquetador gramatical como un servidor TCP y Mayordomo realiza las consultas como un cliente. La información que se envía al módulo ixa-pipe-pos es el documento NAF devuelto por ixa-pipe-tok.

7.7. Multilenguaje

Por último, en esta sección se muestra, a modo de resumen, una tabla con los idiomas con los que el sistema es capaz de trabajar por defecto.

Módulo	**Idiomas**
Speech to Text	>80
Lematización	as, ca, cy, de, en, es, eu, fr, gl, it, nl, pt, ru, sl
Traducción	>100
Text to Speech	cy, da, de, en, es, eu, fr, is, it, ja, nl, no, pl, pt, ro, ru, sv, tr

Tabla 7.1: Idiomas soportados por Mayordomo.

8. CAPÍTULO

Whitepaper

Este capítulo pretende servir de guía para cualquiera que esté interesado en ampliar las capacidades de la plataforma. En la sección 8.1 se describen los pasos a seguir para añadir más idiomas al programa y en la 8.2, se detalla el proceso de creación de un plugin.

8.1. ¿Cómo añadir mas idiomas?

A pesar de que, como se muestra en la tabla 7.1, Mayordomo es capaz de trabajar de principio a fin en más de 8 idiomas (desde el proceso de STT a TTS), estos no son todos los idiomas existentes en el mundo. Por tanto, dependiendo de las necesidades del usuario, puede ser interesante añadir más idiomas a la plataforma. Debido a las diferentes partes involucradas en el proceso, a continuación se detallan los pasos para añadir un nuevo idioma a cada sección.

- Speech to Text

  La parte de STT está únicamente ligada a los clientes Mayordomo, por lo tanto es responsabilidad de cada cliente incorporar el nuevo idioma a la conversión de voz a texto. Para el cliente implementado en python, se deberá crear un nuevo módulo con un método público llamado “start()” que se encargue de registrar el audio del micrófono, procesarlo y devolver la frase de usuario en forma de string. La implementación concreta, tanto si el sistema procesa el audio en local como en la nube, debe estar encapsulada en el mismo módulo.

- Traductor

  El módulo traductor, como se vió en el capítulo 7, está dividido en tres partes: traducción manual, Wordnet y Google Translate. La parte de traducción manual es compatible con cualquier idioma, por lo que no es necesario editar esa parte. La de WordNet, al ser un conjunto de bases de datos gestionadas de forma externa, tampoco puede ser ampliada, el sistema está limitado a los idiomas que Open Multilingual WordNet es capaz de procesar. Finalmente nos queda Google Translate, que es el único elemento modificable de este módulo. Para añadir un nuevo traductor se deberá crear un módulo de Python en la carpeta "translation_provider", con un método público llamado "translate()". Los parámetros admitidos por este método serán: sentence (la frase o palabra a traducir), src (el idioma del texto original) y dest (el idioma al que se deberá traducir el sentence).

- Lematización

  Añadir un nuevo idioma a la plataforma también exige que ésta sea capaz de lematizar los utterance recibidos desde los clientes. Para ello, se deberá desarrollar una API en Python que será la que se comunique con la nueva herramienta de procesamiento del lenguaje natural. Esta API deberá estar representada por un fichero con extensión ".py" dentro de una subcarpeta del directorio "nlp_providers". El nombre de la subcarpeta y del fichero ".py" tendrán que ser iguales para que el servidor pueda importarlo automáticamente. Los métodos públicos a implementar de forma obligatoria por el nuevo módulo deberán ser:

  - "get_quality()": método que devuelve (con un número del 0 al 10) la calidad ofrecida por el servicio. Este valor es utilizado por la plataforma para elegir la herramienta NLP a emplear cuando más de una es compatible con el mismo idioma.
  - "get_supported_languages()": método que devuelve una lista con los idiomas soportados.
  - "get_sentence_lemmas()": método que, dada una frase como parámetro, devuelve la misma frase lematizada como una cadena de caracteres.

- Text to Speech

  Dado que el servidor ofrece un servicio de texto a voz para que los clientes que lo necesiten hagan uso de él, será necesario actualizarlo cuando se añada un nuevo idioma. El mecanismo es muy similar al expuesto en el punto anterior, ya que

será necesario implementar una API que haga de pasarela entre Mayordomo y el servicio empleado. La API deberá ser empaquetada en una carpeta cuyo nombre tendrá que coincidir con el del fichero ".py" y está carpeta será copiada al directorio "tts_providers". A su vez, la API deberá incluir los siguiente método públicos:

- "get_quality()": método que devuelve (con un número del 0 al 10) la calidad ofrecida por el servicio. Este valor es utilizado por la plataforma para elegir el proveedor a emplear cuando más de uno es compatible con el mismo idioma.
- "get_supported_languages()": método que devuelve un diccionario de Python con los idiomas soportados. Las claves del diccionario serán los identificadores de los idiomas en formato BCP-47 y los valores serán una lista con el género de la voz (female o male).
- "set_cache_directory()": método que empleará la plataforma para indicarle al módulo en qué carpeta debe guardar los ficheros de audio.
- "create_audio_file()": método que, dado un texto como parámetro, devuelve la ruta del fichero que contiene el audio generado. El método debe permitir también los parámetros "language" y "gender".

8.2. ¿Cómo desarrollar un plugin?

Esta sección profundiza en el desarrollo de un plugin para la plataforma Mayordomo. El framework se encarga de cargar el plugin y de ejecutarlo, de modo que el desarrollador del mismo sólo tiene que ocuparse de registrar los entities e intents en los que está interesado.

8.2.1. Estructura

Todos los plugins deben ser empaquetados en una carpeta, cuyo título debe ser el nombre del plugin. Dentro de la carpeta principal se pueden incluir todas las carpetas, librerías y ficheros extra que se deseen, pero debe existir obligatoriamente un archivo llamado "nombre_del_plugin.py", en cuyo interior se deberá definir una clase con el mismo nombre. Este archivo será el fichero principal del plugin, el que ejecutará la plataforma, y por tanto debe satisfacer las condiciones que se presenten en las siguientes secciones.

Si el plugin va a permitir al usuario editar una serie de parámetros de configuración, Mayordomo ofrece la opción de definir un archivo de configuración llamado "configura-

tion.yml" en la carpeta raíz del plugin. El archivo de configuración será leído y procesado automáticamente por el programa y los valores de los parámetros están disponibles en un diccionario con el nombre "self.config".

8.2.2. Librerías

Además de las librerías que cada desarrollador necesite importar para el correcto funcionamiento del plugin, el fichero "nombre_del_plugin.py" deberá importar las siguientes dos:

- La clase "Plugin" y las funciones "register_entities" y "register_intent" del módulo "plugins.plugin_base", mediante el comando "from plugins.plugin_base import Plugin, register_intents, register_entities".
- La clase "Notification" del módulo "lib.notification", mediante el comando "from lib.notification import Notification".

8.2.3. Clase base

La clase principal del plugin, cuyo nombre tendrá que ser igual al nombre del plugin y al fichero .py, deberá heredar de la clase "Plugin" importada en la sección anterior. Las funcionalidades ofrecidas por esta clase padre son:

- Crea automáticamente los ficheros .po con los entities y frases definidas en las funciones self._() y self._sentence(). Estos ficheros son los empleados para la gestión de traducciones manuales.
- Lee e importa los parámetros de configuración del fichero "configuration.yml".
- Ejecuta el código implementado por el desarrollador en la función setup().
- Importa los entities definidos por el desarrollador en la función register_entities().
- Importa los intents definidos por el desarrollador en la función register_intents().

8.2.4. Entities e intents

Dado que el objetivo de un plugin es ayudar al usuario a realizar unas determinadas acciones a través de la definición de intents (intents a los que se asocian una serie de entities), el plugin tendrá que registrar en la plataforma los entities e intents en los que está interesado. Esto se hará haciendo uso de dos funciones:

- "register_entities()"

  El resultado devuelto por esta función deberá ser un diccionario de Python. Cada una de las claves del diccionario será una categoría de un grupo de entities y su valor, otro diccionario con estos pares clave/valor:

  - "values": lista con cada uno de los entities.
  - "question": pregunta que se le formulará al usuario cuando el sistema detecte que no tiene información sobre este entity para completar el intent.
  - "type": tipo de entity, sólo será necesario especificarlo si se desea registar un entity del tipo "regex".

  La imagen 8.1 muestra la implementación de la función register_entities() para el plugin ChangeLanguagePlugin.

```
@register_entities
def register_entities(self):
    entities = {}
    entities["SpeakKeyword"] = {}
    entities["SpeakKeyword"]["values"] = [self._("speak")]
    entities["SpeakKeyword"]["question"] = self._sentence('What do you want me to do?')

    entities["Language"] = {}
    entities["Language"]["values"] = [self._("English"), self._("Spanish"), self._("Basque"), self._("Italian"),
                                      self._("German")]
    entities["Language"]["question"] = self._sentence("In which language?")

    return entities
```

Figura 8.1: Ejemplo *register_entites()* de ChangeLanguagePlugin.

- "register_intents()"

  Tal y como sucede en register_entities(), el resultado devuelto por esta función también deberá ser un diccionario. Cada clave del diccionario será un intent y el valor correspondiente a cada clave será otro diccionario con las siguientes opciones:

- “require”: lista de entities que deben figurar obligatoriamente en el utterance para que el intent sea aceptado.
- “optionally”: lista de entities que pueden aparecer de forma opcional en el utterance.
- “one_of”: lista de entities de los cuales es suficiente con que uno figure en el utterance para que el intent sea considerado.

En la imagen 8.2 se muestra el código de la función register_intents() del plugin ChangeLanguagePlugin.

```
@register_intents
def register_intents(self):
    intents = {}
    intents["change_language"] = {}
    intents["change_language"]["require"] = ["SpeakKeyword", "Language"]

    return intents
```

Figura 8.2: Ejemplo *register_intents()* de ChangeLanguagePlugin.

Ambas funciones deben ser decoradas con las funciones @register_entities y @register_intents importadas del módulo plugin_base.

Cabe recordar que todo desarrollador tiene a su disposición las funciones self._() y self._sentence() para que su plugin sea capaz de funcionar en todos los idiomas listados en el apartado 7.7.

9. CAPÍTULO

Experimento

Con el fin de poner a prueba la plataforma desarrollada en un entorno real, se ha realizado un pequeño experimento cuyos resultados se exponen en el presente capítulo.

El experimento ha consistido en pedir a cuatro voluntarios que resolvieran cinco situaciones diferentes a través de comandos de voz. Estos no tenían información sobre el modelo utterance-intent-entities y la única instrucción que han recibido es que para activar el sistema tenían que pronunciar la palabra "mayordomo", más allá de eso era la primera vez que interactuaban con la plataforma.

9.1. Usuarios

Los voluntarios elegidos para el experimento presentan edades y conocimientos informáticos diferentes, con el objetivo de que los resultados pudieran representar fielmente un hogar estándar.

La tabla 9.1 resume las características de cada usuario, aportando información sobre su edad, su perfil informático y el idioma en el que van a interactuar con la plataforma. Los nombres de los participantes han sido omitidos para preservar su anonimato.

Usuario	Edad (años)	Perfil informático	Idioma
Usuario 1	31	Medio	Gallego
Usuario 2	56	Bajo	Castellano
Usuario 3	57	Alto	Euskera
Usuario 4	25	Medio	Inglés

Tabla 9.1: Características de los usuarios.

9.2. Situaciones

Las situaciones planteadas han sido pensadas de antemano, pero los participantes en el experimento no han tenido acceso previo a ellas. Se han ido planteando individualmente: primero se enunciaba la situación, el usuario interactuaba con el sistema a través de voz y un vez resuelta, se continuaba con el siguiente problema. El motivo por el que se ha planteado así es para que el usuario respondiera de la forma más espontánea posible, sin detenerse a pensar qué es lo que iba a decir.

Los enunciados de las situaciones planteadas se presentan a continuación:

- Situación 1: "Quieres configurar el sistema en el idioma en el que más cómodo te encuentres. Si el sistema está configurado en castellano, ¿qué le dirías?"
- Situación 2: "¿Qué le dirías al sistema para que saludara a tu mejor amigo?"
- Situación 3: "Te acabas de acostar y te das cuenta de que te has dejado encendida la luz del pasillo, ¿qué comando le darías al sistema?"
- Situación 4: "Vas a salir de casa y no tienes claro si coger una chaqueta para abrigarte, ¿qué información puedes pedirle al sistema para que te ayude?"
- Situación 5: "Acabas de sentarte en el sofá, pero el mando está demasiado lejos, ¿cómo podrías encender la televisión?"

Como se puede ver, cada una de las situaciones describe un intent diferente. Además, se han incluido intents para cada uno de los plugins desarrollados, si bien el plugin que más representación obtiene es el relacionado con la automatización del hogar.

9.3. Resultados

Los resultados del experimento se representan en forma de tablas, una para cada uno de los usuarios. Las tablas contienen tres columnas: la primera lista las situaciones, la segunda contiene el comando de voz pronunciado por el usuario y en la cuarta se indica si el resultado es satisfactorio o no.

Al final de cada tabla se incluyen comentarios sobre los motivos por los que algunos comandos han fallado y las posibles soluciones.

9.3.1. Usuario 1

Situación	Comando	Resultado
Situación 1	¿Me puedes hablar en gallego?	✓
Situación 2	Saúda a Aitor	✗
Situación 3	Apaga a luz do corredor	✓
Situación 4	¿Cal e a temperatura da rúa?	✓
Situación 5	Acende a televisión	✓

Tabla 9.2: Resultados del Usuario 1.

El único comando erróneo ha sido el de la situación 2, ya que el sistema no ha incluido el verbo "saudar" ("saludar" en gallego) como traducción de "greet". Aún así, como en el comando se ha detectado un nombre propio, el sistema ha preguntado al usuario "Qué debo dicir?", y tras contestar éste que "ola", Aitor ha sido correctamente saludado. Cabe destacar que Mayordomo no posee ningún servicio de texto a voz en gallego, por lo que, tanto la pregunta como la respuesta final, han sido leídas en gallego pero con acento inglés. El sistema de interacción con el usuario ha funcionado correctamente.

9.3.2. Usuario 2

El segundo usuario no ha sido capaz de completar con éxito la última situación, debido a que WordNet no incluye el verbo "poner" como traducción de "switch on". Las tres posibles soluciones son: indicarle al usuario que debe usar el verbo "encender", traducir manualmente el verbo en el fichero ".po" correspondiente o ampliar el plugin openHAB2Plugin para contemplar el uso de "poner". En este caso, la opción más recomendable es indicarle al usuario el uso del otro verbo.

Situación	Comando	Resultado
Situación 1	Quiero que me hables siempre en castellano	✓
Situación 2	Saluda a mi amiga Ana	✓
Situación 3	Apaga la luz del pasillo en dos segundos	✓
Situación 4	¿Qué temperatura hace en la calle?	✓
Situación 5	Pon la televisión	✗

Tabla 9.3: Resultados del Usuario 2.

9.3.3. Usuario 3

Situación	Comando	Resultado
Situación 1	¿Puedes poner el idioma del sistema en euskera?	✗
Situación 2	Esaiozu kaixo Kepari	✓
Situación 3	Itzali, mesedez, korridoreko argia	✓
Situación 4	Zein da kaleko tenperatura?	✓
Situación 5	Piztu egongelako telebista	✓

Tabla 9.4: Resultados del Usuario 3.

De las cinco situaciones planteadas al usuario 3, sólo la primera de ellas (el cambio de idioma) presentó alguna dificultad. El plugin espera encontrar la palabra "hablar" y el nombre de uno de los idiomas soportados por la plataforma, pero en este caso, el verbo usado por el usuario ha sido "poner". De todos modos, el usuario fue capaz de completar el ejercicio cuando, tras la pregunta "¿Qué quieres que haga?", respondió "hablar en euskera". Nuevamente, el sistema de interacción con el usuario resultó de gran ayuda.

9.3.4. Usuario 4

Situación	Comando	Resultado
Situación 1	¿Podrías cambiar el idioma del sistema al inglés?	✗
Situación 2	This is Pepe, say hello to him	✓
Situación 3	Could you switch off the light in the corridor?	✓
Situación 4	What is the temperature outside?	✗
Situación 5	Could you turn on the television	✓

Tabla 9.5: Resultados del Usuario 4.

Por último, el sistema no ha sido capaz de completar dos de los comandos indicados por el cuarto usuario, aunque en ambos casos los errores son fácilmente subsanables. El primer problema surgió con el cambio de idioma al inglés, ya que el usuario hizo uso del verbo "cambiar" sin hacer mención al verbo "hablar". Tras ver qué tipo de frases son las más empleadas para cambiar el idioma, se sugiere editar el intent "change_language" y, en vez de marcar el entity "SpeakKeyword" como obligatorio, indicarle al sistema que debe aparecer al menos uno de los siguientes entities: "SpeakKeyword", "ChangeKeyword" o "LanguageKeyword". El error en la situación 4 se debe a que en openHAB, la variable (item) que contiene el dato de temperatura del exterior está definida como "calle", y el módulo traductor la ha traducido sólo como "street".

10. CAPÍTULO

Conclusiones

En este capítulo, último del informe, se presentan las conclusiones extraídas de la elaboración del proyecto, así como las posibles mejoras a implementar en un futuro.

10.1. Conclusiones generales

El trabajo realizado ha cumplido con su objetivo principal, que era completar el alcance propuesto al inicio del proyecto. La plataforma desarrollada es open source, permite enviar órdenes básicas para el control del hogar, es flexible y ampliable, solicita información al usuario mediante preguntas, recuerda órdenes previas, y, sobretodo, es multilenguaje. Esta última características convierte a Mayordomo en una plataforma única, ya que es uno de los primeros sistemas de voz que puede ser usado íntegramente en euskera.

Una de las grandes cualidades de la plataforma es que es una solución completa y no un módulo que ha de ser integrado dentro de otra solución. El sistema se encarga del proceso completo: desde la captura de audio a través de un micrófono USB, hasta la respuesta por voz. El usuario sólo tiene que hablarle al sistema y, pase lo que pase, recibirá siempre una respuesta indicándole el resultado final de su orden.

El proyecto también ha permitido profundizar en el uso de varias herramientas relacionadas con el procesamiento del lenguaje natural. Trabajar con programas como Freeling o IXA-PIPES permite comprender la dificultad a la que se enfrentan los investigadores de tecnologías NLP y, a pesar del trabajo que queda por recorrer, sirve para ver lo mu-

cho que se ha avanzado en los últimos años. Los conversores de texto a voz y de voz a texto también forman parte del abanico de temas en los que ha habido un gran desarrollo recientemente, sobretodo desde la popularización de las redes neuronal y el deep learning.

El resultado final del framework desarrollado también destacar por su capacidad para ser usado en un entorno real y servir de utilidad a muchos usuarios. Las tablas presentadas en el capítulo 9 muestran que la plataforma ofrece resultados satisfactorios y que los errores encontrados se pueden solucionar fácilmente con las herramientas facilitadas. La arquitectura implementada (cliente-servidor) permite que el usuario final pueda instalar, sin un gran desembolso económico, varios clientes distribuidos por toda la casa, para poder interactuar con el sistema desde cualquier estancia de la misma. El desarrollo de clientes en forma de aplicaciones móviles habilitará el uso programa desde cualquier parte del mundo, siempre que se disponga de una conexión a internet.

Por último, cabe destacar que, a pesar de que ha llevado más tiempo del previsto, ha sido posible superar uno de los obstáculos que ponían en riesgo la viabilidad de proyecto: compaginar la elaboración del PFG con el trabajo a tiempo completo.

10.2. Futuras mejoras

Pese a que la solución final es plenamente funcional, el desarrollo de Mayordomo no acaba aquí y sus capacidades pueden ser expandidas en futuros trabajos. Liberar el código en GitHub sirve, tanto para que cualquiera pueda analizarlo y corregirlo, como para se forme una comunidad alrededor del proyecto que ayude en el desarrollo de nuevas características. Entre todas las posibles mejoras, se van a destacar cuatro:

- Cliente para Android

  Implementar un cliente Mayordomo como una aplicación nativa de Android es uno de los siguientes pasos naturales en su evolución. Android dispone de APIs para los procesos de STT y TTS, librerías para la comunicación por MQTT y el desarrollo de aplicaciones se realiza en un lenguaje de programación conocido (Java), por lo que es una plataforma ideal para construir un segundo tipo de cliente.

- Interfaz web para editar y crear los plugins, entities e intents

  Si bien la plataforma facilita la creación de plugins a los desarrolladores, un usuario final que no sepa programar en Python difícilmente será capaz de crear nuevas

extensiones. Crear una aplicación web capaz de comunicarse con el servidor para visualizar, crear, editar o borrar entities e intents facilitaría la configuración del sistema y aumentaría el uso de la plataforma por parte de usuarios menos tecnófilos.

- Implementar un módulo STT basado en Kaldi

  Gracias a su sencillez de uso y a los múltiples idiomas soportados, para el desarrollo del cliente se ha hecho uso de la plataforma Google Cloud Speech en la conversión de voz a texto, pero existen herramientas open source con el mismo fin. Kaldi [35] es una librería escrita en C++ y liberada bajo licencia Apache 2.0 en la que se podría basar la implementación de un modulo STT que trabajara de forma local, sin depender de una conexión a internet.

- Múltiples comandos en una misma oración

  Por último, una característica que convendría añadir a Mayordomo es la capacidad de expresar varios comandos (intents) en una misma oración (utterance). La forma más sencilla de implementarlo sería separando los comandos mediantes una conjunción copulativa (“and”, “y”, “eta”,...) y que el servidor dividiera el utterance recibido usando dicha conjunción como separador. A partir de ese momento, el sistema trataría el utterance como si fueran dos distintos y los procesaría de forma independiente, haciendo uso de su gestión del contexto para resolver los dos intents satisfactoriamente.

  Por ejemplo, el utterance “Apaga la luz de la sala y del dormitorio” se convertiría en “apaga la luz de la sala” y “del dormitorio”. El primer intent sería fácilmente detectado, y el segundo también, porque el sistema guardaría los entities “apagar” y “luz” de la frase anterior. Dichos entities, junto a la palabra “dormitorio” presente en la oración, serían suficientes para activar el intent “set_light_state”.

Bibliografía

[1] Linguistic short introduction:. https://www.decodedscience.org/linguistics-short-intr

[2] Rodrigo Agerri, Josu Bermudez, and German Rigau. IXA pipeline: Efficient and Ready to Use Multilingual NLP tools. 2014.

[3] Chris Quirk, Raymond Mooney, and Michel Galley. Learning Semantic Parsers for If-This-Then-That Recipes. *Proceedings of 53rd ACL and 7th IJCNLP*, 2015.

[4] Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to Transform Natural to Formal Languages. *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.

[5] G. Ferreira, H. Macedo, L. Matos, A. Leandro, E. Seabra, W. Sampaio, A. Silva, T. Mendonça, and M. Soto. Semantic Validation of Uttered Commands in Voice-activated Home Automation. *Proceedings of the 14th International Conference on Artificial Intelligence ICAI'12*, 2012.

[6] API.AI. https://api.ai/.

[7] WIT.AI. https://wit.ai/.

[8] SEMPRE. https://nlp.stanford.edu/software/sempre/.

[9] ADAPT. https://adapt.mycroft.ai/.

[10] MQTT. http://mqtt.org/.

[11] Publish-Subscribe pattern. https://en.wikipedia.org/wiki/Publish-subscribe_pattern/

[12] Roger A Light. Mosquitto: server and client implementation of the MQTT protocol. *The Journal of Open Source Software*, 2(13), may 2017.

[13] Eclipse Paho. https://www.eclipse.org/paho/.

[14] Código fuente Mayordomo Servidor: . https://github.com/bodiroga/mayordomo-server.

[15] BCP-47. https://tools.ietf.org/html/bcp47.

[16] ISO-639-3. https://en.wikipedia.org/wiki/ISO_639-3.

[17] openHAB. https://www.openhab.org.

[18] Los items de openHAB. http://docs.openhab.org/configuration/items.html.

[19] Ivona TTS. https://www.ivona.com/.

[20] Ibon Saratxaga, Eva Navas, Inmaculada Hernáez, and Iker Luengo. Designing and recording an emotional speech database for corpus based synthesis in basque. 2006.

[21] Código fuente Mayordomo Cliente: . https://github.com/bodiroga/mayordomo-python-client.

[22] Google Cloud Speech (STT). https://cloud.google.com/speech/.

[23] Bing (Microsoft) STT. https://azure.microsoft.com/es-es/services/cognitive-services/sp

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

[25] Snowboy. https://snowboy.kitt.ai/.

[26] Wordnet. https://wordnet.princeton.edu/.

[27] Estadísticas Wordnet. http://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html.

[28] Open multilingual wordnet. http://compling.hss.ntu.edu.sg/omw/.

[29] Aitor Gonzalez-Agirre, Egoitz Laparra, and German Rigau. Multilingual central repository version 3.0: upgrading a very large lexical knowledge base. In *Proceedings of the 6th Global WordNet Conference (GWC 2012)*, Matsue, 2012.

[30] Elisabete Pociello, Eneko Agirre, and Izaskun Aldezabal. Methodology and construction of the Basque wordnet. *Language Resources and Evaluation*, 45(2):121–142, 2011.

[31] NLTK. http://www.nltk.org/howto/wordnet.html.

[32] py googletrans. https://github.com/ssut/py-googletrans.

[33] Gettext. https://www.gnu.org/software/gettext/.

[34] Freeling. http://nlp.lsi.upc.edu/freeling/node/1.

[35] Kaldi. http://kaldi-asr.org/doc/about.html.

Anexos

A. ANEXO

Código fuente

El código fuente del servidor puede ser consultado en el siguiente repositorio:
https://github.com/bodiroga/mayordomo-server

El código fuente del cliente se encuentra en:
https://github.com/bodiroga/mayordomo-python-client

B. ANEXO

Actas de reunión

B.1. 1ª reunión

Fecha: 20/09/2016

Asistentes: Aitor Iturrioz - Kepa Sarasola - Rodrigo Agerri

Tema principal: Primera toma de contacto

Resumen

Primera reunión para plantear la viabilidad del proyecto y definir el interés de Kepa y Rodrigo por tutorizar el PFG de Aitor. Se ha hecho un breve repaso por las características actuales de "Mayordomo" y cuáles serían las características nuevas a implementar durante el PFG. La idea consiste en sustituir el "Intent Parser" actual desarrollado en API.AI por un módulo open source que funcione en local. En una primera fase, el módulo tendrá un dominio limitado, ya que se centrará en reconocer "Intents" relacionados con la automatización del hogar; los comandos que podrá reconocer serán en castellano y tendrá que ser capaz de relacionar comandos actuales con comandos previos (contexto lingüístico). La posibilidad de hacerlo multilenguaje (Euskera) y de ampliar el dominio se valorará más adelante. Temas tratados pero que requieren de una mayor profundización: Adapt, WordNet, Machine Learning,...

Tareas

- Realizar un pequeño informe con la definición del proyecto, el alcance, los riesgos

y una planificación inicial. [Aitor]

- Estudiar la implementación y la API del programa "Adapt" desarrollado por Mycroft. [Aitor]
- Comenzar a analizar el estado del arte en el campo del "Semantic Parsing" enfocado a la automatización del hogar. [Aitor]

Duración: 60 minutos (17:00 - 18:00)

B.2. 2ª reunión

Fecha: 27/09/2016

Asistentes: Aitor Iturrioz - Kepa Sarasola - Rodrigo Agerri

Tema principal: Oficializar el proyecto en la plataforma GAUR

Resumen

El principal objetivo de la reunión era oficializar el proyecto subiendolo a la plataforma de GAUR. Tras revisar y confirmar la validez de la definición y el alcance redactados por Aitor, se ha procedido a subir el documento a GAUR, pero finalmente la gestión ha tenido que ser pospuesta hasta mañana por problemas técnicos. Desde el punto de vista más técnico, se han analizado los pros y los contras de "Adapt" como plataforma sobre la que basar el desarrollo del proyecto y, a pesar de sus carencias, se ha acordado seguir profundizando en el mismo. Uno de los mayores problemas que presenta es la "literalidad" de sus palabras claves y "Entities", por lo que se ha sugerido que una de las líneas de investigación del proyecto puede ser el uso de "lematizadores" para dotar de más abstracción al parser ("temperatura" vs "temperaturas"). También se han detectado problemas a la hora de parsear intents que no contienen Entities definidos como obligatorios (.require()), ya que en estos casos Adapt no devuelve ningún resultado, cuando la opción de preguntar al usuario por el Entity que falta podría ser una solución más adecuada. Por último, y debido al dominio tan limitado al que se va a responder el programa, se ha acordado hacer el programa lo más multilenguaje posible.

Tareas

- Subir el proyecto a GAUR y oficializarlo. [Rodri]

- Analizar la API del intent_recognizer de Adapt para entender hasta qué punto el framework hace uso de un simple pattern machine o realmente hay algo de deep learning detrás. [Aitor]
- Leer los papers propuestos y analizar su viabilidad y uso como parte del proyecto. [Aitor]

Duración: 90 minutos (17:00 - 18:30)

B.3. 3ª reunión

Fecha: 27/02/2017

Asistentes: Aitor Iturrioz - Rodrigo Agerri

Tema principal: Vuelta al trabajo y exposición del trabajo realizado

Resumen

El motivo principal para esta tercera reunión era retomar el proyecto después de varios meses de inactividad y presentar el trabajo realizado hasta la fecha. Se han descrito las mejoras realizadas en el framework Adapt: lematizar la frase antes de pasarla por el parser, introducir la opción de realizar preguntas al usuario cuando no se hayan detectado todos los "entities" de un "intent", crear sesiones de usuario de 5 minutos de duración, guardar los "entities" detectados en frases previas y que sirvan como contexto para las futuras frases,... Por otro lado, dentro del framework Mayordomo, pero fuera de Adapt, se ha reescrito el funcionamiento principal del programa para dividirlo en pequeños módulos con el fin de que el programa sea más fácil de mantener y modificar. Además, se ha creado el concepto de "device" y se ha ideado un sistema para que los dispositivos se registren en el sistema, los proveedores de "entities" e "intents" se registran como plugins y se pueden ampliar de forma sencilla, se ha dotado al sistema de un módulo TTS extensible (por ahora sólo hay un plugin de Ivona) del que todos los dispositivos pueden hacer uso mediante MQTT,... Fuera del core de mayordomo, se ha desarrollado un primer cliente (dispositivo) escrito en Python y que se comunica con el servidor mediante MQTT. Para la conversión de voz a texto (STT) se hace uso de dos herramientas: Snowboy (para la detección offline de palabras clave que hagan que el sistema se active -24/7 a la escucha-) y Google Cloud Speech (para la conversión real de voz a texto). Se ha usado la opción de streaming en Google Cloud Speech para disminuir al máximo el retardo, ya que de

esta forma el audio se va procesando en tiempo real mientras se va grabando, y no hay que esperar a que el usuario termine de hablar antes de enviar el audio a los servidores de Google (0.5s de mejora).

De cara a la mejora del proyecto, se ha hablado de la posibilidad de introducir Wordnet como herramienta del framework (a través de NLTK) y hacer uso de los synsets para buscar sinónimos de los "entities" registrados, con el fin de permitir más flexibilidad a la hora de realizar las peticiones por voz. El ejemplo más claro es que, si hemos registrado "sala" como un "entity", demos validez a que el usuario diga "salón" por ser palabras sinónimas. También se valorará el uso de hipónimos e hiperónimos para tal fin. Queda pendiente comenzar a desarrollar diferentes plugins (openHAB2.0, dbus, spotify,...) para enriquecer el sistema.

Tareas

- Añadir Wordnet al sistema para que busque sinónimos de los entities. [Aitor]
- Comenzar a diseñar el plugin de openHAB2.0 [Aitor]
- Analizar los requisitos mínimos para presentar el proyecto como un proyecto de computación [Kepa]

Duración: 90 minutos (15:30 - 17:00)

B.4. 4ª reunión

Fecha: 13/03/2017

Asistentes: Aitor Iturrioz - Rodrigo Agerri

Tema principal: Exposición del trabajo realizado (con Rodri)

Resumen

Repaso del trabajo realizado hasta la fecha y del estado actual del proyecto con Rodri (no se ha podido hacer una demo por problemas con la conexión WiFi).

Tras esto, se ha hablado sobre los próximos pasos a dar y, tal y como se habló con Kepa, se ha llegado a la conclusión de que la integración de Wordnet a través de NLTK es clave. Para conseguir que "mayordomo" sea lo más flexible y multilenguaje posible, es necesario

que el sistema sea capaz de extraer los lemas de cada intent y cada palabra de un utterance para ver posibles relaciones no definidas de antemano. Estas relaciones se "mapearán" automáticamente a través del lemma de las palabras y buscando su equivalente en varios idiomas a través de "open multilingual wordnet". Para que el programa pueda funcionar completamente en Euskera, es indispensable introducir un lematizador (ixa-pipes) y un servicio tts (ahotts) que lo soporten.

También se ha comentado la posibilidad de añadir entities predefinidos de carácter temporal: hora concretas (dentro de 5 minutos, a las 3 de la tarde,...), fechas concretas (mañana, el próximo jueves, el día 15 de octubre,...) fechas y horas concretas (mañana a las cinco menos cuarto, dentro de dos lunes a las seis de la tarde,...) o intervalos de fechas (durante una semana, hasta dentro de una hora,...). Es un tema ya trabajado en la facultad, pero su aplicación será opcional y dependerá del desarrollo del proyecto.

Por último, se ha mencionado el tema del "text generation", crear frases sintácticamente correctas a través de palabras clave, pero su introducción se ha descartado debido a la complejidad del tema.

Tareas

- Añadir Wordnet al sistema para que busque sinónimos de los entities. [Aitor]
- Añadir AhoTTS al sistema. [Aitor]
- Finalizar el plugin de openHAB2.0 [Aitor]

Duración: 70 minutos (15:30 - 16:40)

B.5. 5ª reunión

Fecha: 15/05/2017

Asistentes: Aitor Iturrioz - Rodrigo Agerri - Kepa Sarasola

Tema principal: Nueva actualización del estado del proyecto

Resumen

Repaso del trabajo realizado hasta la fecha y del estado actual del proyecto con Rodri y Kepa.

Nuevos añadidos del sistema:

- AhoTTS para reproducir texto a voz (TTS) en Euskera.
- IXA-PIPES para el proceso de tok, pos y lematización en Euskera (también se han incluído el resto de idiomas soportados por la herramienta, pero Freeling seguirá teniendo prioridad).
- Capacidad de cambiar el idioma del sistema en tiempo real y que vuelva a cargar las herramientas necesarias para ello.
- Capacidad de lanzar el STT (Google Cloud Speech) del cliente en el idioma del sistema.

Después de una larga charla y de barajar varias opciones, se llegó a la conclusión de que la forma de dotar de capacidad multilenguaje al sistema será la siguiente:

- Los 'entities' de los plugins se deberán definir, por defecto, en inglés. Aprovechando las capacidades de internacionalización de python (i18n) se podrán crear diferentes traducciones de dichos 'entities' y distribuirlos junto al plugin (carpeta Locale en cada plugin).
- En caso de que los 'entities' del plugin no estén traducidos, se usará Wordnet para extraer todos los 'lemmas' en el idioma correspondiente e introducirlos en Adapt. Este proceso se hará en tiempo real cuando el programa arranque. El sistema de esta forma introducirá 'lemmas' que no corresponderán con el 'synset' que nosotros estamos buscando, pero el riesgo está justificado.

También se ha discutido la posibilidad de buscar prospectos que hablen y listen comandos empleados en la automatización del hogar para poder automatizar el proceso de emparejar comandos en distintos idiomas.

Tareas

- Añadir Wordnet al sistema para buscar 'lemmas' en otros idiomas a la hora de hacer multilenguaje el programa. [Aitor]
- Añadir Wordnet al sistema para que busque sinónimos de los entities. [Aitor]

Duración: 150 minutos (15:30 - 18:00)

B.6. 6ª reunión

Fecha: 29/05/2017

Asistentes: Aitor Iturrioz - Rodrigo Agerri - Kepa Sarasola

Tema principal: Actualización del estado del proyecto

Resumen

Durante estas dos semanas se ha completado todo el trabajo acordado en la última reunión. A saber:

- Se ha añadido la opción de traducir los entities y las frases de respuesta a mano usando archivos .mo y .po.
- Para la traducción automática, se ha hecho el mapeo automático de conceptos mediante la herramienta open multiligual wordnet del paquete NLTK. Para los casos en los que WordNet no devuelve ningún resultado, el sistema hace uso de Google Translate a través de la librería py-googletrans para completar la traducción.
- Se ha desarrollado un plugin nuevo que es capaz de cambiar el sistema de un idioma a otro por comandos de voz.

Para demostrar el correcto funcionamiento del programa, se ha hecho una demo en directo mostrando las capacidades del sistema, conmutando entre diferentes idiomas y ejecutando comandos incluidos en el mismo. El resultado ha sido satisfactorio para Rodri y Kepa.

A partir de ahora, se dedicará todo el tiempo disponible a la redacción del informe del proyecto.

Cabe destacar que para darle más solidez al trabajo realizado, se hará un pequeño experimento con 2-3 voluntarios. El experimento consistirá en plantear entre 5 y 10 acciones a realizar a través del asistente, anotar la frase 'hablada' por el usuario, guardar el resultado devuelto por el sistema y elaborar una tabla con los resultados obtenidos.

Tareas

- Arreglar posibles bugs. [Aitor]
- Redactar el informe del proyecto. [Aitor]

Duración: 60 minutos (16:00 - 17:00)