PROYECTO FIN DE GRADO

TÍTULO: Integración de un asistente de voz abierto y multimodal para el Hogar Digital Accesible

AUTOR: Pablo Pérez Alfonsín

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Miguel Ángel Valero Duboy

DEPARTAMENTO: DTE, Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Nicolás Sáenz Lechón

TUTOR/A: Miguel Ángel Valero Duboy

SECRETARIO/A: Javier Malagón Hernández

Fecha de lectura: 18 JULIO 2025

Calificación:

El Secretario/La Secretaria,

Resumen

Este proyecto tiene como propósito, dentro del contexto del Hogar Digital Accesible (HDA), el diseño e implementación de un asistente de voz local, multimodal y basado en tecnologías libres y de código abierto. Su objetivo es permitir y facilitar la interacción del usuario con los distintos dispositivos domóticos del hogar a través de comandos de voz de manera sencilla, privada y flexible.

A diferencia de soluciones comerciales como Alexa o Google Home, actualmente instalado en el HDA, el asistente desarrollado prescinde de servicios conectados a internet y garantiza un funcionamiento completamente offline, preservando así la privacidad del usuario y ofreciendo un mayor grado de modularidad, personalización y control sobre el sistema. Además, permite esquivar las limitaciones de compatibilidad con ciertos dispositivos impuesta por los fabricantes de cara a la ampliación del ecosistema del HDA.

Tras una fase previa de estudio comparativo entre distintas plataformas y soluciones disponibles, se ha optado por utilizar la herramienta Rhasspy como base del sistema, por su arquitectura modular, integración con plataformas de monitorización domótica como OpenHAB y la posibilidad de configurar y adaptar cada uno de los servicios del asistente de forma independiente y modular adaptándose a las necesidades y recursos disponibles.

Para alojar el sistema completo se ha empleado una Raspberry Pi, dedicada exclusivamente a la ejecución y funcionamiento del asistente, separándola de la Raspberry Pi ya instalada en el HDA, que alberga y ejecuta OpenHAB. Esta decisión responde a la necesidad de garantizar un rendimiento adecuado y evitar interferencias con otros servicios ya desplegados en el HDA. Gracias a esta elección, el asistente puede gestionar de manera eficaz los recursos del sistema y mantener un tiempo de respuesta fluido para el usuario.

El resultado de todo esto es un sistema robusto, accesible y escalable, que permite al usuario interactuar con el hogar digital de manera natural a través de la voz, promoviendo así la autonomía de las personas y la adaptabilidad tecnológica bajo criterios de sostenibilidad, eficiencia y ética en el tratamiento de los datos personales.

Abstract

The purpose of this project, within the context of the Accessible Digital Home (ADH), is the design and implementation of a local, multimodal voice assistant based on free and open source technologies. Its objective is to enable and facilitate user interaction with the different home automation devices in the home through voice commands in a simple, private and flexible way.

Unlike commercial solutions such as Alexa or Google Home, currently installed in the HDA, the developed assistant dispenses with internet-connected services and guarantees completely offline operation, thus preserving user privacy and offering a higher degree of modularity, customisation and control over the system. In addition, it circumvents the limitations of compatibility with certain devices imposed by manufacturers in order to expand the HDA ecosystem.

After a previous phase of comparative study between different platforms and solutions available, the Rhasspy tool was chosen as the basis of the system, due to its modular architecture, integration with home automation monitoring platforms such as OpenHAB and the possibility of configuring and adapting each of the assistant's services independently and modularly, adapting to the needs and resources available.

To host the complete system, a Raspberry Pi has been used, dedicated exclusively to the execution and operation of the assistant, separating it from the Raspberry Pi already installed in the HDA, which hosts and runs OpenHAB. This decision responds to the need to ensure adequate performance and avoid interference with other services already deployed in the HDA. Thanks to this choice, the wizard can efficiently manage system resources and maintain a smooth response time for the user.

The result of all this is a robust, accessible and scalable system, which allows the user to interact with the digital home in a natural way through voice, thus promoting people's autonomy and technological adaptability under criteria of sustainability, efficiency and ethics in the treatment of personal data.

Índice de contenidos

Índice de figuras

Índice de tablas

Lista de acrónimos

ADH: Accessible Digital Home

AI: Artificial Intelligence (Inteligencia Artificial)

ALSA: Advanced Linux Sound Architecture

API: Application Programming Interface

ARM: Advanced RISC Machines

ARPA: Advanced Research Projects Agency

ASR: Automatic Speech Recognition

CPU: Central Processing Unit

DHCP: Dynamic Host Configuration Protocol

DIY: Do It Yourself

DNS: Domain Name System

GB: Gigabyte

GPIO: General Purpose Intpu/Output

GPU: Graphics Processing Unit

HDA: Hogar Digital Accesible

HDMI: High-Definition Multimedia Interface

IA: Inteligencia Artificial

IP: Internet Protocol

ISO: International Organization for Standardization

JSON: JavaScript Object Notion

MQTT: Message Queuing Telemetry Transport

ODS: Objetivos de Desarrollo Sostenible

ONU: Organización de las Naciones Unidas

ONNX: Open Natural Network Exchange

OS: Operating System

PING: Packet Internet Groper

RAM: Random Access Memory

REST: Representational State Transfer

SD: Secure Digital

SO: Sistema Operativo

SSH: Secure Shell

SSD: Solid State Drive

STT: Speech-to-Text

TTS: Text-to-Speech

UDP: User Datagram Protocol

URL: Uniform Resource Locator

USB: Universal Serial Bus

WAV: Waveform Audio File Format

WSL: Windows Subsystem for Linux

YAML: Yet Another Markup Language

1. Introducción

1.1 Marco y motivación del proyecto

En la última década, los avances, evolución y desarrollo de la tecnología doméstica ha facilitado la incorporación de soluciones inteligentes en el ámbito del hogar. La combinación de microcontroladores, sensores y actuadores, herramientas de monitorización y automatización y la proliferación de asistentes de voz ha dado lugar a un ecosistema domótico cada vez más presentes en la vida cotidiana y en los hogares. Sin embargo, gran parte de estas soluciones están dominadas o monopolizadas por entidades y plataformas comerciales que, si bien ofrecen funcionalidades avanzadas en formato plug-and-play, presentan importantes limitaciones en cuanto a aspectos como la privacidad, personalización o dependencia tecnológica.

Este proyecto se desarrolla en el ámbito del **Hogar Digital Accesible (HDA)**, un laboratorio de innovación tecnológica del Campus Sur de la UPM orientado a promover el desarrollo e implementación de soluciones domóticas abiertas, sostenibles y replicables con el objetivo de hacer más accesible y humana la interacción de las personas con el hogar. En este contexto, se identifica la necesidad de integrar un asistente de voz local y multimodal, que permita interactuar de manera simple con los dispositivos domóticos del hogar sin necesidad de conexión a internet y siendo agnóstico de tecnologías o fabricante propietarios.

Todo el entorno domótico del HDA está cimentado sobre OpenHAB, una plataforma open source que permite el control y monitorización de dispositivos a través de una arquitectura flexible y ampliable. Actualmente, se está empleando como interfaz de voz el dispositivo Google Home que, aunque práctico y sencillo, su uso acarrea una serie de inconvenientes tecnológicos, técnicos y de privacidad. Estos problemas funcionales motivas la creación e implantación de un asistente de voz que, además de garantizar la privacidad, sea fácilmente adaptable y ejecutable en computadoras de bajo consumo y recursos limitados, como una Raspberry Pi.

Para llevarlo a cabo, se ha optado emplear Rhasspy, una herramienta open source y local que permite construir y desarrollar asistentes de voz configurables y modulables y que pueden ser integrados con plataformas de monitorización como OpenHAB sin necesidad de conexiones externas. La arquitectura modular y la compatibilidad con diversos motores de reconocimiento y/o síntesis de voz permiten ajustar y adaptar el sistema al nivel de recursos y complejidad deseados.

Así, este proyecto se orienta en dar solución a un problema concreto como es dotar al HDA de una interfaz de voz rápida, eficiente, ética y accesible, que complete la infraestructura ya existente y sirva como base para futuros desarrollos o ampliaciones de las soluciones ya existentes.

1.2 Objetivos técnicos y académicos

Los objetivos de este proyecto fin de carrera son, desde el punto de vista técnico:

- Desarrollar un asistente de voz independiente de servicios en la nube, conexión a internet o a otros servicios externos y que funcione de forma completamente local.
- Utilizar hardware y equipamiento de bajo coste para garantizar la sostenibilidad, escalabilidad futura y la viabilidad económica del proyecto. En concreto emplear una Raspberry Pi.
- Evaluar distintas tecnologías open source disponibles en el marco tecnológico actual y seleccionar y estudiar aquellas más adecuadas para el ámbito del HDA
- Conseguir implementar una solución adaptable que permita configurar y adaptar los distintos servicios y componentes del asistente a los recursos disponibles y las necesidades.
- Diseñar una arquitectura modular y escalable que permita modificar, actualizar o reemplazar componentes individuales del asistente de voz sin necesidad de rediseñar el conjunto completo
- Asegurar la compatibilidad del asistente propuesto con OpenHAB, posibilitando el control por voz de sensores, actuadores y otros dispositivos domóticos del hogar ya monitorizados en la plataforma.
- Minimizar el tiempo de procesamiento y la latencia entre la orden de voz y la respuesta del asistente para conseguir una ejecución fluida y natural.

Los objetivos de este proyecto de fin de grado son, desde el punto de vista técnico:

- Desarrollar habilidades clave en la gestión, desarrollo e implementación de sistemas embebidos, procesamiento de voz y comunicaciones entre dispositivos.
- Adquirir experiencia en el uso e integración de protocolos de comunicación orientados al Intenet of Things como MQTT, WebSockets o API Rest
- Desarrollar destrezas en el desarrollo de scripts de automatización y monitorización de dispositivos domóticos.
- Aplicar de forma práctica los conocimientos y destrezas adquiridos durante el Grado en Ingeniería Telemática en un caso real de integración tecnológica.
- Promover la autonomía en la investigación y selección de tecnologías libres ampliando la capacidad de autogestión en el desarrollo de tecnologías accesibles.
- Fomentar la concienciación sobre el valor y la importancia de la privacidad, la ética digital y la accesibilidad como vectores del desarrollo tecnológico presente y futuro.

1.3 Estructura del resto de la memoria

Los próximos capítulos de la memoria se estructuran y ofrecen información sobre los siguientes aspectos:

- **2. Marco Tecnológico**: Se analiza en qué punto se encuentra el estado del arte actual sobre las tecnologías a explorar y se describen las principales a utilizar en él. Se detalla el uso de los microordenadores Raspberry Pi y se revisan sus modelos más relevantes.

Se explica el funcionamiento de la plataforma de automatización domótica OpenHAB y se presentan herramientas útiles para el desarrollo del asistente, como Mycroft AI, Rhasspy, Whisper AI o Piper TTS, atendiendo a sus funcionalidades y características principales.

- **3. Especificaciones y restricciones de diseño**: Se exponen y definen los requisitos funcionales del sistema, así como las limitaciones impuestas por diversos elementos como el entorno, el hardware disponible y los objetivos del proyecto.
- **4. Descripción de la solución propuesta**: Se exponen y detallan las elecciones y decisiones de implementación del asistente de voz, desde la elección del hardware adecuado hasta la implementación de los servicios para cubrir todas las funcionalidades necesarias para el desarrollo del asistente y su integración con OpenHAB.
- **5. Resultados**: Se realiza un análisis de los resultados obtenidos tras la implementación del asistente y la realización de pruebas, destacando los aspectos de rendimientos, calidad del reconocimiento de voz o la fluidez en la interacción con el usuario.
- **6. Presupuesto**: Se presenta un desglose detallado de los costes estimados del proyecto completo, tanto en componentes y materiales como en recursos humanos y de desarrollo.
- **7. Impacto del proyecto**: Se hace una reflexión sobre los posibles impactos y beneficios de la solución creada desde los puntos de vista tecnológico, ético y social, así como su aportación en entornos domésticos o académicos.
- **8. Conclusiones**: Se resumen los aprendizajes adquiridos, se hace una valoración completa de los resultados obtenidos y se proponen o sugieren líneas de mejora y evolución de la solución propuesta.
- **9. Referencias**: Se incluye de manera estructurada todas las fuentes bibliográficas y documentales empleadas durante el desarrollo de todo el trabajo de fin de grado.
- **Anexos**: Se adjuntan y añaden documentación y material adicional como scripts, código, manuales de usuario o configuraciones relevantes de dispositivos.

2. Marco tecnológico

En este apartado se van a presentar y exponer las distintas tecnologías y herramientas empleadas en el desarrollo del proyecto.

2.1 Raspberry Pi

La Raspberry PI es un microordenador de placa única y bajo costo desarrollada por la Raspberry Pi Foundation en 2012 con la finalidad de democratizar y acercar la informática a todo el mundo, pero acabó transcendiendo ese objetivo inicial. Las posibilidades que ofrece para programarla y modificarla con diversos accesorios la dotan de gran versatilidad para utilizarla para una amplia variedad de proyectos. Esto, unido a la capacidad de ejecutar sistemas operativos basados en Linux, son los principales motivos de su gran popularidad. Además, poco después de su lanzamiento inicial, la Raspberry Pi ya contaba con una comunidad de miles de personas que no ha hecho más que crecer a lo largo de los años. [1] [2]

2.1.1 Características Técnicas Principales

A pesar de su pequeño tamaño, este microordenador goza de las siguientes especificaciones técnicas generales: [3] [4]

- Componentes: Con un tamaño similar al de una tarjeta de crédito, dispone de un procesador ARM, memoria RAM (varía en función del modelo) y un procesador gráfico Broadcom VideoCore
- Conectividad: Dependiendo del modelo puede ofrecer puertos USB, HDMI, Ethernet y GPIO para poder conectar microcontroladores y dispositivos electrónicos. Además, las últimas versiones pueden ofrecer también WiFi o Bluetooth.
- Almacenamiento: No disponen de almacenamiento interno, utilizan una tarjeta microSD para almacenar el sistema operativo y datos. Puede ampliarse con discos duros externos a través de USB.
- Sistema Operativo: El sistema operativo más utilizado es Raspberry Pi OS, anteriormente Raspbian. Es una versión de Linux, basada en Debian, distribuida de manera oficial por la Raspberry Pi Foundation. Permite la instalación de otros sistemas operativos compatibles.

2.1.2 Últimos Modelos

Se presenta a continuación un resumen detallado de los últimos modelos de Raspberry Pi comercializados en los últimos años.

2.1.2.1 Raspberry Pi 3 Model B+

Lanzada al mercado en 2018 como una evolución de su antecesora la Raspberry Pi 3. Cuenta con un procesador Broadcom Cortex-A53 de cuatro núcleos a 1.4 GHz. Al igual que el modelo anterior, incorpora arquitectura de 64 bits y memoria RAM de 1GB. En cuanto a conectividades, ofrece conexión WiFi (bandas 2.4GHz y 5GHz), Bluetooth 4.2 e incorpora puerto Gigabit Ethernet. Mantiene los puertos USB, HDMI y GPIO para conectarse a equipos electrónicos [5]

2.1.2.2 Raspberry Pi 3 Model A+

Este modelo es una versión simplificada del anterior, más pequeña, barata y de menor consumo. Está orientada a proyectos más simples donde estos últimos factores sean relevantes. A diferencia del Model B+, este modelo dispone sólo de 512MB de memoria RAM y reduce las prestaciones en cuanto a conectividad con sólo 1 puerto USB y sin puerto Gigabit Ethernet. [6]

2.1.2.3 Raspberry Pi 4 Model B

Se lanzó en 2019 al mercado presentando avances importantes en relación a sus predecesores. El procesador se actualiza al Broadcom Cortex-A72 de cuatro núcleos a 1.5GHz. En cuanto a memoria, se puede elegir entre versiones de 2, 4 o 8 GB de RAM LPDDR4. En conectividad, aumenta las prestaciones incorporando puertos 2 x USB 3.0 y 2 x USB 2.0, USB tipo C para la alimentación y doble entrada micro-HDMI (4K @ 60Hz)

2.1.2.4 Raspberry Pi 5

Último modelo de Raspberry Pi disponible que se lanzó en 2023. Supone un salto cualitativo en la capacidad de procesamiento gracias al nuevo procesador Broadcom Cortex-A76 con cuatro núcleos a 2.4GHz junto con la GPU VideoCore VII (800MHz), que la hacen hasta 3 veces más potente que su modelo anterior. Tiene opciones de memoria RAM de 4, 8 o 16 GB LPDDR4X.

Es la primera Raspberry Pi que dispone de soporte dedicado para SSD NVMe mediante PCIe 2.0 además del almacenamiento estándar de la tarjeta microSD. En conectividades, mantiene WiFi 5 (802.11ac), USB tipo C, HDMI y puerto Gigabit Ethernet, pero mejora con Bluetooth 5.0. Los puertos USB se mantienen igual, ofreciendo los USB 3.0 tasas de transmisión de 5 Gbps. [7]

El aumento de prestaciones hace que esta versión sea óptima para usos avanzados como desarrollos de IA, soporte para servicios ligeros o automatizaciones y el trabajo en tiempo real con periféricos.

2.2 OpenHAB

The open Home Automation Bus, comúnmente llamada OpenHAB, es una plataforma de código abierto, escrita en Java, orientada y diseñada a la automatización de la domótica del hogar. El objetivo principal de OpenHAB es actuar como centro de control unificando, bajo una misma interfaz, múltiples dispositivos, permitiendo crear ecosistemas domóticos personalizables y escalables. [8]

Entre las principales ventajas o puntos clave de OpenHAB destaca principalmente su independencia tecnológica, en cuanto a la capacidad de integrar y automatizar dispositivos y equipos domóticos de diferentes proveedores o fabricantes en una única herramienta. Además, permite implementar múltiples tecnologías y protocolos de comunicación distintos como Z-Wave, MQTT, HTTP o Zigbee, dotándola de una gran versatilidad. [9]

OpenHAB es una herramienta multiplataforma que permite su instalación tanto en Windows, Linux o sistemas embebidos como una Raspberry Pi.

***Figura 1**. OpenHAB*

2.2.1 Componentes Principales

Esta plataforma permite ejecutar órdenes definidas por el usuario sobre los dispositivos domóticos. Para facilitar todo esto, OpenHAB posee una arquitectura estructurada en una serie de conceptos básicos que se definen a continuación:

2.2.1.1 Things

También denominados entidades, representan los dispositivos físicos (sensores, actuadores…) o servicios web que OpenHAB puede integrar y gestionar y que pueden ofrecer una o varias funcionalidades o servicios. Definir un Thing es el primer paso a la hora de integrar un dispositivo, que supone configurar la conexión con la capa física. [10]

2.2.1.2 Channels

Son el punto de conexión entre los Things y los Items. Los Things proporcionan un Channel en representación de cada una de las funciones o servicios que ofrecen para poder acceder a ellos. Conforman el nexo entre la capa física, el dispositivo, y la capa virtual, el Item asociado a una función y, además, es bidireccional, transportando eventos y comandos de los Things a los Items vinculados y viceversa. [11]

2.2.1.3 Bindings

Los Bindings son adaptadores de software que permiten la conexión y comunicación entre el dispositivo físico y OpenHAB, concretamente con su Thing. Cada tecnología o protocolo de comunicación (HTTP, Z-Wave, Zigbee…) necesita por tanto de su Binding específico en OpenHAB para poder comunicarse con los dispositivos.

Este es uno de los elementos esenciales que permiten la interoperabilidad de OpenHAB con multitud de equipos independientemente de la marca o protocolo de comunicación que utilicen y se encuentra en continuo desarrollo para poder implementar dispositivos de nuevas tecnologías. [8]

2.2.1.4 Items

Representan las propiedades o capacidades de un Thing o dispositivo físico, vinculadas a cada uno de los Channels, y que pueden ser utilizadas, ejecutadas o consultadas desde OpenHAB.

Cada Item poseen un valor o estado según la propiedad o funcionalidad que representan y estos pueden ser modificados o consultados a través del envío de eventos. [12]

2.2.1.5 Interfaces de Usuario

Desde la incorporación de la versión OpenHAB 3, para cada instancia de la herramienta se pueden configurar y mostrar distintas interfaces de usuarios en función de quién vaya a utilizarla, para qué y en bajo qué condiciones. A continuación, se detallan las principales interfaces de usuario disponibles:

MainUI: Interfaz principal de OpenHAB introducida en la versión 3 de la herramienta para la administración, control y monitorización de los dispositivos del hogar inteligente de manera sencilla. Permite distinguir entre usuarios simples, personalizables y con acceso a elementos interactivos y de gestión del hogar, y usuarios administradores.

Los administradores dispondrán de acceso completo a toda la configuración de la herramienta, la tienda de Add-Ons para adquirir nuevas herramientas y/o aplicaciones y herramientas para desarrolladores que ofrecen aún más personalización de la interfaz. [13]

Pages: Es el sistema de páginas personalizables que ofrece el MainUI con distintos formatos. La configuración de estas páginas se guarda y almacena en formato JSON.

Los principales tipos de páginas son *Layout Pages*, páginas con widgets editables como gráficos, controles etc; *Map/Floorpan Pages*, planos o mapas interactivos de viviendas o estancias donde se pueden representar interactivamente los dispositivos y su ubicación; y los *Chart Pages*, gráficos que permiten ver el históricos de algunos datos con filtro temporales [14]

Sitemaps: Sistema de interfaces tradicional de OpenHAB para definir y crear interfaces personalizadas mediante sintaxis declarativa en archivos de texto (.sitemap). Los sitemaps son, al contrario que MainUI, una opción más sencilla y ligera, por lo que forman parte de la interfaz web minimalista *BasicUI* y además son también utilizados en la aplicación móvil de OpenHAB.

HABPanel: Es una interfaz simple y ligera pensada para utilizarse en tablets u otro tipo de pantallas táctiles. Similar a un *Page*, permite editar widgets arrastrables para diseñar dashboards personalizables y adaptados a las necesidades del usuario sin la necesidad de usar código. Dispone de un modo de edición en el que configurar el panel directamente desde la pantalla sin tener que acceder a la configuración de OpenHAB. [15]

2.2.1.6 Rules

Son el elemento principal de la automatización domótica, permitiendo definir reglas inteligentes basándose en eventos, condiciones o acciones. Son scripts o programas que pueden ejecutar o realizar una o varias tareas automáticas al detectar los eventos definidos para ello. Permiten por lo tanto dotar de un cierto grado de inteligencia a los dispositivos monitorizados en OpenHAB.

Todas las reglas están compuestas como mínimo por un *trigger*, o elemento activador de la regla, y las acciones a realizar al activarse. Adicionalmente pueden incluir condiciones lógicas previas a la activación para limitar o definir más la regla. [16]

2.2.2 Interfaces de Comunicación

OpenHAB proporciona diversas maneras de interacción y comunicación con los distintos Items y dispositivos domóticos integrados. Entre ellas se destacan las siguientes:

2.2.2.1 API Rest

Interfaz principal basada en la comunicación a través del protocolo HTTP. Permite interactuar con OpenHAB y los Items monitorizados de manera externa desde cualquier lugar y dispositivo que soporte esta tecnología.

A través del envío de peticiones HTTP, permite consultar o actualizar el estado de los *Items*, gestionar la configuración de *Things* y *Bindings* y ejecutar reglas o acciones. Ofrece soporte además para la autenticación de usuarios a través de tokens API o credenciales.

2.2.2.2 WebSockets

Interfaz bidireccional que proporciona una comunicación bidireccional en tiempo real con el *Event Bus* de OpenHAB. Mediante la suscripción a este elemento, permite notificar de manera instantánea los cambios y eventos de los dispositivos monitorizados sin necesidad de realizar polling repetidamente para cada cambio. Por esto mismo, ofrece una velocidad de respuesta mucho mayor que la API Rest.

La suscripción puede realizarse para los eventos de todos los eventos o sólo los de algunos dispositivos. Todos los mensajes por WebSocket se envían en formato JSON, lo que permite su uso y transformación posterior y, al igual que la API Rest, ofrece métodos de autenticación a través de tokens o credenciales. [17]

2.3 Mycroft AI

Mycroft AI es un asistente de voz de código abierto que tiene como objetivo priorizar la privacidad y la personalización. Se desarrolló para usarse en sistemas operativos Linux pensado como una alternativa completa y ética a soluciones comerciales existentes como Alexa o Siri.

Es considerado el primer asistente de voz virtual completamente open source y que permite al usuario tomar control sobre el funcionamiento y las acciones a realizar por parte del asistente de forma personalizada. Dispone de soporte para varios idiomas y permite su integración con diversas plataformas de monitorización de dispositivos domóticos como OpenHAB o Home Assistant, así como directamente con servicios web como Spotify o YouTube. [18]

El objetivo principal de Mycroft AI es la privacidad del usuario. Por un lado, el procesamiento de voz e información se realiza de forma totalmente local, aunque para algunos servicios es necesario la conexión a servidores propios de Mycroft. Por otro lado, aseguran que las grabaciones de voz se realizan bajo demando u orden explícita del usuario y que en ningún momento estas son almacenadas por la empresa.

Aunque Mycroft ofrece algunos productos comerciales ya completos, como el Mark II, un altavoz con el asistente de voz Mycroft AI ya incorporado; la eficiencia y optimización de su

software permite correr Mycroft AI en dispositivos con poca capacidad de recursos como una Raspberry Pi

El Mycroft Core es el motor principal del asistente de voz, que integra el *parser* de intenciones Adapt, la conversión de texto-voz, así como el reconocimiento de voz para la detección y grabación del audio. Además, cuenta con un ecosistema de Skills, habilidades personalizadas que se pueden desarrollar a través de Mycroft AI para dotar de un control mayor de los dispositivos IoT domotizados o para integrar otro tipo de acciones como reproducción de audio, tareas con servicios externos a través de APIs etc. [19] [20]

La arquitectura de Mycroft AI se divide en cuatro módulos: [21]

- Skills. Agrupa las habilidades y capacidades que puede realizar el asistente como se ha visto anteriormente.
- Voice. Se encarga del reconocimiento de voz, grabación de audio y transcripción del mismo a texto. También incluye la generación de audio a partir de texto.
- Enclosure. Gestiona lo relacionado con el equipo en que se ejecuta el asistente como la interfaz o los componentes necesarios para grabación y reproducción de audio.
- Services. Se encarga de la comunicación del asistente tanto interna como externa y de la coordinación de trabajado de los demás módulos.

***Figura 2**. Mycroft AI*

2.4 Rhasspy

Rhasspy es una herramienta o conjunto de servicios open source, desarrollada principalmente por Michael Hansen, para facilitar la creación de asistentes de voz personalizados. Compatible con diversos idiomas, está pensada para usarse en conjunto con plataformas domóticas de monitorización como Home Assistant o OpenHAB pudiendo controlar y manejar dispositivos inteligentes por medio de comandos de voz.

A diferencia de soluciones comerciales como Alexa o Google Home, Rhasspy está pensado y optimizado para funcionar sin conexión a la red garantizando en todo momento la privacidad de los datos y la información de los usuarios y eliminando la dependencia modelos o fabricantes externos.

Además de ser una tecnología de código abierto, está diseñada con una arquitectura modular que permite adaptar el asistente y sus componentes a las necesidades del usuario y las características del entorno en todo momento. La integración y configuración de componentes o de los comandos de voz está facilitada a través de modelos predefinidos y empleando lenguaje natural para especificar frases o parámetros.

Rhasspy incluye una interfaz web sencilla para realizar todas las configuraciones necesarias, así como para realizar pruebas y entrenamiento del asistente. Proporciona también una API Rest y WebSockets para poder interactuar con el asistente e integrarlo junto con otros sistemas o dispositivos. [22]

***Figura 3**. Rhasspy*

2.4.1 Servicios

Rhasspy ofrece una serie de servicios fácilmente configurables a través de una barra de menús en su interfaz web. A continuación se detallan cada uno de ellos junto con sus posibles motores de implementación.

2.4.1.1 Internal vs External MQTT

Si se desea usar o interactuar con Rhasspy mediante MQTT, éste ofrece dos modos de configuración, Internal o External, en función de cómo se quiera gestionar la comunicación entre Rhasspy y los dispositivos o componentes.

MQTT es un protocolo de comunicación ligero, estandarizado y de bajo consumo, por lo que es ampliamente utilizado en entorno de domótica o *IoT*. Se basa en la publicación y suscripción de los elementos a topics jerárquicos, gestionado por un bróker, para enviar y recibir mensajes de forma rápida entre sí. Se puede emplear tanto para control como ejecución de acciones en dispositivos domóticos. [23]

Internal MQTT: Rhasspy lanzará internamente el bróker Mosquitto. Todos los servicios de Rhasspy se conectarán automáticamente a ese bróker local a través del cual enviarán y recibirán mensajes. Adecuado para despliegues simples o sin infraestructura MQTT.

External MQTT: Permite conectar los servicios de Rhasspy a un bróker externo ya existente del que se disponga. Además de con Rhasspy, este bróker se puede compartir con otros elementos o herramientas como Home Assistant al mismo tiempo. En este modo, la transmisión de audio del micrófono por MQTT puede saturar el bróker por lo que se recomienda hacerlo por UDP.

2.4.1.2 Audio Recording

Este servicio es el encargado de la captura del audio que se usará como entrada del asistente de voz. Es un servicio flexible que permite múltiples formas de audio y soporta varios métodos de captura, permitiendo adaptarse adecuadamente al entorno y sus necesidades. Por ejemplo, acepta desde audio proveniente de un micrófono conectado hasta transmisiones de audio remotas en vivo.

Para usar este servicio Rhasspy ofrece algunas herramientas de manera predefinida ya instaladas, las cuales se comentan resumidamente a continuación:

PyAudio: Es la opción predeterminada y recomendada de entrada de audio de Rhasspy. Utiliza la biblioteca PyAudio para acceder a los dispositivos de entrada y captura de audio. Además, es compatible tanto con ALSA (controlador directo del hardware de audio) como PulseAudio (servidor de control y gestión de audio)

Arecord: Esta opción hace referencia al propio comando de Linux *arecord*, el cual se emplea directamente para capturar el audio de dispositivos compatibles con ALSA.

Local Command: Permite recibir fragmentos de audio en crudo, en formato WAV, a través de peticiones HTTP que se envíen desde otros dispositivos. Esta opción resulta útil para integraciones remotas o distribuidas donde se utilicen varios sistemas o dispositivos.

Hermes MQTT: Se reciben los paquetes de audio a través de MQTT, utilizando el protocolo Hermes para ello. La recepción de la información se realiza en un topic MQTT específico.

En común para todas estas herramientas, el servicio Audio Recording trabaja únicamente con audio en formato de archivo WAV y, más concretamente, audio mono, codificado a 16-bit y a una frecuencia de 16kHz. [24] [25]

2.4.1.3 Wake Word Detection

Se trata de uno de los componentes clave de la arquitectura de un asistente de voz. Este servicio de detección de palabra permite activar vía voz el sistema del asistente a través de una frase o palabra previamente especificada por el usuario. Concretamente, el servicio escucha constantemente en segundo plano fragmentos WAV de audio, capturados en la entrada del asistente, en busca de la palabra de activación. Hasta que no se detecta la palabra clave el asistente permanece inactivo.

Rhasspy ofrece en su instalación varios motores de integración para ejecutar la detección de palabras de activación cada uno con un cierto grado de configuración. A continuación, se indican brevemente las características principales de cada uno:

Rhasspy Raven: Motor propio del asistente que basa el reconocimiento del *wake word* en tres muestras o plantillas de audio grabadas o proporcionadas por el propio usuario, lo que le aporta un grado de personalización al asistente.

La grabación de los fragmentos puede realizarse directamente desde al interfaz web del asistente. Con estos tres archivos WAV, Rhasspy puede entrenarse para detectar la palabra indicada fácilmente sin necesidad de conexión a la red.

Porcupine: Desarrollado por Picovoice, es un motor muy eficiente y preciso que ofrece el mejor rendimiento de entre los motores disponibles, siendo el recomendado por el propio asistente. Permite crear también *wake words* personalizadas por el usuario, a través de la consola de Picovoice, además de las que se encuentran ya disponibles para descargar desde su repositorio oficial.

Aunque permite la detección de palabras de manera *offline* y soportando múltiples plataformas, su problema principal es que las *wake words* personalizadas tienen sólo un periodo de vida de 30 días, después del cual habrá que renovarlas manualmente.

Snowboy: Se trata de un motor popular dentro del mundo de los asistentes de voz pero que dejó de tener soporte y desarrollo desde el año 2020. Tiene capacidad de crear palabras de activación personalizadas, aunque ya no es posible generar nuevos modelos. Funciona sin conexión y, además de un rendimiento notable, es capaz de reconocer varias palabras de activación de manera simultánea.

Mycroft Precise: Es el motor de código abierto que utiliza Mycroft AI para su propio asistente. Permite entrenar modelos de reconocimiento personalizados, pero el proceso de entrenamiento es más complejo y además requiere de un mayor número de muestras o datos de audio. Pese a ello, este proceso se puede realizar totalmente *offline* y con un cierto grado de personalización o caracterización del reconocimiento.

Pocketsphinx: Motor de código abierto desarrollado en la Universidad Carnegie Mellon. Es el motor de activación más flexible de todos, permitiendo configurar cualquier frase como *wake word*, pero no permite personalizar el motor entrenándolo con las frases de reconocimiento.

Además de esta limitación, es el motor que ofrece un peor rendimiento en cuanto a la relación entre falsos positivos y falsos negativos.

Local Command: A través de esta opción Rhasspy permite que se active o despierte el asistente mediante la recepción de un comando o petición HTTP. Aunque ofrece versatilidad a la hora de realizar pruebas y test de funcionamiento o integraciones con otros sistemas o herramientas, esta opción no realiza detección de palabras de activación per se.

Hermes MQTT: Al igual que el caso anterior, no es un motor de detección de *wake word* en sí, sino una integración que permite a Rhasspy recibir eventos de activación del asistente mediante mensajes enviados a través del protocolo MQTT a un topic específico.

A través de la interfaz web, en todos los motores de detección se dispone de parámetros de configuración para ajustar los umbrales de detección. [26] [27]

2.4.1.4 Speech-to-Text

Se trata del motor de reconocimiento de voz y transcripción de audio a texto. A través de este servicio, Rhasspy convierte los comandos de voz indicados por el usuario a texto y, posteriormente, los transforma a eventos en formato JSON para que puedan ser procesados e interpretados por otros servicios y componentes del sistema del asistente.

Este proceso se ejecuta automáticamente utilizando el audio grabado o transmitido desde el servicio *Audio Recording*. Es uno de los componentes fundamentales de un asistente de voz dada la importancia de reconocer y capturar las órdenes y comandos por voz de los usuarios.

Mediante este servicio Rhasspy proporciona flexibilidad y modularidad para configurar y definir el reconocimiento de voz ajustándose a las necesidades de cada proyecto. De manera predeterminada, proporciona varios motores de STT para utilizar en la implementación de un asistente con distintas capacidades entre ellos. Se describen a continuación: [28]

Pocketsphinx: Este motor open source, desarrollado por la Universidad Carnegie Mellon, realiza el reconocimiento de audio empleando diccionarios y modelos acústicos para cada idioma y de manera completamente *offline*. Estos dos componentes pueden customizarse para perfilar el entrenamiento del modelo o para capacitar al motor para reconocer nuevas lenguas.

Aunque está orientado a dispositivos con pocos recursos o poca capacidad de procesamiento, recomienda su ejecución en un servidor externo e implementarlo a través de conexión remota HTTP para evitar problemas de latencia o demora en el procesamiento.

Kaldi: Se trata de un reconocedor de voz avanzado que se ejecuta sin conexión a internet, de manera totalmente local.

Como herramientas de reconocimiento emplea modelos de lenguaje generados a partir de frases o gramáticas definidas por el usuario. El modelo de lenguaje puede configurarse en modo ARPA, más flexible para el reconocimiento general de cualquier frase, o text_fst, adecuado para el reconocimiento en exclusiva de frases o palabras predefinidas.

Kaldi requiere más capacidad de cómputo y recursos que otros motores de los disponibles, pero a cambio de mayor precisión.

Mozilla DeepSpeech: Emplea la versión 0.9 de este motor de STT. DeepSpeech está basado en el aprendizaje profundo, utilizando modelos de lenguaje grandes y con capacidad para reconocimiento de lenguaje natural abierto sea cual sea y de manera totalmente *offline*.

Debido a esto, requiere de un hardware de cierta potencia para ejecutarse correctamente y para entrenarse ya que emplea archivos de modelos de más de 1GB de la librería *deepSpeech*. El modo de reconocimiento *open transcription* ofrece flexibilidad para el reconocimiento de palabras a cambio de una ejecución más lenta y un mayor consumo de recursos.

Vosk: Es un motor de STT basado en Kaldi y diseñado para ser eficiente, rápido y con soporte para muchos idiomas distintos. Destaca por su baja latencia incluso ejecutándose en dispositivos modestos con pocos recursos. Como contrapartida, precisa que el audio a transcribir sea limpio, de calidad y sin ruido para lograr una tasa alta de precisión en el reconocimiento. [29]

Inicialmente no se encontraba disponible de forma nativa en Rhasspy, pero fue incorporado en actualizaciones posteriores.

Remote HTTP: De la misma manera que para otros servicios, permite delegar las funciones de reconocimiento de voz a un servidor externo con conexión mediante consultas HTTP. Resulta

útil para integrar motores de reconocimiento STT no integrados nativamente en Rhasspy, proporcionando gran versatilidad a la herramienta de asistente de voz.

El servicio se encarga de transmitir los archivos de audio al servidor externo y de recibir la transcripción en formato JSON de manera transparente al usuario.

Local Command: Similar al servidor HTTP remoto, permite realizar las funciones de transcripción a un comando, programa o script local. A diferencia del servidor remoto, requiere que el comando o programa se ejecute también en la misma máquina que el asistente, de manera que puede verse limitado por la capacidad de recursos del dispositivo.

Hermes MQTT: Como en otros servicios, no es un motor de STT como tal, sino un protocolo de comunicación que, a través de topics MQTT definidos, servirá de intercambiador de información entre Rhasspy y el sistema externo que realizará la transcripción del audio.

En función de la configuración del servicio MQTT, interno o externo, el programa que realizará la transcripción podrá ejecutase en otro dispositivo o en local. [30]

2.4.1.5 Intent Recognition

Este es el otro componente clave en el desarrollo de un asistente de voz. El servicio Intent Recognition de Rhasspy es el encargado de realizar la interpretación y el reconocimiento de comandos de voz y órdenes dirigidas a los dispositivos domóticos del entorno. Cada una de las acciones a realizar sobre un dispositivo se denominan Intents, de ahí el nombre del servicio.

El reconocimiento se realiza sobre la transcripción de audio a texto realizada por el servicio STT y en base a una serie de frases o sentencias previamente definidos. Para ello, debe definirse el archivo sentences.ini donde se indicarán los Intents a reconocer y, dentro de cada uno, las distintas sentencias asociadas que el servicio buscará en las transcripciones. La definición de sentencias e Intents a través de un archivo aporta gran modularidad y escalabilidad permitiendo modificarlo o ampliarlo fácilmente.

Rhasspy dispone de ciertos sistemas de reconocimiento de Intents disponibles de forma nativa, todos ellos operando sin conexión. Se resumen sus características a continuación: [31] [32]

Fsticuffs: Sistema de reconocimiento de coincidencia aproximada diseñado para reconocer únicamente las sentencias con las que haya sido entrenado. Fsticuffs dispone de opciones de ajuste para ignorar aquellas palabras no definidas en el sentences.ini, o para habilitar el cálculo aproximado con *fuzzy*, permitiendo un cierto margen de error en la interpretación y reconocimiento de las frases y sentencias.

Puede manejar fácilmente cientos de sentencias definidas, pero, a la hora del reconocimiento, el orden de las palabras debe ser exacto a como estén definidas en el archivo sentences.ini

Fuzzywuzzy: Está basado en el reconocimiento de sentencias por coincidencia difusa, de manera que, para cada oración, busca el Intent con mayor coincidencia de entre las sentencias proporcionadas. Funciona mejor con un número pequeño de sentencias

Aunque se puede escoger este sistema por separado, actualmente no se suele utilizar en la práctica al estar ya integrado en Fsticuffs para la detección aproximada de Intents en las sentencias.

Rasa NLU: Este sistema de reconocimiento funciona de manera remota a través del servidor externo Rasa NLU, el cual debe instalarse de manera local al que Rhasspy tenga acceso.

Tiene un funcionamiento óptimo con grandes números de sentencias, del orden de cientos o miles, y también es capaz de trabajar con palabras o sentencias distintas del dataset de entrenamiento, analizándolas en tiempo real.

Es posible personalizar o modificar la configuración predeterminada de entrenamiento ajustando parámetros o modificando el idioma.

Remote HTTP: Igual que hace Rasa NLU, este sistema permite realizar el reconocimiento de Intents de manera remota empleando cualquier herramienta de reconocimiento que se desee a través de comunicación mediante consultas y peticiones HTTP. El servidor remoto debe asegurar el envío de la respuesta a Rhasspy en el mismo formato JSON utilizado por el resto de sistemas.

Local Command: De igual forma que en otros servicios, habilita a Rhasspy a realizas la labor de reconocimiento de Intents a través de un programa o script personalizado ejecutado en local. Este servicio recibirá como entrada la transcripción realizada por STT y devolverá la respuesta en formato JSON.

Hermes MQTT: Protocolo de comunicación que se utiliza para delegar en otros servidores, internos o externos, el reconocimiento de Intents de las sentencias. Mediante distintos topics MQTT, por un lado, se publican las sentencias a reconocer por el servidor y por otro se leen las respuestas en formato JSON transmitidas por el mismo.

El formato JSON que se menciona en este servicio es común para todos los sistemas. A través de él se puede transmitir información relevante sobre el comando, además del propio intent reconocido. Esto es posible mediante la definición de variables en las sentencias definidas para cada Intent en el archivo sentences.ini como pueden ser *acción*, *ubicación*, *dispositivo* etc. La definición y elección de estas variables es totalmente libre.

2.4.1.6 Intent Handling

Este servicio de Intent Handling es uno de los que proporciona mayor versatilidad y flexibilidad en cuanto a las implementaciones o automatizaciones que se pueden realizar y llevar a cabo a través de un asistente de voz. Es el nexo entre la interpretación de órdenes en los comandos de voz por parte de Intent Recognition y la ejecución propia de las acciones asociadas a ese Intent.

A través de este servicio Rhasspy tiene la capacidad para crear y ejecutar automatizaciones sobre los dispositivos domotizados del hogar, convirtiéndolo en un hogar inteligente como tal. [33]

La ejecución de este servicio puede realizarse mediante alguno de los tres modos predefinidos:

Home Assistant: Se trata de una integración nativa y rápida de usar para enviar las acciones a ejecutar cuando se está usando Home Assistant como plataforma de monitorización de los dispositivos del hogar.

Requiere de poca configuración tanto en Rhasspy como en Home Asssistant a través de ficheros YAML o interfaz gráfica. El envío de las acciones, denominadas eventos en este caso, se realiza a la API Rest de la implementación de Home Assistant. Ofrece soporte para realimentación auditiva a través de TTS. [34]

Remote HTTP: El funcionamiento de esto modo es similar al de Home Assistant, con la diferencia de que permite enviar las acciones a través de peticiones POST a cualquier servidor externo.

Esta opción es idónea para integraciones de hogar digital que dispongan de arquitecturas distribuidas con sistemas ejecutados en distintos dispositivos y entornos.

Local Command: En este modo Rhasspy permite la ejecución de scripts o comandos, a los que se les pueden añadir argumentos, para ejecutar las acciones a llevar a cabo en función del intent reconocido.

Esta opción es la más versátil de todas pues admite cualquier implementación o lógica de ejecución que se puede realizar a través de un script. El programa o script en cuestión recibirá por la entrada estándar un JSON con la información del intent reconocido por Intent Recognition y devolverá una respuesta, si se ha configurado, por medio de la salida estándar.

Con este modo se puede dotar de un máximo control sobre los dispositivos domóticos a través de lógica específica y además no precisa de conexión a la red al ejecutarse sobre la máquina en la que se encuentra Rhasspy.

2.4.1.7 Text-to-Speech

Cuando un asistente realiza o ejecuta la acción o el comando que se le ha indicado por voz, aunque no es imprescindible, una de las respuesta más habituales o naturales es generar una respuesta de voz al usuario como realimentación o confirmación de la acción realizada. Este servicio de Text-to-Speech es el encargado de llevar a cabo esta tarea permitiendo convertir líneas de texto en audio de manera sencilla y transparente facilitando una interacción con el usuario de forma más natural.

El audio generado puede reproducirse por cualquier salida de audio disponible o conectada al asistente y, además, el servicio TTS puede ser invocado externamente al asistente a través de la API Rest que expone Rhasspy, pudiendo usarse este servicio de manera más flexible o en otras automatizaciones.

Por estos motivos, se considera al TTS la otra pieza clave o fundamental de un asistente de voz. En esta categoría, Rhasspy ofrece de manera nativa, y casi totalmente *offline*, una gran variedad de motores TTS a utilizar. A continuación, se detallan los aspectos más importantes de cada uno de ellos: [35], [36]

Espeak: Motor de TTS muy ligero y rápido óptimo para la lectura de oraciones no demasiado extensas. De los motores ofrecidos de serie por Rhasspy es el que soporta una mayor variedad de idiomas distintos.

Por el contrario, dispone de poco margen de configuración por el usuario y ofrece una única voz de reproducción disponible con un sonido bastante robótico.

PicoTTS: Similar a Espeak, se trata de un motor de TTS rápido y muy ligero, apto para dispositivos con bajos recursos. Este, por su parte, emplea los *picotts* (unidades mínimas de sonido) de la marca SVOX para la conversión de texto a audio de manera más fluida proporcionando una voz más natural. Tiene mejor calidad que Espeak pero está más restringido en cuanto al número de idiomas, soportando principalmente los europeos.

NanoTTS: Este servicio, como sun ombre deja entender, supone una versión mejorada de PicoTTS y la tecnología de SVOX empleada en el mismo. Ofrece soporte para el mismo número de idiomas, pero con mejoras en los apartados de calidad y rendimiento para los mismos recursos. [37]

MaryTTS: Se trata de un motor para la conversión de texto a audio basado en Java. En su versión habitual, utiliza un servidor web remoto, MaryTTS, para realizar la conversión. Sin embargo, existe disponible una imagen Docker de este servidor que se puede descargar para ejecutar de manera completamente local el asistente.

Esta imagen soporta un amplio número de idiomas y ofrece de base muchas voces distintas entre las que elegir. Para ahorrar recursos y mejorar el tiempo de ejecución, puede restringirse la imagen para usar una sola voz preseleccionándola vía comando.

Google Wavenet: Este motor de TTS utiliza para la generación de audio a partir de texto el sistema desarrollado por Google de mismo nombre que ofrece una calidad de voz bastante alta y con una voz muy natural.

Pese a que Rhasspy es capaz y puede guardar en caché frases previas con las que se haya entrenado para minimizar las consultas externas, este es un motor que se ejecuta exclusivamente en la nube de Google, por lo que precisará de conexión a internet y una cuenta de Google para poder usarse.

OpenTTS: Más que un motor en sí, OpenTTS es un framwork que expone y permite usar múltiples sistemas de conversión TTS a través de conexión HTTP. Dentro de este entorno están disponibles voces de motores como Espeak, NanoTTS, MozillaTTS, etc.

Al igual que MaryTTS, existe una imagen Docker disponible para su descarga para poder ejecutar el framwork en local sin conexión a internet. Es un sistema bastante flexible y escalable con una precisión y calidad de audio dependiente de la voz que se elija utilizar.

Larynx: Este motor TTS emplea para la conversión texto-voz, a diferencia del resto, modelos *onnx* de redes neuronales preentrenados, como por ejemplo GlowTTS. También emplea *gruut* para la conversión de texto a fonemas a través de la tokenización de palabras.

Tiene una calidad de voz muy alta en comparación con el resto de motores, especialmente combinada con el uso de vocoders, y un catálogo de voces bastante amplio. Sin embargo, requiere el uso de hardware potente para obtener resultados de voces con alta calidad.

De cara a ejecutarlo de manera limitada sobre una Raspberry Pi, supone una gran mejora de rendimiento utilizar un sistema operativo de 64 bits frente a uno de 32.

Remote HTTP: Al igual que para otros servicios de Rhasspy, permite conectar e implementar cualquier motor TTS externo que tenga expuesto un endpoint HTTP. En la consulta enviada por Rhasspy se incluirá el texto a transformar a audio en formato JSON. Por su parte el motor devolverá en la petición correspondiente un fichero de tipo WAV como respuesta con el audio

Local Command: Permite a Rhasspy ejecutar cualquier tipo de programa o script local para realizar las funciones de TTS. A este servicio, al igual que en Remote HTTP, se le pasará como parámetro un archivo JSON con el contenido o el texto a transformar a audio. A su vez, el programa o script deberá devolver el audio en un archivo con formato WAV. Estos dos últimos modelos para el servicio TTS aumentan considerablemente el abanico de oportunidades para realizar la generación de audio.

Hermes MQTT: Protocolo de comunicación que se emplea para transmitir información, en este caso el JSON con el texto a transformar y el archivo WAV con el audio generado, entre Rhasspy y un servidor interno o externo que realice las funciones de TTS. El intercambio de información se realiza mediante la publicación y suscripción a topics MQTT

2.4.1.8 Audio Playing

Este servicio es el encargado de realizar la reproducción de audio en Rhasspy. La fuente de datos habitualmente suele ser el audio generado por el servicio TTS, pero también permite reproducir archivos de audio indicados o adjuntados manualmente al servicio. Además, para algunos motores de Wake Word, este servicio se encarga también de ofrecer realimentaciones sonoras para indicar eventos como la activación del asistente tras detectar la wake word o si el Intent Recognition ha podido detectar o no alguno de los intents definidos en el audio grabado.

Este servicio resulta fundamental al ofrecer la capacidad de mandar mensajes, respuestas o dar algún tipo de realimentación al usuario final del asistente en el hogar. Permite que exista una cierta interacción entre usuario y asistente de manera natural y multimodal.

A nivel de configuración o personalización, no dispone de muchas opciones al respecto más allá de las genéricas de control de volumen y selección de dispositivo de salida. Dada la simplicidad del servicio, Rhasspy apenas ofrece unas cuantas opciones entre las que elegir: [38] [39]

Aplay: Se trata de un motor muy simple y sencillo de utilizar. Se basa únicamente en utilizar el comando de Linux *aplay* para reproducir archivos de audio en formato WAV directamente desde el dispositivo. Este es compatible tanto con ALSA como con PulseAudio y, por ende, con dispositivos como una Raspberry Pi.

El dispositivo a emplear para reproducir el audio puede seleccionarse a través de argumentos del comando *aplay*, así como ajustar el volumen de salida del mismo.

Remote HTTP: Envía los datos de audio en formato WAV mediante una petición POST a un servicio externo con un endpoint HTTP. Esta opción permite utilizar servicios o dispositivos remotos cómodamente para la reproducción de audio. [40]

Local Command: Capacita a Rhasspy para poder ejecutar otro comando o programa, en lugar de *aplay* para la reproducción de audio en un dispositivo. Al igual que en la opción anterior, el audio debe adjuntarse al comando como un argumento en formato WAV.

Esta opción dota a Rhasspy de gran modularidad y flexibilidad para usar cualquier reproductor de audio compatible.

Hermes MQTT: Protocolo de comunicación que se utiliza para transmitir los archivos de audio a ser reproducido, en formato WAV, a servidores o dispositivos que pueden ser tanto internos como externos en función de la configuración del bróker MQTT.

El envío de los archivos de audio se realiza mediante publicación/subscripción de topics MQTT.

2.4.1.9 Dialogue Manager

Este servicio es el encargado de gestionar las sesiones de diálogo que tienen lugar en el asistente. Estas sesiones son iniciadas tras la detección de una palabra de activación o Wake Word o bien a través de una orden directa indicada mediante la interfaz web de Rhasspy. La función principal que realiza es ordenar y coordinar el flujo de procesamiento y la conversación que tiene lugar entre usuario y asistente de voz.

Se encargará de asegurar que las distintas etapas o servicios por los que pasa una orden o interacción del usuario se ejecuten correctamente. También asegurará o comprobará que la información intercambiada entre las entradas y salidas de los distintos servicios sea correcta y esté en el formato adecuado o necesario. [38]

Un flujo típico del funcionamiento habitual del asistente sería:

- Se detecta una palabra de activación
- El asistente graba el audio a continuación y lo envía al servicio de STT
- El texto transcrito es analizado por el Intent Recognition para identificar y reconocer la orden
- El Intent Handling recibe el Intent reconocido y ejecuta las acciones pertinentes según su lógica. Esta lógica puede incluir o no una respuesta de audio.
- TTS recibe el texto a convertir en audio, lo genera y el asistente lo reproduce
- En función de la lógica, el Dialogue Manager decide si la interacción ha terminado o debe permanecer activa aún.

El asistente de voz dispone de dos modos de funcionamiento para el Dialogue Management, estos son los siguientes: [41]

Rhasspy: El asistente en este modo utilizará el propio bróker MQTT interno para entablar y organizar la comunicación entre los distintos servicios que se ejecuten durante un diálogo entre asistente y usuario. Esta ejecución es recomendada cuando todos los servicios y ejecuciones se

realizan en un mismo dispositivo de forma interna, garantizando el aislamiento de otros sistemas

Hermes MQTT: El funcionamiento y ejecución del servicio es prácticamente igual al del modo anterior. La diferencia principal es que el asistente se conectará a un bróker MQTT externo que puede estar localizado en otra máquina. Este modo permite gestionar y mantener el diálogo entre servicios o ejecuciones que se realicen en servidores remotos o en dispositivos distintos de una arquitectura distribuida.

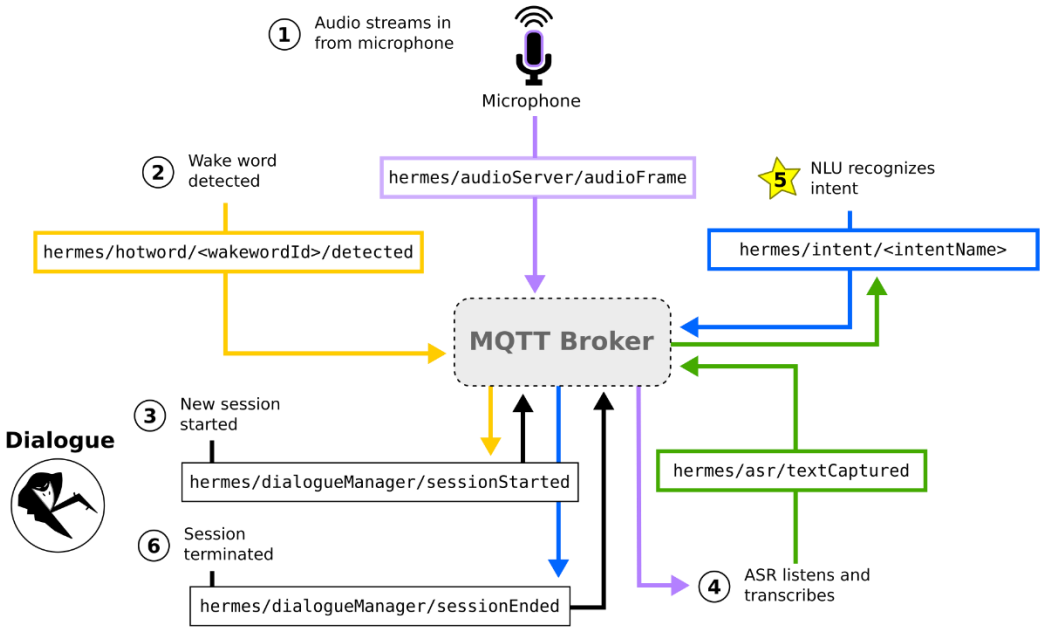

Figura 4. *Flujo de diálogo de una sesión iniciada*

2.5 Whisper AI

Whisper AI es un sistema de reconocimiento de voz automático, o ASR, de código abierto desarrollado por OpenAI en 2022. Este sistema está diseñado para realizar transcripciones multilingües y traducciones de voz a texto con una alta precisión.

El principal fuerte frente a otros sistemas de STT es que Whisper AI ha sido entrenado con más de 680.000 horas de datos, recopilados de internet, en más de 90 idiomas. Esto hace que tenga muy buena precisión a la hora de reconocer audio de distintos idiomas y acentos. Además, todo el proceso (identificación de idioma, transcripción y/o traducción al inglés) se realiza en un único modelo o sistema, simplificando su uso y funcionamiento. [42]

A nivel de arquitectura, el funcionamiento de Whisper AI se basa en un transformador codificador-descodificador (encoder-decoder) de extremo a extremo. El audio de entrada se segmenta en trozos de 30 segundos y estos se transforman en un espectrograma Mel para ser codificados. Posteriormente, el descodificador emplea tokens especiales para indicar al modelo las tareas a realizar en cada momento hasta obtener la transcripción final. [43]

En cuanto a rendimiento, la cantidad de datos de entrenamiento le concede una capacidad casi humana para reconocer y manejar voces rápidas, dialectos o vocabulario técnico o específico.

Además, también le aporta cierta tolerancia al ruido, siendo capaz de reconocer voces incluso con ciertos niveles de interferencia. [44]

Para usar este sistema STT, se dispone de la API Whisper, disponible en Python, a través de la cual se podrán realizar transcripciones de manera sencilla y a tiempo real mediante la conexión a la nube de OpenAI. Sin embargo, también están disponibles modelos, en diferentes versiones según su rendimiento, para instalar localmente. Existe un repositorio oficial en GitHub con los modelos, información y pautas de instalación e implementación.

Pese a sus ventajas, tiene algunos inconvenientes al tratarse de un sistema STT tan potente y preciso:

- La fragmentación del audio en segmentos puede conllevar retardos o latencia si se emplea para aplicaciones en tiempo real.
- Pese a disponer de varias versiones instalables, las limitaciones a nivel de hardware pueden obligar al uso sólo de ciertos modelos, lo que puede suponer una peor precisión en el reconocimiento
- La traducción de audio está optimizada únicamente para inglés, resultando poco eficiente para otros idiomas si se precisa utilizar.

Figura 5. *Whisper AI*

2.6 Piper TTS

Piper TTS es una solución de síntesis de voz neuronal open source desarrollado principalmente por Michael Hansen con el objetivo de convertir texto en habla natura de alta calidad con baja latencia en una amplia variedad de dispositivos.

Utiliza modelos de redes neuronales avanzadas basadas en ONNX, acompañados de ficheros de configuración JSON, para sintetizar el audio, permitiendo generar una voz natural, expresiva y clara con un gran parecido a la voz humana. De serie, tiene soporte para sintetizar voz en múltiples idiomas y/o dialectos además de contar con varios modelos de voz, pudiendo seleccionar entre los disponibles en función de gustos o necesidades. Además, permite configurar algunos parámetros como el tono, velocidad o volumen haciéndolo más personalizable. [45]

Este sistema se ejecuta completamente en local sin ningún tipo de conexión necesaria a servicios en la nube garantizando la privacidad total de los datos y reduciendo considerablemente la latencia de ejecución. Además, es compatible con sistemas operativos como Linux, macOS y Windows y está especialmente optimizado para poder ejecutarse en dispositivos de bajos recursos como una Raspberry Pi, permitiendo integrarlo en entornos domésticos o dispositivos embebidos sin sacrificar la calidad. [46]

Piper TTS normalmente se ejecuta como un servicio HTTP local al cual se pueden enviar peticiones HTTP con el texto a sintetizar obteniendo como respuesta el audio generado. Existe soporte también para integraciones directas con algunas plataformas como OpenHAB o HomeAssistant. Piper ofrece para cada idioma una serie de modelos instalables a través de ficheros ONNX y JSON aunque es posible entrenar un modelo personalizado a partir de conjuntos de datos permitiendo adaptarlo al contexto necesario. [47] [48]

Por todo esto, Piper TTS se considera uno de los sintetizadores de voz más asequibles en proyectos de bajos recursos debido sobre todo a la flexibilidad que ofrece, la fácil escalabilidad, el reducido coste y la latencia casi instantánea, vital en asistentes de voz con interacción con humanos.

Figura 6. *Piper TTS*

2.7 Altavoces

Aunque el altavoz no es un elemento imprescindible de un asistente de voz, es un elemento que aporta un gran valor añadido al permitir transmitir al usuario respuestas, confirmaciones o cualquier otro tipo de información por vía auditiva generada por el sistema. De esta manera, permiten una interacción completa y bidireccional entre el usuario y el asistente, ideal para entornos domésticos y personas con ciertas dificultades o discapacidades cognitivas.

En el análisis de las distintas alternativas, los altavoces deben cumplir una serie de requisitos mínimos:

- Compatibilidad con la Raspberry Pi u otro tipo de microordenadores
- Calidad de reproducción adecuada para el espectro frecuencial de la voz digital (8kHz-10kHz) que facilite la inteligibilidad de la voz
- Consumo energético reducido
- Facilidad de integración y coste reducido.

En cuanto a las posibilidades de conectividad que ofrece una Raspberry Pi, se analizan varias opciones:

Conexión USB: Son altavoces de fácil integración ya que todos acostumbran a incluir tecnología plug-and-play que facilita su conexión e integración al sistema sin necesidad de configuración adicional. Dependiendo del modelo pueden ofrecer un rango bastante amplio de calidad de audio para la reproducción de voz con relativo bajo consumo pero sin demasiada potencia.

Conexión Jack 3.5mm: Similares a los altavoces USB, también disponen de tecnología plug-and-play habitualmente pero, en este caso, el microordenador al que se conecte debe contar con entrada o puerto para Jack de 3.5mm. Son modelos muy económicos que permiten ofrecer calidades de audio bastante buenas y adecuadas para la reproducción de voz.

Conexión Bluetooth: Este tipo de altavoces precisan que la Raspberry Pi cuente con tecnología para conectar dispositivos vía Bluetooth. Su mayor fuerte es su tamaño reducido y la capacidad de colocación en cualquier lugar del entorno dentro de un cierto rango de distancia. A la par, este también es un punto negativo, al suponer una limitación la posible latencia en la reproducción o la pérdida de información derivado de la conexión inalámbrica con el dispositivo.

Módulos HATS específicos: A nivel de compatibilidad resultan la mejor opción al ser unos altavoces desarrollados exclusivamente para usar en una Raspberry Pi. Resulta una solución de gran calidad, muy precisa y con capacidad de ampliación o mejora, pero es un dispositivo con un coste muy superior a las demás tecnologías o posibilidades.

2.8 Micrófonos

El micrófono, en el ámbito de un asistente de voz, es uno de los componentes fundamentales, al tratarse de la interfaz de interacción principal entre el usuario y el sistema a través del cual se recogerán las órdenes e indicaciones del usuario. La calidad y éxito del reconocimiento de voz dependerá en gran medida de este elemento y sus características como fidelidad, sensibilidad o direccionalidad, y lo buenas que sean las muestras de audio que recoja.

En este apartado se ha hecho un análisis de las distintas opciones de micrófonos disponibles en línea con la idea de este proyecto y atendiendo a una serie de requisitos mínimos:

- Compatibilidad directa con microordenadores como la Raspberry Pi.
- Calidad de grabación de audio en entornos domésticos.
- Bajo consumo de recursos
- Simplicidad de integración y coste limitado.

De entre las distintas opciones se destacan las siguientes:

KLIM Voice: Micrófono USB omnidireccional con base fija. Goza de tecnología plug-and-play para facilitar su instalación y uso y es compatible con sistemas Linux como Raspberry Pi OS. También ofrece un botón de mute. Entre sus características técnicas destaca una

sensibilidad de -58 ±2dB, una respuesta de frecuencia de entre 100Hz-10kHz y una relación señal-ruido superior a los 50dB. Muestrea audio a 48kHz [49]

Su diseño robusto, su coste (en torno a los 20€), la facilidad de uso y manejo al no precisar controladores adicionales de ningún tipo y la calidad aceptable para entornos interiores como el hogar lo hacen un buen candidato para proyecto como el de un asistente de voz orientado al HDA.

ReSpeaker 4-Mic Array: Se trata de un micrófono de array avanzado que incorpora cuatro micrófonos analógicos gestionados mediante códec AC108. Está diseñado específicamente para instalarse de manera acoplada a una Raspberry Pi, orientado a proyecto de inteligencia artificial como un asistente de voz. Está dotado de algoritmos embebidos como Natural Language Understanding, detección de palabra clave o detección de dirección de llegada. [50]

Aunque actúa bien en entornos ruidosos con buena precisión en el reconocimiento, su instalación y configuración son bastante más complejas, el precio es superior al de otro tipo de micrófonos o soluciones y el hecho de estar embebido en la Raspberry Pi limita su posición o colocación a la ubicación donde se instale la Raspberry Pi.

Sony PlayStation Eye: Aunque no se trata de un micrófono puramente, esta cámara diseñada para la consola PlayStation 3 incluye un array de cuatro micrófonos integrados que le permite capturar audio direccional de forma bastante precisa. Este tipo de opciones son populares en proyecto de este tipo DIY (Do It Yourself) por su relación calidad precio. [51]

El problema que dispone esta cámara es la compatibilidad con microcontroladores como la Raspberry Pi, necesitando configuración e instalación de controladores específicos para funcionar correctamente. Además, el hecho de disponer de cámara también no aporta ningún tipo de valor añadido al rango de este proyecto.

STJF Mini: Se trata de una de las opciones más básicas y asequibles del mercado. Un micrófono USB con tecnología plug-and-play, compacto y barato que está orientado a actividades de dictado o amplificación de voz de uso ocasional. Destaca por su funcionalidad y simplicidad de uso siendo reconocido por la propia Raspberry Pi Foundation. [52]

Por el contrario, es un micrófono con rango de captación muy limitada, baja sensibilidad y con una calidad promedio insuficiente para un sistema de reconocimiento de voz que precisa de un nivel de captación de audio más alto.

3. Especificaciones y restricciones de diseño

En este apartado se definen las especificaciones técnicas y las restricciones de diseño que guían el desarrollo del proyecto. Se establecen tanto los requisitos funcionales como las limitaciones derivadas de normativas, estándares de telecomunicaciones y las condiciones del entorno. Estos aspectos resultan clave para asegurar la viabilidad del proyecto y una correcta ejecución.

3.1 Especificaciones

En el diseño y desarrollo del asistente de voz virtual se siguieron las siguientes especificaciones de diseño:

- Debe poder controlar y monitorizar el estado de luces, sensores, enchufes y demás dispositivos domóticos presentes en el hogar a través de comandos de voz de manera sencilla, reconociendo las órdenes indicadas por el usuario
- El usuario debe poder definir y modificar la lista de comandos y órdenes, que el asistente reconocerá y ejecutará a través de una interfaz de usuario amigable. El asistente mapeará las órdenes recibidas con las acciones a ejecutar sobre los dispositivos monitorizados permitiendo que puedan definirse varias órdenes o expresiones para una misma acción.
- El sistema se mantendrá en estado de espera, escuchando sin ejecutar ninguna acción, hasta ser activado al detectar una palabra clave (*wake word*) a definir por el usuario, optimizando el consumo y la privacidad.
- El asistente debe poder proporcionar realimentación al usuario por medio de sonidos y respuestas de voz generadas. Las respuestas podrán ser personalizables según la orden, dispositivo o situación y el usuario podrá elegir cuándo activar o desactivar esta respuesta por voz.
- La actividad del asistente (reconocimiento de órdenes, procesado, ejecución de acciones o generación de voz) debe realizarse de forma completamente offline y toda en el mismo dispositivo.

3.2 Restricciones

A la hora de desarrollar una solución al problema propuesto se presentan una serie de restricciones descritas a continuación:

- El asistente debe poder ejecutarse y funcionar en dispositivos de bajos recursos como una Raspberry Pi para poder ser fácilmente integrados en cualquier hogar
- El asistente y los servicios o tecnologías utilizadas deben ser herramientas de código abierto, gratuitas y sin dependencia de servicios en la nube o internet, para poder ser implementado por cualquiera y preservar en todo momento la privacidad y los datos del usuario
- Los comandos de voz reconocibles por el asistente deben ser previamente definidos por el usuario
- La latencia entre la detección de un comando y la ejecución de la acción debe ser inferior a 5 segundos para proporcionar una interacción fluida y natural entre usuario y asistente.

- El asistente debe ser compatible y capaz de interactuar con OpenHAB ya que es la plataforma de monitorización utilizada en el HDA para el control de los dispositivos domotizados.

4. Descripción de la solución propuesta

A continuación se describirán los pasos seguidos en la integración y desarrollo del asistente de voz abierto y multimodal para el control y automatización de los dispositivos domóticos presentes en el HDA de la escuela.

Se comienza describiendo los elementos hardware y dispositivos que se han elegido para desarrollar el proyecto y, a continuación, las herramientas y elementos software desarrollados.

4.1 Dispositivos y Periféricos

En el HDA, actualmente, se emplea una Raspberry Pi 4 para albergar la plataforma de monitorización domótica OpenHAB y que sirve como cerebro del entorno. Dada la limitación de recursos de esta, y la necesidad de un rendimiento mínimo adecuado del asistente, se decide emplear un sistema físico externo en exclusiva sólo para el asistente.

Aunque las marcas comerciales existente de asistentes ofrecen soluciones todo en uno, integrando sistema de procesamiento, micrófono para capturar la voz y altavoz para reproducirla, dadas las características del proyecto y las restricciones de diseño del mismo se usan dispositivos y periféricos independientes e interconectados para aportar mayor modularidad y personalización.

Se describen a continuación los distintos elementos seleccionados:

4.1.1 Microordenador

Siguiendo las restricciones de diseño mencionadas en el apartado 3, el asistente debe poder ejecutarse en un dispositivo de bajos recursos para poder ser implementado fácilmente en cualquier hogar. Por ello, y para garantizar la disponibilidad de recursos suficientes funcionar correctamente, se decide emplear una Raspberry Pi 4 model B para integrar el asistente y sus periféricos y funcionar como núcleo y unidad de procesamiento del asistente.

La Raspberry Pi 4, además de ofrecer unas características técnicas suficientes para poder ejecutar el asistente correctamente, permite integrar el altavoz y micrófono necesarios para el funcionamiento del asistente a través de los puertos USB que dispone. El uso de este microordenador también garantiza el uso de tecnologías Open Source al usar sistemas operativos basados en Linux.

4.1.2 Altavoz

Para poder realizar la reproducción de respuestas, feedback o información general del asistente de voz vía audio, se incorpora un altavoz al sistema. De entre las tecnologías barajadas de conexión e implementación se ha optado por la conexión vía Jack 3.5mm ya que la Raspberry Pi 4 utilizada dispone del puerto de conexión y porque responde a diversos factores como los criterios técnicos, audio de calidad orientado a voz, económicos, relación calidad-precio adecuada, y prácticos, implementación mediante tecnología plug-and-play.

En este caso, y debido a que se disponía ya de un altavoz de estas características en el HDA, se ha optado directamente por aprovecharlo e incorporarlo al proyecto del asistente de voz. El altavoz en cuestión es el Tacens Mars Gaming MS1 2.0.

Este altavoz, además de la conexión de Jack de 3.5mm, precisa de una toma de corriente la cual realiza a través de conexión USB. Esta opción facilita la integración en el conjunto del asistente al poder conectarse directamente a uno de los puertos libres de la Raspberry Pi y no precisar de conexiones a mayores. Una vez conectadas ambas conexiones, y gracias a la tecnología plug-and-play, se configura y selecciona directamente como dispositivo predeterminado de salida de audio en la Raspberry Pi.

El Tacens MS1 es un juego de dos altavoces simétricos de tamaño compacto que reproducen audio en mono, facilitando los requisitos y la generación de audio del asistente. A nivel técnico, ofrece 10W de potencia en conjutno con 6 drives de sonido, 2 activos y 4 pasivos. Ofrece una respuesta en frecuencia de 100Hz-20kHz, apto para audio digitalizado, y que favorece la inteligibilidad del audio generado. [53]

Cuenta también con un control de volumen analógico y manual integrado que permite ajustar el nivel de salida de forma sencilla sin necesidad de acceder al sistema del asistente de voz y permite ajustar el volumen del audio según el entorno, el usuario y las necesidades.

4.1.3 Micrófono

Para llevar a cabo la captación de voz del usuario, el sistema incorpora el micrófono USB KLIM Voice. Este dispositivo supone una solución plug-and-play efectiva, funcional y asequible con unas características técnicas adecuadas para ofrecer una calidad de audio suficiente para el procesamiento del asistente de voz.

El micrófono se conecta directamente a la Raspberry Pi mediante USB y es reconocido de forma automática por el sistema operativo sin necesidad de instalación de controladores de ningún tipo gracias al soporte plug-and-play, facilitando su instalación reduciendo la complejidad técnica. Una vez conectado, se establece automáticamente como dispositivo principal de entrada de audio de la Raspberry Pi, listo para utilizarse para la grabación de órdenes del usuario.

Desde la perspectiva técnica, el KLIM Voice cuenta es un micrófono omnidireccional con un patrón de captación cardioide, permitiendo grabar la voz de los usuarios desde casi cualquier posición con buena calidad. Esta versatilidad permite que se instale en casi cualquier ubicación del hogar sin requerir una posición concreta, dotando al usuario de la capacidad de instalarlo donde prefiera.

Dado que en el entorno del HDA no se dispone de acceso directo a la interfaz gráfica de la Raspberry Pi, las pruebas de funcionamiento y calibración del micrófono se realizaron a través de la consola de comandos mediante la herramienta *arecord* para verificar la correcta captura de audio y la idoneidad de este micrófono para actuar como tal en el conjunto del asistente de voz de este proyecto.

4.2 Herramientas y Software

4.2.1 Raspberry Pi OS

Como sistema operativo de la Raspberry Pi, se ha optado por instalar Raspberry Pi OS por la facilidad de instalación y para asegurar la compatibilidad al tratarse de una distribución Linux desarrollado de manera oficial por la propia Raspberry Pi Foundation.

Para la instalación, existe una herramienta llamada Raspberry Pi Imager disponible en la página web oficial de Raspberry Pi de manera gratuita. A través de esta herramienta se formatea una tarjeta SD y se descarga e instala la imagen de Raspberry Pi OS requerida.

En primer lugar, insertar directamente, o a través de un adaptador, una tarjeta SD de memoria de al menos 8GB según la recomendación oficial [54]. En los desplegables del panel principal de Raspberry Pi Imager, seleccionar “Raspberry Pi 4” como modelo, la versión de 64-bit del sistema operativo Raspberry Pi OS y la tarjeta SD como dispositivo de instalación. Al hacer clic en “Siguiente” podremos configurar en una ventana emergente el hostname, la contraseña de acceso y habilitar SSH para conectarse a la Raspberry Pi desde otro dispositivo mediante línea de comandos. Una vez completado se descarga e instala la imagen en la tarjeta SD.

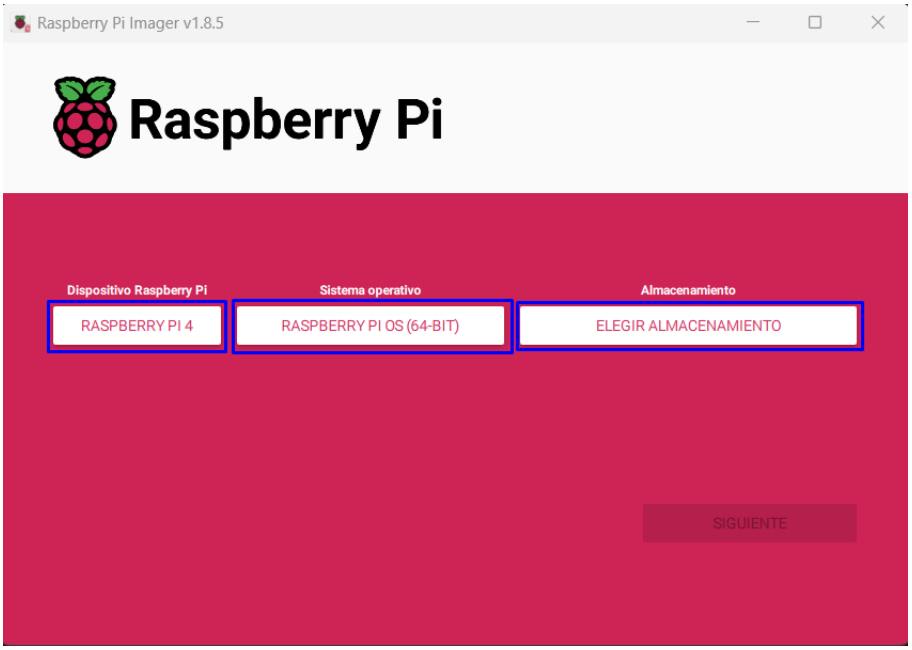

Figura 7. *Raspberry Pi Imager*

Finalmente, se introduce la tarjeta SD en la Raspberry Pi para finalizar las configuraciones básicas pertinentes de cualquier sistema operativo.

4.2.2 Rhasspy

Como herramienta para la implementación del asistente de voz en el proyecto, complementario a la monitorización de dispositivos domóticos mediante OpenHAB, se ha optado por utilizar Rhasspy. Durante el desarrollo del mismo se realizaron pruebas con esta herramienta y con Mycroft AI, siendo finalmente esta descartada al encontrarse problemas como dificultad en el reconocimiento de la palabra clave para ciertas voces o soporte único oficial del idioma inglés.

Estos problemas ya fueron detectados y documentados previamente en anteriores proyectos desarrollados en el entorno del HDA [55]

4.2.2.1 Instalación

Rhasspy es una herramienta que puede ser instalada directamente en sistemas operativos Linux de la rama Debian así como en entornos virtuales, Home Assistants o incluso Windows a través de WSL. Sin embargo, la mejor opción y más sencilla es instalar Rhasspy mediante un Docker, aislándola del resto de aplicaciones para eliminar problemas de dependencias y permitiendo instalarlo en cualquier sistema operativo que soporte Docker.

En primer lugar, por tanto, es necesario instalar Docker en la Raspberry Pi, que se puede hacer fácilmente a través de la terminal ejecutando el siguiente comando para descargarlo del repositorio oficial y ejecutarlo:

```
curl -sSL https://get.docker.com | sh
```

Además, es necesario asegurar que el usuario de la Raspberry Pi pertenece al Docker group para una ejecución correcta. Esto se realiza mediante el siguiente comando:

```
sudo usermod -a -G docker pi
```

Tras esto, se necesita reiniciar la Raspberry Pi para hacer efectivos los cambios. Finalmente sólo quedaría instalar y arrancar la imagen Docker de Rhasspy con los parámetros necesarios para adecuar la instalación a nuestras necesidades. Todo esto se realiza mediante un comando único:

```
docker run -d \
    --name rhasspy \
    --restart unless-stopped \
    --network host \
    -v "$HOME/.config/rhasspy/profiles:/profiles" \
    -v "/etc/localtime:/etc/localtime:ro" \
    --device /dev/snd:/dev/snd \
    rhasspy/rhasspy \
    --user-profiles /profiles \
    --profile es
```

Mediante *docker run -d* se lanza un nuevo contenedor que se ejecutará automáticamente en segundo plano y *--name rhasspy* le asigna un nombre al contenedor para facilitar su gestión y administración. Con *--restart unless-stopped* se configura el contenedor para reiniciarse automáticamente ante cualquier fallo excepto cuando se haya parado manualmente y *--network host* hace que el contenedor comparta la misma red que usa la Raspberry Pi.

La opción *-v "$HOME/.config/rhasspy/profiles:/profiles"* monta un volumen para guardar la configuración del usuario en un directorio accesible fuera del contenedor, enlazando la carpeta

local con el contenedor en */profiles*. Esta carpeta se indica como lugar donde leer las configuraciones del asistente a través de *--user-profiles /profiles*.

Por último, mediante *--device /dev/snd:/dev/snd* se concede acceso al contenedor a los dispositivos de sonido disponibles en el host, *rhasspy/rhasspy* indica la imagen Docker concreta a utilizar y *--profile es* especifica el idioma de la instalación.

Tras esto, la imagen de Rhasspy estará instalada y en funcionamiento. La interfaz web de control y configuración de Rhasspy está disponible en el puerto 12101 de la Raspberry Pi, accesible mediante un explorador web en la URL *http://localhost:12101*.

4.2.2.2 Configuración

En la pantalla principal de la interfaz web encontramos una barra de menús de acceso directo al panel de configuración de cada uno de los elementos y servicios que ofrece Rhasspy para confeccionar un asistente de voz. En la barra vertical lateral, clicando sobre el icono de los engranajes podremos acceder también a este menú. Comenzamos configurando cada uno de los servicios necesarios.

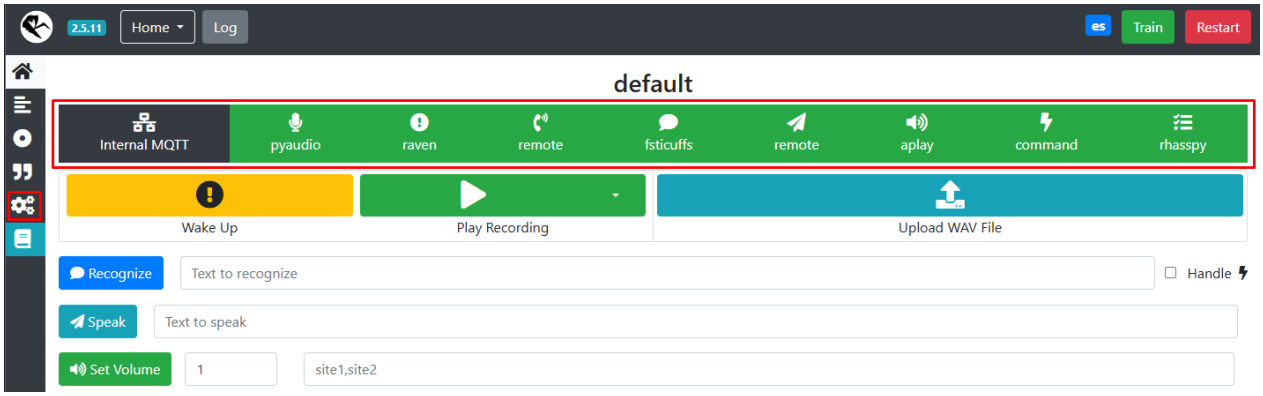

Figura 8. *Interfaz Web Rhasspy. Panel Principal*

En la solución planteada, como se observa en la imagen anterior, no se contempla la utilización de un bróker MQTT propiamente. Aunque OpenHAB para la monitorización y control de algunos dispositivos del HDA utiliza este protocolo de comunicación, la transmisión de datos de voz por esta vía resulta ineficiente al introducir retardos en el envío y recepción, incluso pudiendo saturarse la comunicación. Por simplificación y sencillez en el intercambio de datos entre servicios y herramientas se opta por utilizar el protocolo HTTP para ello como se verá más adelante.

Para el servicio de *Audio Recording*, se sigue la recomendación dada por la documentación de Rhasspy y se selecciona el motor **PyAudio** [24]. En las opciones de configuración de este servicio, seleccionar el desplegable de Devices y seleccionar el micrófono a utilizar por el asistente que previamente debe haberse conectado a la Raspberry Pi.

Si está conectado y no aparece en la lista, pulsar el botón Refresh de la derecha para recargar la detección de dispositivos. Seleccionado el equipo, se pulsa el botón Test para realizar una prueba de grabación y asegurar que Rhasspy detecta la voz a través del micrófono.

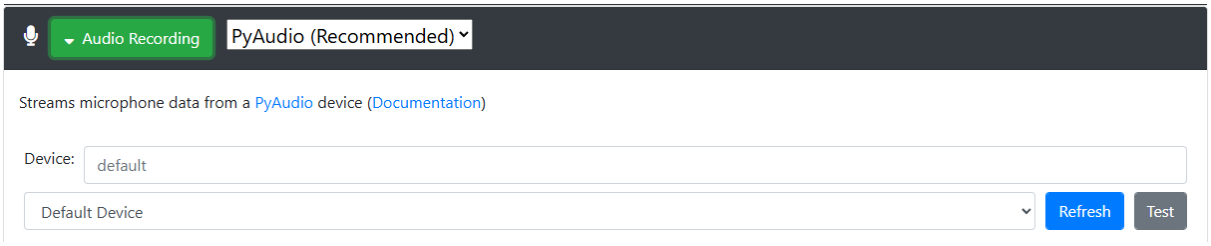

Figura 9. *Configuración Audio Recording*

Para el servicio de *Wake Word*, por la posibilidad de elegir la palabra de activación que se quiera, y la capacidad de entrenamiento de Rhasspy se selecciona el motor nativo de **Rhasspy Raven**.

En el proceso de establecer una Wake Word, escribimos primero un nombre para identificarla. Después, Raven requiere que se graben tres muestras de voz de la Wake Word para poder entrenar el motor de reconocimiento. Estas se guardarán en el directorio */profiles/es/raven* con el nombre que se les haya asignado anteriormente.

Pulsar en el botón *Record* para grabar las muestras. El el botón con el símbolo de "Play" se puede escuchar la grabación de la muestra y, en caso de que considerarla inadecuada o defectuosa, se puede regrabar pulsando el botón *Re-record*. Finalmente pulsar el botón *Save*. El propio asistente solicitará que se reinicie y entrene el modelo de reconocimiento mediante una ventana emergente, aceptarlo.

Rhasspy Raven permite guardar y entrenar el modelo para varias Wake Words, borrarlas, crear nuevas y seleccionar fácilmente la que se desea utilizar. Por defecto el parámetro Probability Threshold se deja en 0.5, suponiendo calidad suficiente del audio de entrada, para evitar falsas activaciones del asistente.

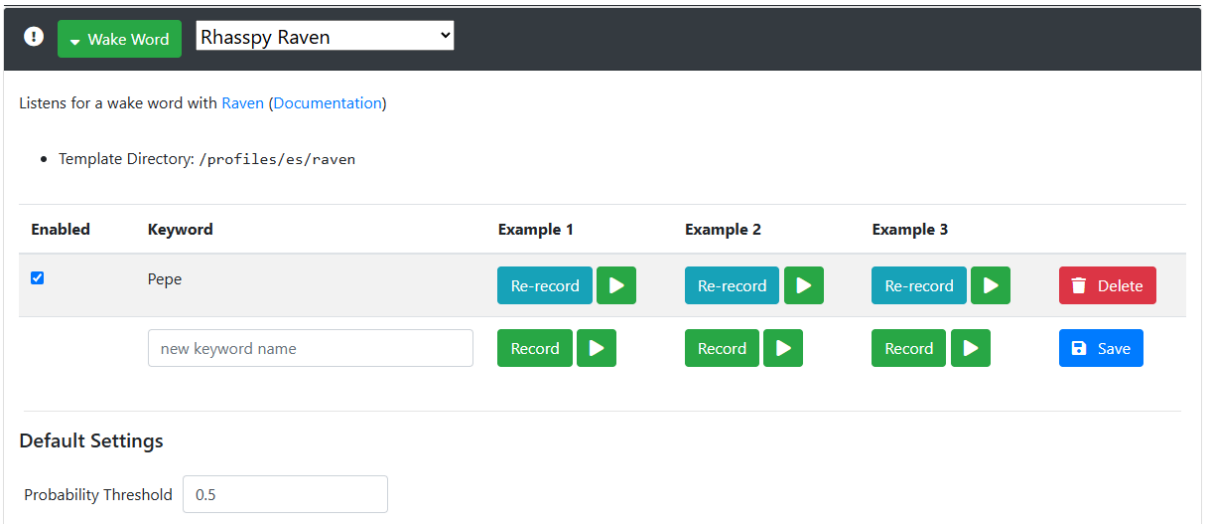

Figura X. Configuración Wake Word

Para el servicio de *Speech-to-Text*, tras realizar pruebas con los distintos motores que ofrece Rhasspy, ninguno de ellos ofrecía un rendimiento y precisión adecuado para el desarrollo de un asistente de voz acorde a los requisitos de este proyecto, siendo el motor Mozilla DeepSpeech

el que mejor rendimiento presentó. Los problemas encontrados se trataban principalmente de confusión entre palabras con pronunciación similar a la hora de transcribir los archivos de voz.

Debido a esto, se decide seleccionar la opción de **Remote HTTP** para utilizar un motor de STT externo ejecutado como un servicio en la Raspberry Pi y exponiéndolo a través de un puerto de la máquina. Se indica por tanto tan sólo la URL del endpoint HTTP donde se encuentra el servicio: http://127.0.0.1:5050/stt o http://localhost:5050/stt. Más adelante se indica la implementación de este servicio detalladamente.

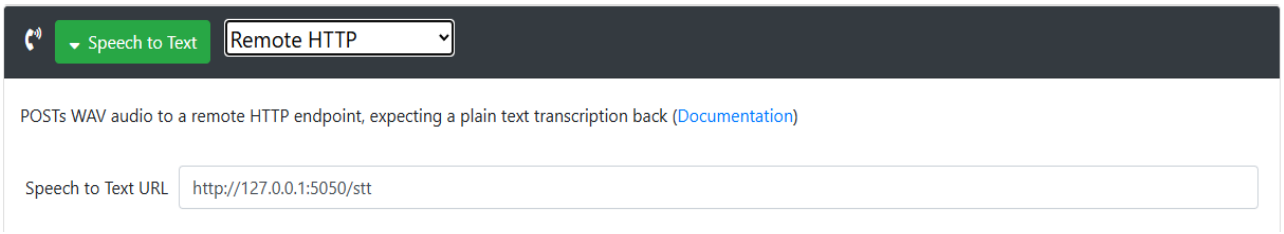

Figura 10. *Configuración Speech-to-Text*

El servicio de *Intent Recognition* se ha decidido seleccionar también siguiendo la recomendación de la documentación escogiendo el motor **Fsticuffs** por la flexibilidad ofrecida a la hora de reconocer intents [31]. En el menú de configuración, simplemente se selecciona este motor, ya que la configuración de los Intents se realiza utilizando los otros menús disponibles en la barra vertical lateral.

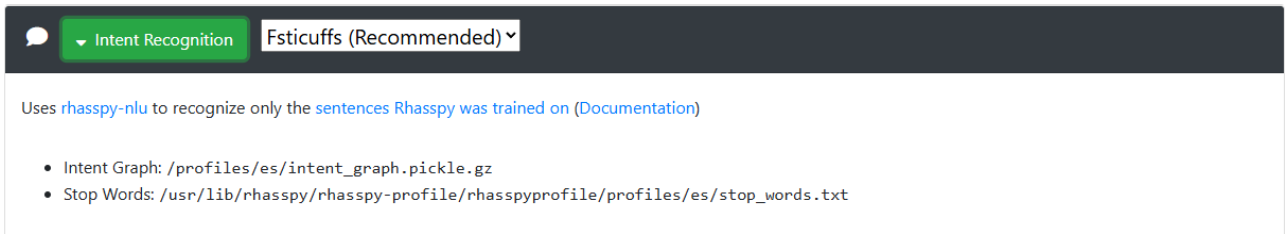

Figura 11. *Configuración Intent Recognition*

En el caso del servicio *Text-to-Speech*, al igual que en Speech-to-Text, ninguno de los motores nativos ofrecidos por Rhasspy cumple con las especificaciones y necesidades para este proyecto. En este caso, tras realizar pruebas con cada uno de los motores, los problemas en la generación de audio eran mayores que en la transcripción a texto. De forma general el tono de voz del audio resultaba robótico e incluso inentendible. Además, en muchas ocasiones se producían errores de pronunciación que hacían difícil entender qué se quería decir.

Por esto, y la necesidad de una generación de audio rápida y eficaz, se selecciona la opción de **Remote HTTP** también para poder utilizar un motor de TTS externo. Al igual que en STT, éste se ejecuta en forma de servicio en la Raspberry Pi por comodidad, exponiéndolo en un puerto de la máquina. Se indica por tanto únicamente la URL del endpoint HTTP donde se encuentra el servicio: http://127.0.0.1:4040/tts o http://localhost:4040/tts

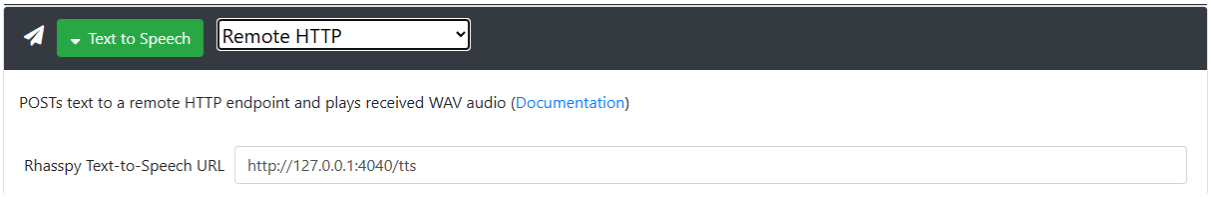

Figura 12. *Configuración Text-to-Speech*

Para el servicio de *Audio Playing* se ha optado por el servicio de **aplay**, recomendado por la documentación oficial, y por ser el único motor de reproducción de audio disponible directamente como tal [56]. De forma similar al servicio de Audio Recording, se debe seleccionar el dispositivo de reproducción de audio que se va a utilizar, que previamente se conecta a la Raspberry Pi.

Una vez conectado, si no aparece disponible en la lista del desplegable, pulsar el botón *Refresh* para recargar la detección de dispositivos de salida. Para probar la reproducción, hay que acceder de vuelta al menú principal de la interfaz web. En la última línea, pulsar el botón *Set Volume* para reproducir un sonido predefinido. En el recuadro de la derecha se puede establecer el rango de volumen de salida con valores de 0 a 1.

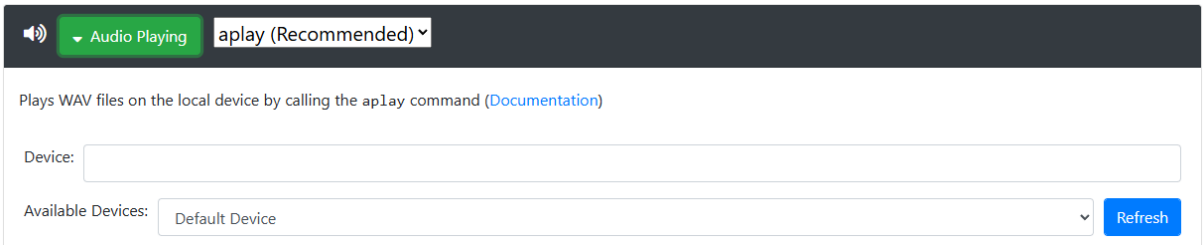

Figura 13. *Configuración Audio Playing*

El servicio *Dialogue Management* no es necesario modificarlo. La opción por defecto, **Rhasspy**, indica al asistente que se comporte como tal conectando los distintos flujos de entrada y salida de los distintos servicios.

Finalmente, para el servicio *Intent Handling*, dado que el propósito del asistente es ejecutar distintas acciones sobre los dispositivos domóticos monitorizados del HDA en función del tipo de dispositivo y lo que indique el usuario, se decide escoger la opción de **Local Command**.

Mediante esta opción, se ejecuta un programa local cuando el asistente identifica un Intent definido previamente por el usuario. Como se va a manejar una serie amplia de dispositivos, en lugar de establecer un único programa, como una petición HTTP; se indica la ubicación de un script Python desarrollado para tratar cada acción de cada dispositivo de manera individual y aislada: */profiles/es/rhasspy-files/intent_handler_v1.py*

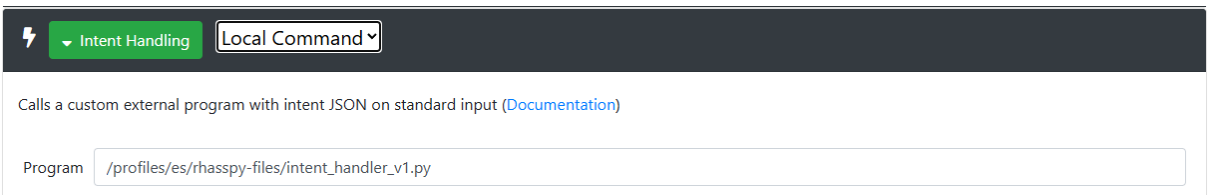

Figura 14. *Configuración Intent Handling*

4.2.2.3 Intents, Sentences y Slots

La definición y configuración de los Intents se realiza a través de los menús verticales laterales de Sentences y Slots.

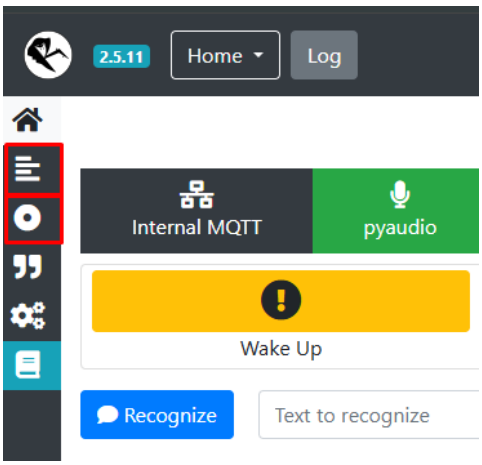

Figura 15. *Menús Laterales. Sentences y Slots*

Accediendo al primero, el de Sentences, en el panel que se muestra definimos el nombre de los distintos Intents que necesitamos reconocer así como las distintas combinaciones de frases o sentencias que están asociadas a cada Intent. Las sentencias son los elementos que el *Intent Recognition* trata de identificar en las muestras de audio transcrito.

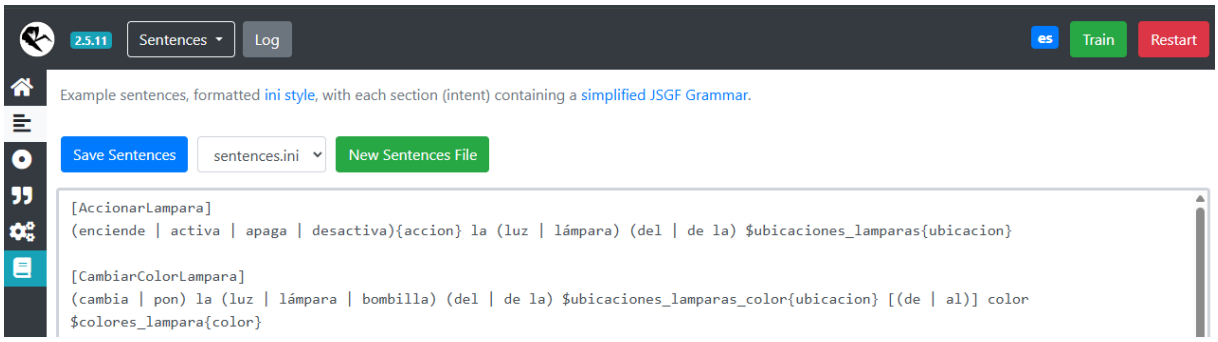

Figura 16. *Menú Sentences*

Definimos los intents siguiendo unas reglas y sintaxis:

- El nombre del intent se especifica entre corchetes, []
- Debajo se indican las sentencias asociadas al intent, una por línea
  - Entre paréntesis, (), y separadas por barra vertical, |, se indican alternativas u opciones para una palabra en esa posición
  - Entre corchetes, [], se indican palabras opcionales que pueden estar o no
  - Puede incluirse variables previamente definidas usando el dólar, $
  - Al final de cualquier palabra o variable puede añadirse una etiqueta entre llaves, {}. Esta etiqueta y la palabra asociada se incluirán en el JSON del Intent reconocido.

Los Intents se definen en función de las acciones o funcionalidades de los dispositivos monitorizados en OpenHAB que el asistente será capaz de ejecutar. En cierto modo, se correspondería con el número de items que se encuentran definidos en OpenHAB. Dada la

externalidad del asistente frente a la plataforma OpenHAB, no es necesario que el nombre indicado para los Intents corresponda con el de los items. Se usan variables definidas en los Slots para simplificar las sentencias y se definen etiquetas para variables relevantes para el procesamiento posterior de los intents.

Una vez definidos todos los Items, pulsamos el botón *Save Sentences* para guardar el archivo. Se guarda con el nombre **sentences.ini** de forma predeterminada en el directorio */.config/rhasspy/profiles/es*. Se pueden definir y guardar varios ficheros de sentencias pero no es imprescindible. Tras guardar, Rhasspy solicita, mediante ventana emergente, reentrenar el motor de reconocimiento de intents lo cual aceptamos.

En el menú Slots, definiremos los slots o variables que se usan en las sentencias de definición de los intents. Al usar una variable en una sentencia, esta podrá tener cualquiera de los valores definidor en el menú Slots. Añadir los Slots necesarios pulsando el botón *New Slots* indicando previamente el nombre del fichero. Indicados todos los valores, pulsar el botón *Save Slots* para guardar el archivo en el directorio */.config/rhasspy/profiles/es/slot*. Igual que con las Sentences, Rhasspy solicitará reentrenar el modelo, lo aceptamos.

Para este proyecto, definimos Slots principalmente para las ubicaciones del HDA que cuentan con cada tipo de dispositivo domotizado, luces, persianas, sensores de temperatura etc.

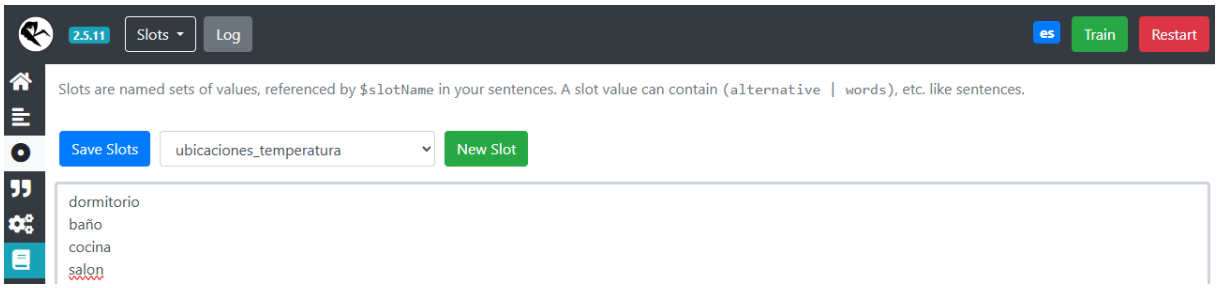

Figura 17. *Menú Slots*

En el Anexo II del documento se incluyen los archivos creados de Sentences y Slots para los elementos monitorizados en este proyecto.

4.2.3 Whisper AI

Como servicio de Speech-to-Text se ha decidido implementar el modelo Whisper AI dada la falta de precisión de las herramientas nativas de Rhasspy. La funcionalidad se ha implementado a través de un script de Python que utiliza Whisper AI para procesar el audio y transcribirlo. Posteriormente se ha integrado en Rhasspy como un servicio de la Raspberry Pi.

En primer lugar es necesario instalar Python:

```
sudo apt install python3 python3-pip -y
```

A continuación se crea un entorno virtual de Python para evitar interferencias con otros programas instalados y se activa:

```
python3 -m venv Whisper-venv
source whiper-venv/bin/activate
```

Dentro del entorno, instalamos los paquetes necesarios para usar en el script. Por un lado se instala la implementación oficial de Whisper con sus modelos, el paquete de dependencia *numpy* y *ffmpeg* como herramienta para manejar el procesado de audio. Para exponer el endpoint HTTP del servicio STT usaremos *flask*.

```
pip install openai-whisper flask numpy ffmpeg
```

Además, se instala *onnxruntime* para ejecutar los modelos de IA de Whisper AI en formato ONNX, *faster-whisper*, una implementación de Whisper AI más rápida y ligera para la transcripción de audio, y *ctranslate2*, el motor sobre el que corre faster-whisper. Adicionalmente instalamos la librería *transformers* para cargar y utilizar los modelos preentrenados ya existentes que emplea Whisper:

```
pip install onnxruntime faster-whisper ctranslate2 transformers
```

Con la instalación de *openai-whisper* se incluyen los modelos preentrenados en el directorio *openai/*. Para mejorar la precisión y lograr que la ejecución sea más rápida y eficiente en términos de recursos, se emplea *ctranslate2* para aplicar una cuantización INT8 y convertir el modelo original en uno procesable por CPU y con bajo consumo de RAM:

```
ct2-transformers-converter –model openai/whisper-base –output_dir
/home/pi/whisper-models/whisper-tiny -quantization int8 --force
```

El siguiente paso es crear el script de Python a ejecutar como servidor para procesar los archivos de audio enviados por Rhasspy, procesarlos mediante Whisper con el modelo creado, obtener la transcripción correcta del audio y devolverla a Rhasspy en formato de JSON. Se crea el script y se le modifican los permisos para hacerlo ejecutable.

```
nano Whisper_server.py                    chmode +x Whisper_server.py
```

El script escucha las peticiones que se hagan al endpoint HTTP definido en el puerto 5050 en la ruta */stt*. Rhasspy enviará a través de un POST los archivos de audio en bruto capturados para ser transcritos. El script emplea *faster-whisper* para la transcripción aplicando el modelo anteriormente convertido de *ctranslate2*. Finalmente, construye un JSON con el texto transcrito para devolverlo a Rhasspy a través de otro mensaje POST. El contenido del script se incluye en el Anexo de este documento.

Para que este script esté ejecutándose constantemente y se pueda utilizar a demanda por Rhasspy, se crea un servicio en la Raspberry Pi para mantener el script en continua ejecución escuchando para recibir archivos de audio en el endpoint. Este servicio se ha definido como *whisper-stt* y el contenido del archivo que lo define se encuentra también en el Anexo:

```
Sudo nano /lib/systemd/system/whisper-stt.service
```

Finalmente se active el servicio. En el archivo anterior se definen las variables Restart y RestartSec para que se intente relanzar el servicio siempre que se pare y a intervalos de 5 segundos.

```
sudo systemctl Daemon-reload          sudo systemctl start whisper-stt
```

Tras esto, el servicio se estará ejecutando y corriendo Whisper AI y expuesto en la URL http:/localhost:5050/stt ya configurada en Rhasspy.

```
● whisper-stt.service - Whisper STT Server
     Loaded: loaded (/lib/systemd/system/whisper-stt.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-03-30 15:26:59 CEST; 1 month 17 days ago
   Main PID: 92414 (python)
      Tasks: 16 (limit: 9421)
     Memory: 821.4M
        CPU: 2h 3min 42.773s
     CGroup: /system.slice/whisper-stt.service
             ├─92414 /home/pablo/whisper-venv/bin/python /home/pablo/whisper_server.py
             └─92421 /home/pablo/whisper-venv/bin/python /home/pablo/whisper_server.py

mar 30 15:27:05 debian python[92421]:  * Debugger PIN: 841-992-903
mar 30 15:27:33 debian python[92421]: 127.0.0.1 - - [30/Mar/2025 15:27:33] "POST /stt HTTP/1.1" 200 -
may 04 13:54:00 debian python[92421]: 127.0.0.1 - - [04/May/2025 13:54:00] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 15:31:57 debian python[92421]: 127.0.0.1 - - [04/May/2025 15:31:57] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 16:49:45 debian python[92421]: 127.0.0.1 - - [04/May/2025 16:49:45] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 17:47:51 debian python[92421]: 127.0.0.1 - - [04/May/2025 17:47:51] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 19:46:32 debian python[92421]: 127.0.0.1 - - [04/May/2025 19:46:32] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 20:23:14 debian python[92421]: 127.0.0.1 - - [04/May/2025 20:23:14] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 20:29:02 debian python[92421]: 127.0.0.1 - - [04/May/2025 20:29:02] "POST /stt?siteId=default HTTP/1.1" 200 -
may 04 20:53:31 debian python[92421]: 127.0.0.1 - - [04/May/2025 20:53:31] "POST /stt?siteId=default HTTP/1.1" 200 -
```

Figura 18. *Servicio whisper-stt corriendo*

4.2.4 Piper TTS

Al igual que para STT, las opciones nativas de Rhasspy para el servicio de Text-To-Speech no son demasiado precisas por lo que se ha optado por implementar Piper TTS como sistema de conversión de texto a voz para el asistente.

La perspectiva de implementación es la misma que para Whisper AI, utilizar Piper TTS para realizar transcripciones a través de un script de Python y crear un servicio Linux para disponer de un endpoint HTTP con el que Rhasspy pueda interactuar con él. Siguiendo el mismo procedimiento, se genera un entorno virtual (*piper-venv*) para implementar los programas necesarios evitando incompatibilidades con otros elementos.

Creado el entorno, se instala *flask* para exponer el endpoint HTTP del servicio TTS y *piper-tts*, que contiene la herramienta principal de Piper TTS y funciones para la carga de modelos y la generación de audio a través de texto. Al igual que con Whisper AI se deben descargar los modelos en español que usará Piper para la generación de audio:

```
wget https://huggingface.co/rhasspy/piper-voices/resolve/v1.0.0/es/es_ES/davefx/medium/es_ES-davefx-medium.onnx
wget https://huggingface.co/rhasspy/piper-voices/resolve/v1.0.0/es/es_ES/davefx/medium/es_ES-davefx-medium.onnx.json
mv es_ES-davefx-medium.onnx.json es_ES-davefx-medium.json
```

Se modifica el nombre del JSON de configuración para evitar problemas con el archivo del modelo en formato ONNX.

A continuación, se crea una carpeta o repositorio para guardar los audios generados en cada ejecución de Piper y se crea el script de Python que lo ejecutará para generar el audio a partir del texto:

```
mkdir /piper-audio-files
nano /piper_server.py
chmode +x piper_server.py
```

Este script escucha las peticiones que se envíen al endpoint HTTP definido en el puerto 4040 en la ruta */tts*. Rhasspy incluirá en la petición POST el texto a transformar en audio dentro de un archivo en formato JSON. *Piper-tts* tomará la cadena del JSON y generará un archivo de audio WAV de 22,5kHz que se transformará a 48kHz, formato aceptado por Rhasspy, empleando *sox*. Por un lado se guardará en la carpeta creada para ello, y por otro lado el script devolverá a través de otra petición POST a Rhasspy para que lo pueda reproducir. El contenido del script se adjunta al final de este documento en el Anexo.

De la misma manera que Whisper AI, es necesario que el script esté ejecutándose constantemente y pueda ser llamado por Rhasspy en cualquier momento. Para ello se crea también un servicio en la Raspberry Pi para mantener el script escuchando continuamente los mensajes recibidos a través del endpoint HTTP. Al servicio se le ha definido como *piper-tts* y el contenido de su archivo de configuración se encuentra en el Anexo del final del documento:

```
sudo nano /lib/systemd/system/piper-tts.service
```

Finalmente se active el servicio. En el archivo anterior se definen las variables Restart y RestartSec para que se intente relanzar el servicio siempre que se pare y a intervalos de 5 segundos.

```
sudo systemctl daemon-reload          sudo systemctl start piper-tts
```

Tras esto, el servicio estará corriendo ejecutando Piper TTs y expuesto en la URL http:/localhost:4040/tts ya configurada en Rhasspy para este servicio.

```
● piper-tts.service - Piper TTS Service
     Loaded: loaded (/lib/systemd/system/piper-tts.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-03-16 13:21:48 CET; 2 months 2 days ago
   Main PID: 636 (python)
      Tasks: 3 (limit: 9421)
     Memory: 144.4M
        CPU: 46min 21.014s
     CGroup: /system.slice/piper-tts.service
             ├─  636 /home/pablo/piper-venv/bin/python /home/pablo/piper_server.py
             └─92376 /home/pablo/piper-venv/bin/python /home/pablo/piper_server.py

mar 30 15:23:59 debian python[91988]:  * Detected change in '/home/pablo/whisper_server.py', reloading
mar 30 15:23:59 debian python[636]:  * Restarting with stat
mar 30 15:23:59 debian python[92204]:  * Debugger is active!
mar 30 15:23:59 debian python[92204]:  * Debugger PIN: 140-445-448
mar 30 15:24:42 debian python[92204]: 127.0.0.1 - - [30/Mar/2025 15:24:42] "POST /tts HTTP/1.1" 200 -
mar 30 15:26:44 debian python[92204]:  * Detected change in '/home/pablo/whisper_server.py', reloading
mar 30 15:26:44 debian python[636]:  * Restarting with stat
mar 30 15:26:44 debian python[92376]:  * Debugger is active!
mar 30 15:26:44 debian python[92376]:  * Debugger PIN: 140-445-448
mar 30 15:27:24 debian python[92376]: 127.0.0.1 - - [30/Mar/2025 15:27:24] "POST /tts HTTP/1.1" 200 -
```

Figura 19. *Servicio piper-tts corriendo*

4.2.5 Respuesta por Voz. OpenHAB

A modo de propuesta o mejora del funcionamiento general del asistente de voz y para permitir dotar al usuario de mayor control y autonomía sobre el uso del asistente en todo momento, se ha incorporado una funcionalidad para permitir activar o desactivar la respuesta o realimentación del asistente por voz a los comandos y órdenes indicados por el usuario.

Para facilitar y simplificar el uso de esta función, se ha implementado a través de un nuevo item en OpenHAB, de manera que pueda ser activado tanto por voz, a través de Rhasspy, como por medio de las interfaces disponibles de OpenHAB. Para crearlo mediante la interfaz web hay que acceder a la pestaña de Configuración/Items de OpenHAB y pulsar el botón azul en la esquina inferior derecha. En la siguiente imagen se muestra la configuración, muy simple, que se ha aplicado al item *ActivadorRespuestaAsistenteVoz*.

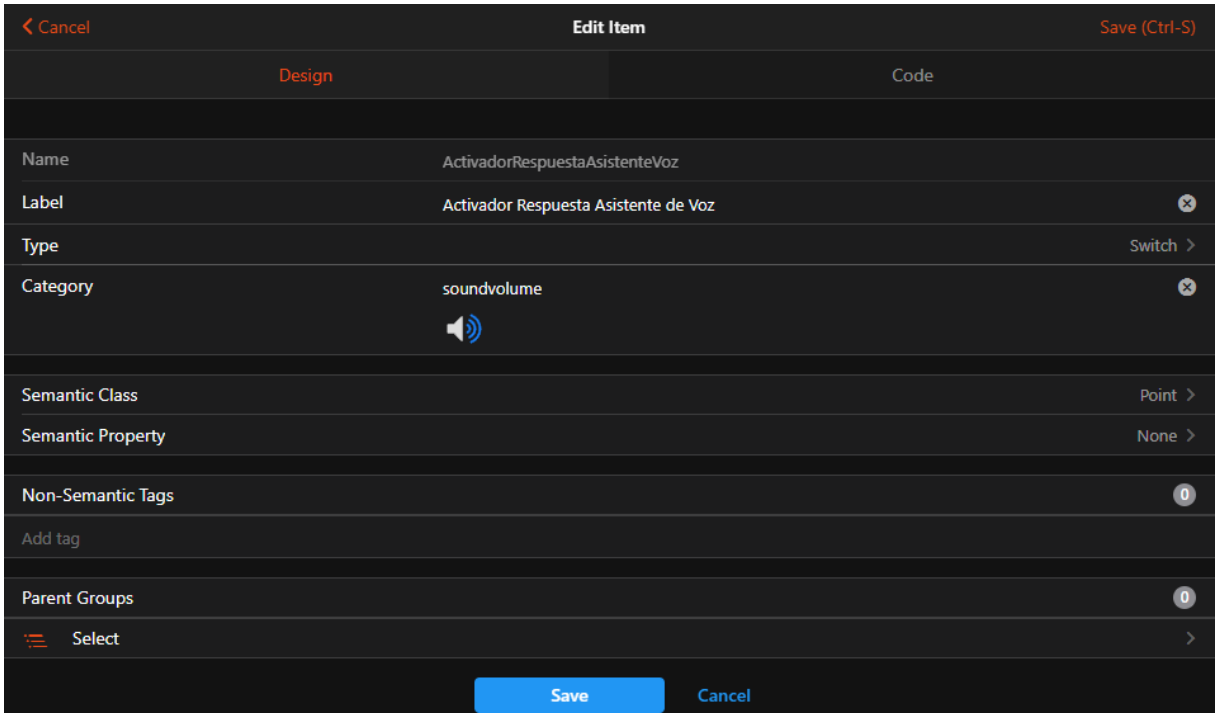

Figura 20. *Configuración Item Respuesta de Voz*

Se trata de un item de tipo switch, virtual en la práctica, al no estar asociado a ningún elemento o dispositivo físico.

Para poder trasladar el valor de este item al entorno de Rhasspy, y poder consultarlo o modificarlo, se ha optado por crear una monitorización constante y en tiempo real del item y almacenar su valor en el fichero *configuracionOpenHAB.json*, que almacena otros datos sobre los items y dispositivos monitorizados en OpenHAB y que se emplea también en el proceso de reconocimiento de intents. Este será explicado más en profundidad en el siguiente apartado.

La monitorización del item, siguiendo la práctica empleada con Whisper AI y Piper TTS, se realiza a través de un servicio de Linux que ejecuta un script de Python con el que se controla el estado del item a través de un WebSocket. Como se indicó en el apartado **2.2.2 Interfaces de Comunicación**, aunque la monitorización a través de la API Rest de OpenHAB es más sencilla, el envío y recepción de las peticiones HTTP introduce un retardo entre la activación

del item y la actualización del parámetro en el JSON, que puede interferir en el procesamiento de intents dando lugar a respuestas erróneas. Este problema, con la monitorización mediante WebSocket y al tratarse de sólo un item, queda resuelto.

De la misma manera que para STT y TTS, se crea primero un entorno virtual y se instala el paquete de *websocket-client* para poder monitorizar el estado del item.

```
python3 -m venv websocket-venv
source websocket-venv/bin/activate
pip install websocket-client
```

A continuación, se crea el script de Python que ejecutará la monitorización constante del estado y los cambios producidos en el item *ActivadorRespuestaAsistenteVoz*. Este, abre un WebSocket con la instancia del log de eventos de OpenHAB donde se grabarán todos los cambios e interacciones con los items. Una vez iniciado, buscará mensajes de tipo *ItemStateChangedEvent* relacionados con el item concreto para detectar los cambios de ON a OFF y viceversa. Cada vez que detecte un cambio, actualizará automáticamente el parámetro *respuesta_voz_activa* del JSON con el valor correspondiente de True o False.

Para conseguir que el script se ejecute de manera constante, se implementa a través de un servicio de Linux en la Raspberry Pi como en los casos anteriores. Al servicio se le ha definido como *monitorizacionVoz* y el contenido de su archivo de configuración se indica en el Anexo del final del documento junto con el script Python:

```
sudo nano /lib/systemd/system/monitorizacionVoz.service
```

Finalmente se active el servicio. En el archivo anterior se definen las variables Restart y RestartSec para que se intente relanzar el servicio siempre que se pare y a intervalos de 5 segundos.

```
sudo systemctl daemon-reload    sudo systemctl start monitorizacionVoz
```

4.3 Arquitectura Completa del Asistente

A continuación, se describe la arquitectura del asistente de voz para el reconocimiento de un intent y el paso de la información y los datos por los distintos procesos y herramientas.

El asistente se encuentra escuchando en todo momento a través del micrófono esperando a reconocer la palabra definida como WakeWord. Una vez detectada, emite un sonido o pitido para informar al usuario de que se va a grabar la voz para recoger la orden indicada.

La voz del usuario es grabada a través del micrófono de entrada instalado y los datos de audio son transmitidos por el servicio de Audio Recording al servicio de Speech-to-Text (Whisper AI) a través de una llamada al servicio instalado en la Raspberry Pi. STT transcribirá a texto plano el audio del usuario con la orden indicada para transmitirlo a su vez al servicio de Intent Recognition

El servicio Intent Recognition, utilizando el archivo de configuración con la definición de las sentencias de reconocimiento para cada intent, trata de identificar alguno de los intents definidos en la transcripción de audio tomando de base las múltiples combinaciones.

En caso de identificar un intent, el servicio Intent Recognition generará un JSON con información sobre la detección y el intent asociado. De este JSON, son relevantes de cara al manejo de intents los siguientes objetos:

- **"Intent" – "name"**: Guarda el nombre del intent reconocido que se definió en el archivo *sentences.ini*
- **"Slots"**: Diccionario que guarda el par clave-valor de las variables definidas en el *sentences.ini* que se hayan detectado en intent. Por ejemplo, acción, ubicación etc.

Este JSON es pasado como entrada del servicio Intent Handling para ejecutar la acción asociada al intent que se ha detectado. En concreto, se pasa al script *intent_handler_v1.py* que ejecuta este servicio, en conjunto con el JSON *configuracionOpenHAB*, que almacena información de los items de OpenHAB relevante para el procesamiento de las órdenes e intents. La lógica del script se encarga de enviar, consultar y ejecutar las acciones pertinentes sobre los items administrados en OpenHAB y, en caso de estar activada, ejecuta la reproducción de un mensaje de retroalimentación sobre la acción que se ha realizado sobre OpenHAB.

El valor del item de OpenHAB para conocer el estado de la reproducción de audio como respuesta se obtiene y monitoriza de manera constante a través del servicio *monitorizacionVoz*. El valor de este item se almacena y actualiza en uno de los objetos del JSON *configuracionOpenHAB* para ser usado por el script del servicio.

Para la reproducción de audio, dentro del script se hace una llamada a la API de Rhasspy para activar el servicio Text-to-Speech (Piper TTS) ,a través del servicio instalado en la máquina, con las palabras a reproducir en texto plano. TTS trata de generar un archivo de audio donde se reproduzca el texto proporcionado y, en caso satisfactorio, envía el archivo .WAV generado al servicio Audio Playing para su reproducción a través de los altavoces o el dispositivo de salida configurado.

Si no se ha podido reconocer en el audio transcrito ninguno de los intents definidos en la configuración de Rhasspy, o si la transcripción de audio a texto falla en algún punto, se emitirá otro pitido o sonido indicando el error en el procesamiento.

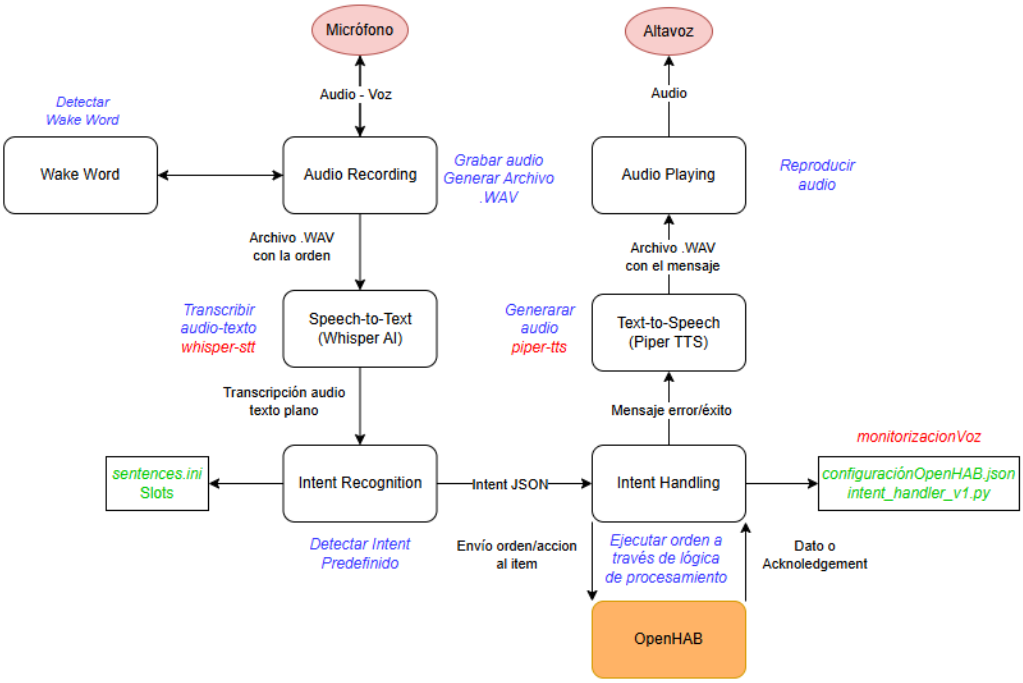

Figura 21. *Diagrama Procesamiento Rhasspy*

5. Resultados

En el siguiente apartado se describen los resultados obtenidos de las pruebas realizadas para comprobar y verificar el comportamiento y funcionamiento del asistente de voz desarrollado.

5.1 Transcripción de audio con Whisper AI

Para verificar el correcto funcionamiento de Whisper AI y que se consigue realizar la transcripción de la voz de un audio a formato texto se realizaron las siguientes pruebas:

- En primer lugar, empleando el micrófono KLIM Voice, se graba una pequeña muestra de audio diciendo la frase "Hola, que tal estás". Para ello, con el micrófono conectado a la Raspberry Pi, se ejecuta el siguiente comando (*arecord -D hw:2,0 -f S16_LE -c 1 prueba.wav*) Con él, el micrófono graba un audio *prueba.wav* en formato de 16kHz estándar.
- Para pasar el archivo de audio al servicio *whisper-stt* creado, se ejecuta el siguiente comando (*curl -X POST –data-binary @prueba.wav http://localhost:5050/stt*) Con él, se llama al servicio instalado que ejecuta Whisper AI y se le pasa como información el archivo de audio grabado.
- Como resultado, se obtiene por consola un fichero JSON cuyo contenido es exclusivamente la transcripción correcta del audio grabado, la frase "Hola, que tal estás"

Con esto, se da por satisfactoria la prueba y funcionamiento de Whisper AI y su servicio asociado creado e instalado.

5.2 Generación de audio con Piper TTS

Para comprobar el funcionamiento adecuado de PiperTTS y verificar que es capaz de generar audio a partir del texto proporcionado se realizaron las siguientes pruebas:

- Directamente se lanza el siguiente comando para proporcionar al servicio de Piper TTS un texto u oración a leer para generar un archivo de audio que lo reproduzca correctamente. (*curl -X POST -H "Content-Type: text/plain" –data "Hola, bienvenido al Hogar Digital" http://localhost:4040/tts --output salida.wav*)
- Con este commando se llama al servicio pasando como parámetro el texto incluído en el campo data para que Piper TTS genere el fichero de audio correspondiente con el nombre *salida.wav*.
- Una vez generado, para comprobar el contenido del mismo y la inteligibilidad de la voz del audio se lanza el siguiente comando (*aplay salida.wav*) para reproducir el archivo de audio. El audio en cuestión resultó ser audio de calidad, entendible y con un todo neutro sin interferencias.

Se da por correcta la prueba en cuestión del correcto y adecuado funcionamiento de Piper TTS

Es necesario remarcar que, aunque ambas pruebas 5.1 y 5.2 resultaron correctas y satisfactorias, el procesado de ambos servicios presentó un ligero retraso en la ejecución y obtención de la respuesta. Esto se debe a que las pruebas de laboratorio se realizaron sobre una Raspberry Pi 4

de tan sólo 1GB de RAM, lo cual limita la capacidad de procesamiento y operativa del asistente al llenarse el buffer de memoria RAM temporalmente durante la ejecución.

5.3 Reconocimiento de órdenes en Rhasspy

Se realizaron pruebas para analizar la lógica de reconocimiento de órdenes por parte del asistente a través de Rhasspy, el procesado de los datos de las mismas y la interacción con los items de OpenHAB y ejecución sobre ellos de las órdenes. Estas pruebas se realizan desde la pantalla principal de la interfaz web de Rhasspy mostrada en la Figura 8.

Primero se prueba el reconocimiento de las diversas órdenes, definidas dentro de Rhasspy mediante sentencias, y la generación de los slots correspondientes a cada uno de los intents reconocidos.

- En el cuadro “Text to recognize” se introducen algunas de las sentencias definidas en Rhasspy y se pulsa el botón Recognize para que se ejecute el motor de Intent Recognition.
- Para la orden *Enciende la luz del salón*, por ejemplo, Rhasspy reconoce correctamente el nombre del Intent, y el contenido de ambos slots definidos para esa sentencia, acción y ubicación.

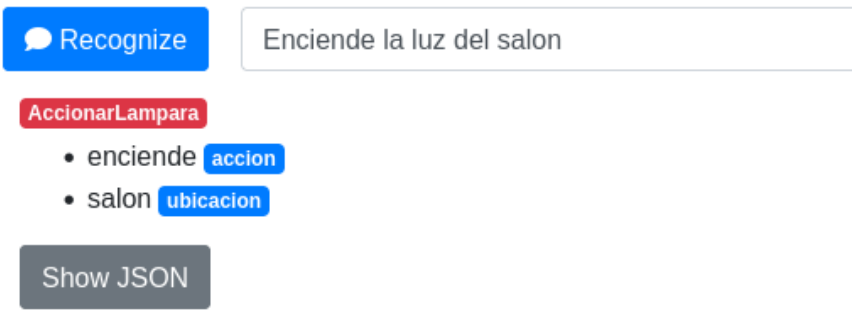

Figura 22. *Test orden "Enciende la luz del salon"*

- Para la orden *Sube la persiana del dormitorio* se reconocer al igual Intent, slots y su contenido.

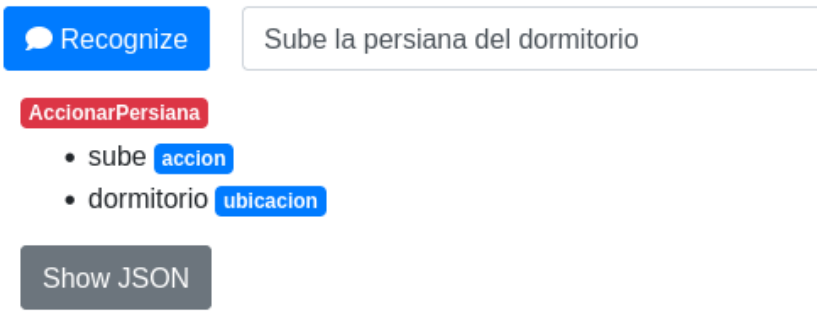

Figura 23. *Test orden "Sube la persiana del dormitorio"*

- Por último, se probaron también ordenes no definidas o u órdenes para dispositivos en ubicaciones incorrectas, para verificar también el correcto manejo de Rhasspy de las variables de ubicación definidas para las sentencias como se muestra en la imagen.

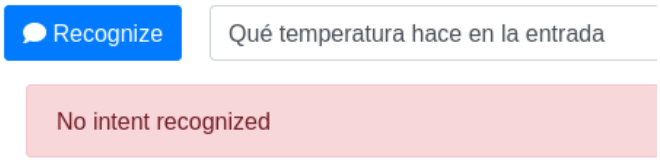

Figura 24. *Test ubicación errónea*

De esta manera se confirma que la estructura y las variables definidas en los Slots y el *sentences.ini* de Rhasspy es correcta y su detección funciona adecuadamente.

5.4 Procesado e interacción de órdenes con OpenHAB

Se realizaron pruebas para verificar que la conexión e integración de Rhasspy con los items de los dispositivos del HDA de OpenHAB es correcta y existe una interacción completa entre ambos. Para realizar esta prueba se empleó el mismo cuadro de las pruebas anteriores de la Figura 8. En esta ocasión, para activar el procesado de los Intents reconocidos, se activa antes del reconocimiento la casilla “Handle” a la derecha del cuadro de texto.

- Dado que la lógica del *manejador_intents.py* está estructurado en base a los tipos de dispositivos asociados a los items, se realiza una prueba para cada uno de ellos. En concreto se probaron:
  - Encendido de la lámpara del baño
  - Cambio de color de la lámpara del salón
  - Apertura de la persiana de la entrada
  - Consulta temperatura del dormitorio
- Todas las acciones se invocaron mediante una orden adecuada al formato definido en el *sentences.ini*. En todos los casos la acción se llevó a cabo satisfactoriamente y el asistente de voz reprodujo a través del altavoz los mensajes de respuesta definidos para cada uno de los elementos.
- En primera instancia, la apertura de la persiana no se realizó correctamente debido a que el item asociado a la persiana de la entrada, a diferencia de las otras que son un *RollershutterItem*, está implementado como un *SwitchItem*, lo cual requería que los comandos enviados fuesen ON/OFF en lugar de UP/DOWN. Corregida la lógica en el script para dicho item, la ejecución se realizó correctamente.

Cabe destacar que, pese a que el asistente reprodujo a través de los altavoces los mensajes de respuesta, entre la ejecución de la orden en OpenHAB sobre el item y la reproducción del mensaje había un pequeño intervalo de tiempo de delay provocado por la limitación de RAM de la Raspberry en la que se realizaron las pruebas.

Con esta prueba queda comprobado el correcto funcionamiento de la lógica diseñada para el manejo de órdenes y la interacción y ejecución de las mismas sobre los items y elementos de OpenHAB.

5.5 Prueba completa de ejecución del asistente de voz

Como prueba final, se analizó la ejecución completa del asistente de voz involucrando todos los elementos desarrollados e implementados. La prueba consiste en la ejecución de una orden o comando, al igual que en las pruebas anteriores, pero ejecutándola mediante la voz, siendo esta transcrita por el servicio de STT, interpretada y procesada por Rhasspy y los demás elementos del asistente.

- El procedimiento de la prueba consiste en, mediante la voz, activar la escucha del asistente a través del servicio Wake Word y tras eso dictar la orden correspondiente para que el audio sea grabado, transcrito a texto y procesado por la lógica de Rhasspy.
- A la hora de realizarla, se detectó un problema en la grabación de audio. De manera aislada, el micrófono KLIM Voice graba audio correctamente al ejecutar una grabación desde la Raspberry Pi. Sin embargo, al integrar el micrófono dentro de Rhasspy, la grabación de audio no se realiza generando un error.
- El problema deriva de la tasa de muestreo del audio grabado. De manera predeterminada el micrófono muestrea el audio grabado a 48kHz, como al grabar desde la Rasbperry Pi, de forma correcta. Rhasspy por su lado está diseñado para trabajar exclusivamente con audio muestreado a 16kHz, de manera que, debido a esta incompatibilidad, Rhasppy no permite o es capaz de utilizar el KLIM Voice como dispositivo de entrada de audio.
- Para poder llevar a cabo la prueba aún así, se optó por apoyarse en el servicio que ejecuta Piper TTS que, como se ve en el script *piper_server.py* del Anexo II, muestrea el audio generado a 16kHz. Con esto, se genero mediante el comando del apartado 5.2 (*curl -X POST -H "Content-Type: text/plain" –data "Enciende la lampara del salon" http://localhost:4040/tts --output salida.wav*) un archivo de audio adecuado para Rhasspy con la orden a ejecutar por el asistente.
- Finalmente, se pasó el archivo de audio manualmente a través de la interfaz web de Rhasspy para realizar la prueba, simulando que el archivo de audio fuese el generado por el micrófono. De esta manera, el asistente de voz procesó adecuadamente la orden y todos los demás elementos y herramientas del asistente de voz operaron y funcionaron correctamente.

Se trató de configurar el micrófono para grabar audio a 16kHz a la hora de interactuar con Rhasspy sin éxito. Pese a configurarlo manualmente, en el *Restart* solicitado tras cualquier cambio en Rhasspy hacer que la configuración del mismo vuelva al estado predeterminado.

Pese al problema indicado, se puede considerar que el funcionamiento y desarrollo del asistente de voz ha sido correcto y que tiene unos resultados bastante satisfactorios. La incompatibilidad del micrófono se propondrá como mejora o punto de evolución futuro para analizar la corrección del problema o el cambio de micrófono.

6. Presupuesto

En este apartado se realizará un análisis a nivel económico del presupuesto y los medios necesarios para llevar a cabo el desarrollo del proyecto. En él se incluye tanto el precio de los componentes físicos como el coste del desarrollo humano.

Tabla 1. *Recursos Hardware Interfaz Raspberry Pi*

Ítem	Unidades	Precio (€/Unidad)
Raspberry Pi 4 B [57]	1	68,62€
Tarjeta SD de 32 GB [58]	1	9,20€
Fuente de alimentación RPi [59]	1	3,50€
Total		**81,32€**

Tabla 2. *Recursos Hardware de Interacción con el Sistema*

Ítem	Unidades	Precio (€/Unidad)
Micrófono KLIM Voice [60]	1	**19,49€**
Altavoces Tacens Mars Gaming MS1 [61]	1	**9,90€**
Total		**29,39€**

Los recursos software usados en el proyecto no suponen ningún coste adicional al tratarse de recursos de código abierto.

Tabla 3. *Recursos Software*

Ítem
Imagen ISO de la Raspberry Pi
Rhasspy 2.5: asistente de voz [62]
Whisper AI: sintetización de audio a texto
Piper TTS: generación de audio a partir de texto
Python: scripts de automatización
OpenHAB
Draw.io: generar diagrama [63]

Tabla 4. *Total Recursos del Proyecto*

Ítem	Precio (€/Unidad)
Recursos hardware interfaz Raspberry Pi	**81,32€**
Recursos hardware adicionales	**29,39€**
Recursos software	**0,00€**
Total	**110,71€**

Además de los recursos físicos, se debe tener en cuenta también el coste humano. Para ello, se ha realizado un desglose con las horas empleadas aproximadamente para cada una de las fases del proyecto. Este resumen se muestra en la siguiente figura, con un total de 319 horas:

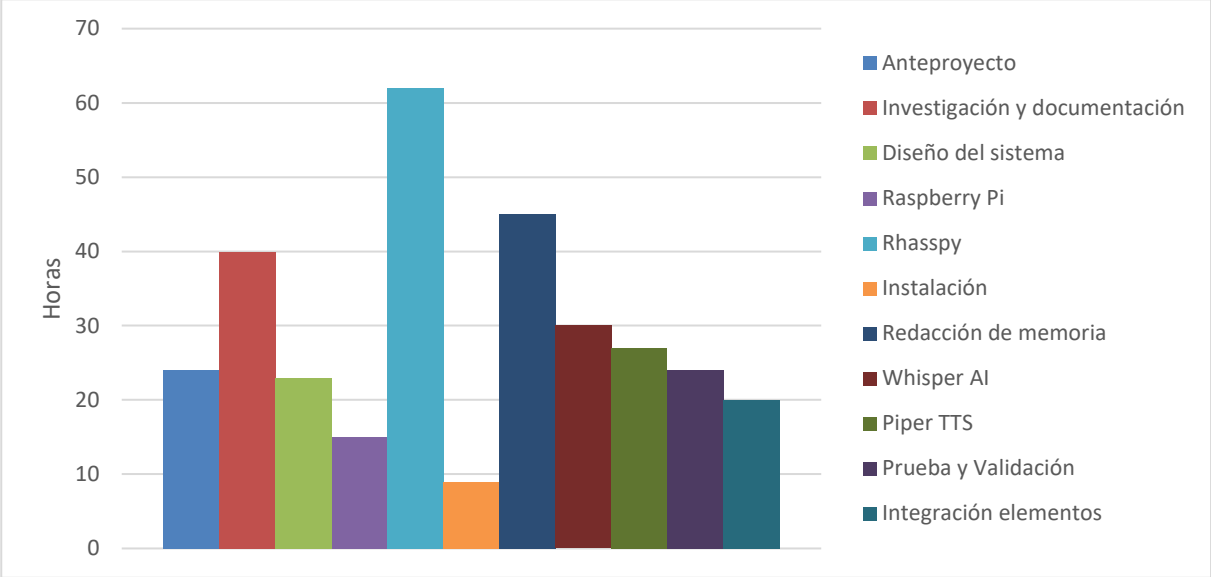

Figura 25. *Desglose de Horas Empleadas*

Teniendo esto en cuenta, se puede hacer una estimación del coste humano del proyecto. El salario promedio en España de un ingeniero de telecomunicaciones es de 36.000€ brutos anuales **[64]**, que corresponde a unos 2.257,40€ al mes y 14,11€/hora. Por tanto, el coste humano del proyecto resultará en un total de 4.501,09€.

Tabla 5. *Coste Humano*

Ítem	Horas	Precio
Ingeniero de Telecomunicaciones	319	14,11€/hora
Total		4.501,09€

Con todos los cálculos hechos, se puede tener una estimación completa y aproximada del presupuesto total de ejecución del proyecto.

Tabla 6. *Presupuesto Total*

Ítem	Precio
Coste recursos del proyecto	110,71€
Coste humano / de personal	4.501,09€
Total	**4.611,80€**

Remarcar que los datos utilizados para los cálculos de presupuesto responden a aproximaciones generales y que los mismo pueden varias en función de diversos factores como la inflación, aumento de los costes de producción o cambios en la fiscalidad entre otros.

7. Impacto del proyecto

Este proyecto, basado en el desarrollo de un asistente de voz local, abierto y multimodal, además del impacto puramente técnico, tiene una serie de implicaciones en otros ámbitos de relevancia así como aportaciones o contribuciones a los Objetivos de Desarrollo Sostenible (ODS).

7.1 Impacto Social

Desde el punto de vista social, el proyecto promueve el acceso a tecnologías avanzadas de automatización y monitorización del hogar sin la necesidad de depender de soluciones comerciales concretas cerradas y de alto coste. Al estar desarrollado con herramientas open source y ejecutarse en hardware de bajo coste como la Raspberry Pi, la solución puede ser adoptada por grupos de gente con recursos limitados, reduciendo así la brecha digital. Además, la naturaleza modular y personalizable del asistente permite que sea adaptado a personas discapacitadas o con necesidades especiales para mejorar la accesibilidad tecnológica del hogar y fomentando su autonomía personal.

7.2 Impacto en salud y seguridad

Al tratarse de un sistema que se ejecuta completamente en local, sin necesidad de conexión externa a internet, se asegura la exposición nula de datos personales de los usuarios al exterior, especialmente conversaciones en el entorno privado del hogar al tratarse de un sistema de interacciones de voz. Con esto se contribuye a un entorno digital más seguro donde el usuario controla y gestiona sus propios datos. Asimismo, el control de dispositivos y elementos del hogar como puertas, luces o persianas puede mejorar la seguridad y la experiencia del día a día en personas con problemas o movilidad reducida mejorando su calidad de vida.

7.3 Impacto ambiental

El empleo de una Raspberry Pi como núcleo de ejecución del asistente de voz garantiza un bajo consumo energético frente a otras soluciones comerciales o dependientes de servicios externos de alto consumo como la computación en la nube. De esta manera, se prioriza y fomenta la eficiencia energética y el uso responsable de los recursos. Al utilizar herramientas y recursos open source y reutilizables, se contribuye a prolongar la vida útil de los dispositivos y a reducir la obsolescencia programada, en línea con los objetivos de sostenibilidad.

7.4 Impacto económico

Desde una perspectiva económica, se demuestra que es posible desarrollar e implementar soluciones domóticas completas, funcionales y de bajo coste, eliminando barreras de entrada a usuarios y empresas para la adquisición y adopción de estas tecnologías. El uso de tecnologías libres promueve el desarrollo tecnológico descentralizado, facilitando la creación de soluciones eficientes sin la necesidad de grandes inversiones iniciales.

7.5 Impacto tecnológico

El proyecto pone en valor el uso de herramientas open source como Rhasspy o Piper TTS, integrándose en una solución única potente, escalable y modular que, aunque orientada al hogar, puede ser implementadas en distintos entornos. Ofrece una alternativa ética y viable a los asistentes de voz comerciales, con la capacidad de poder ser replicada fácilmente. Su modularidad permite ampliar y extender sus funciones en base a las necesidades de cada entorno, usuario o situación.

7.6 Contribución a los ODS

Este sistema está alineado con varios de los ODS definidos por la ONU dado su impacto y beneficios en diferentes aspectos

- **ODS 9** (Industria, innovación e infraestructura): Se promueve la innovación mediante la implementación de tecnologías libres y accesibles, fomentando la infraestructura sostenible
- **ODS 10** (Reducción de las desigualdades): Facilita el acceso de grupos con recursos limitados o con necesidades especiales a tecnología de alto nivel.
- **ODS 12** (Producción y consumo responsable) y ODS 13 (Acción por el Clima) : Se minimiza el consumo energético y se promueve la reutilización y eficiencia de los recursos alargando así su vida útil
- **ODS 16** (Paz, justicia e instituciones públicas): Se protege la privacidad de los usuarios resguardando sus datos de carácter personal de manera transparente.
- **ODS 17** (Alianza para lograr los objetivos): El uso de tecnologías open source favorece la colaboración y el desarrollo sostenible entre comunidades.

8. Conclusiones

El presente Proyecto Fin de Grado ha abordado con éxito el diseño e implementación de un asistente de voz local, abierto y multimodal en el Hogar Digital Accesible empleando exclusivamente tecnologías libres y de código abierto para dar lugar a una solución robusta, eficiente y éticamente responsable. Este proyecto surge como respuesta a las limitaciones que presentan las soluciones comerciales existentes, como Google Home o Alexa, en materia de privacidad, dependencia de conexión a internet constante o a servicios en la nube y restricciones de desarrollo impuestas por los fabricantes.

La integración del asistente, ejecutado sobre una Raspberry Pi independiente, ha permitido cumplir el objetivo de garantizar su funcionamiento de manera completamente offline, aportando así seguridad, privacidad y un mayor control sobre la información del usuario. Además, la arquitectura del asistente, basada en la herramienta Rhasspy, ofrece una amplia modularidad y escalabilidad del sistema, facilitando así la evolución y mejora a futuro del mismo y proporcionando un alto grado de personalización.

Desde la perspectiva técnica, durante el desarrollo se han analizado diferentes herramientas para abordar cuestiones clave como el reconocimiento y transcripción de audio o la síntesis de voz. Para llevarlas a cabo se han seleccionado aquellas herramientas que por su rendimiento y compatibilidad mejor se adaptan a las características y necesidades del HDA y la plataforma hardware utilizada, como son Whisper AI y Piper TTS respectivamente.

Los resultados obtenidos en las pruebas llevadas a cabo han demostrado que es viable desarrollar y desplegar un sistema de interacción por voz eficiente, con tiempos de respuesta aceptables, bajo consumo energético y un nivel de precisión, tanto en la escucha como en la reproducción del habla, suficiente para entornos domésticos reales.

Además de haberse alcanzado los objetivos técnicos planteados, el proyecto supone una aportación al entorno a nivel de accesibilidad y sostenibilidad tecnológica y energética. El uso de hardware de bajo coste, así como la flexibilidad que aportan las diversas tecnologías de código abierto utilizadas, convierten la solución planteada y desarrollada en este proyecto en una alternativa real, replicable y escalable de los asistentes comerciales, alineada además con los principios de autonomía personal, ética y privacidad y seguridad.

En definitiva, el proyecto desarrollado sienta las bases y demuestra la posibilidad de construir de forma autónoma un asistente de voz privado y personalizable, contribuyendo significativamente al ecosistema del Hogar Digital Accesible y a la democratización y promoción de la tecnología libre.

8.1 Trabajos futuros

A pesar de la satisfacción y éxito alcanzado con el desarrollo e implementación del asistente de voz, se es consciente de las múltiples vías de mejora y evolución que un proyecto de tales características tiene. Algunas de estas vías de desarrollo se exponen a continuación:

- **Mejora de usabilidad/accesibilidad:** El diseño actual incorpora la retroalimentación o respuesta por voz tras la interacción con el asistente. Resultaría interesante explorar la incorporación de otros sistemas, luminosos o visuales por ejemplo, para diversificar las opciones de retroalimentación para mejorar la interacción de aquellos usuarios con dificultades auditivas.
- **Resolución de la incompatibilidad micrófono-Rhasspy:** Dado el problema detectado durante la realización de las pruebas, que afecta a los servicios de Audio Recording y Wake Word, sería recomendable explorar la posibilidad de forzar la grabación de audio del micrófono a 16kHz de forma predeterminada para hacerlo compatible con Rhasspy. En caso de no poder lograrlo, sería recomendable sustituirlo por otro compatible para poder disfrutar de las características totales del asistente de voz.
- **Dispositivo de captura y reproducción de audio único:** Para simplificar los procesos de interacción con el usuario y mejorar los flujos de entrada y salida de datos de Rhasspy, sería interesante sustituir el micrófono y el altavoz por un único dispositivo que desempeñe ambas funciones simplificando así el uso y configuración necesaria y posibles focos de fallos o errores.
- **Integración de interacción multimodal avanzada:** El asistente dispone únicamente de retroalimentación o ayuda sonora, mediante respuestas de voz, para proporcionar información adicional o contextual sobre las acciones tomadas. Sería interesante explorar la incorporación de otras modalidades visuales o gestuales de interacción asistente-usuario para ofrecer una experiencia de uso del asistente más fluida, versátil y adaptada al usuario.
- **Gestión de múltiples equipos de interacción:** Actualmente sólo se plantea el uso de un micrófono y altavoz. Para hogares o entornos domésticos grandes o con múltiples estancias alejadas, sería adecuado analizar la posibilidad de utilizar dos o más puntos de captura y reproducción de audio, dispersos por el hogar, para poder comunicarse con el asistente desde cualquier estancia y manteniendo un único núcleo central de procesamiento del asistente (Raspberry Pi)
- **Evolución progresiva del asistente:** Dado el uso como laboratorio que tiene el HDA, continuar expandiendo las capacidades del asistente de voz a más tecnologías que se incorporen al HDA tanto en términos de nuevos dispositivos domóticos como la adaptación y extensión a nuevas plataformas de gestión y monitorización más allá de OpenHAB.

9. Referencias

[1] «¿Que es Raspberry Pi? - Raspberry Pi». Accedido: 5 de abril de 2025. [En línea]. Disponible en: https://raspberrypi.cl/que-es-raspberry/

[2] «¿Qué es Raspberry Pi y para qué sirve? - Definición», GEEKNETIC. Accedido: 5 de abril de 2025. [En línea]. Disponible en: https://www.geeknetic.es/Raspberry-Pi/que-es-y-para-que-sirve

[3] cgarcia, «¿Qué es Raspberry Pi y para qué sirve? | Escuela de programación, robótica y pensamiento computacional | Codelearn.es». Accedido: 5 de abril de 2025. [En línea]. Disponible en: https://codelearn.es/blog/que-es-raspberry-pi-y-para-que-sirve/

[4] L. Calvo, «Raspberry Pi: Qué es, cómo funciona y proyectos que puedes realizar», GoDaddy Resources - Spain. Accedido: 5 de abril de 2025. [En línea]. Disponible en: https://www.godaddy.com/resources/es/tecnologia/que-es-raspberry-pi

[5] «Raspberry Pi 3 Model B+ - Placa base Raspberry - LDLC». Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://www.ldlc.com/es-es/ficha/PB00246555.html

[6] R. P. Ltd, «Buy a Raspberry Pi 3 Model A+», Raspberry Pi. Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://www.raspberrypi.com/products/raspberry-pi-3-model-a-plus/

[7] P. F. Hola, soy P. S. administrador de sistemas Linux, y me apasiona la R. P. y todos los proyectos sobre este tema H. creado este sitio para compartir con ustedes lo que he aprendido al respecto, «Raspberry Pi 5: Fecha de lanzamiento y especificaciones – RaspberryTips». Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://raspberrytips.es/raspberry-pi-5-informacion/

[8] A. Castro, «OpenHAB: Transforma tu Hogar con Automatización Inteligente». Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://saberpunto.com/tecnologia/openhab-transforma-tu-hogar-con-automatizacion-inteligente/

[9] J. A. A. Durán, «Analizamos openHAB, la plataforma domótica a nuestro alcance», Panel Sistemas. Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://www.panel.es/plataforma-openhab-domotica-a-nuestro-alcance/

[10] «Things». Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/configuration/things.html

[11] «Things II». Accedido: 6 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/concepts/things.html

[12] «Items». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/configuration/items.html#items

[13] «Main UI». Accedido: 11 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/mainui/

[14] «Pages». Accedido: 11 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/tutorial/pages_intro.html#pages-and-other-user-interfaces

[15] «HABPanel». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/docs/ui/habpanel/habpanel.html

[16] kuskitkd, «Rules», kuskitkd. Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://kuskitkd.wordpress.com/2021/02/18/openhab-primeras-automatizaciones-rules/

[17] «WebSockets». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://next.openhab.org/docs/configuration/websocket.html

[18] vmotos vis0r, «Hey MYCROFT! AI para todos». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.hackplayers.com/2017/03/hey-mycroft-ai-para-todos.html

[19] R. Velasco, «Mycroft, una nueva Inteligencia Artificial libre basada en Snappy Ubuntu Core», RedesZone. Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.redeszone.net/2015/09/12/mycroft-una-nueva-inteligencia-artificial-libre-basada-en-snappy-ubuntu-core/

[20] «Why use Mycroft AI? | Mycroft AI». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://mycroft-ai.gitbook.io/docs/about-mycroft-ai/why-use-mycroft

[21] «Descubre Mycroft AI, el asistente de voz de código abierto». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.toolify.ai/es/ai-news-es/descubre-mycroft-ai-el-asistente-de-voz-de-cdigo-abierto-1538397

[22] «Rhasspy». Accedido: 15 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/

[23] «Servicios - Rhasspy». Accedido: 15 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/services/#web-server

[24] «Audio Input - Rhasspy». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/audio-input/

[25] «rhasspy/docs/tutorials.md at master · rhasspy/rhasspy», GitHub. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://github.com/rhasspy/rhasspy/blob/master/docs/tutorials.md

[26] «Wake Word I», Sherlock. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://ip-team4.intia.de/pages/knowledge/wake-word.html

[27] «Wake Word II». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/wake-word/

[28] «STT II». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/speech-to-text/

[29] «STT I (Vosk/Coqui)», Rhasspy Voice Assistant. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://community.rhasspy.org/t/vosk-coqui-stt-asrs-integration/2848

[30] *rhasspy/rhasspy-remote-http-hermes*. (23 de junio de 2021). Python. Rhasspy. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://github.com/rhasspy/rhasspy-remote-http-hermes

[31] «Intent Recognition I». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/intent-recognition/

[32] «Intent Recognition II». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://community.openhab.org/t/oh-voice-control-using-rhasspy-and-gstreamer-installation-and-configuration/108300

[33] «Intent Handling I». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/intent-handling/

[34] H. Assistant, «Intent Handling - Home Assistant», Home Assistant. Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.home-assistant.io/integrations/rhasspy/

[35] «TTS I». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/text-to-speech/

[36] H. Herrero, «TTS II». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.bujarra.com/rhasspy-control-por-voz-segura-en-home-assistant-en-castellano/

[37] «TTS - NanoTTS», Sherlock. Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://ip-team4.intia.de/pages/knowledge/tts/nano-tts.html

[38] «Services». Accedido: 26 de abril de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/services/

[39] H. Herrero, «Audio Playing I». Accedido: 26 de abril de 2025. [En línea]. Disponible en: https://www.bujarra.com/rhasspy-control-por-voz-segura-en-home-assistant-en-castellano/

[40] «Audio Playing II», Rhasspy Voice Assistant. Accedido: 26 de abril de 2025. [En línea]. Disponible en: https://community.rhasspy.org/t/audio-play-through-remote-http/1813

[41] «Dialogue Manager», Rhasspy Voice Assistant. Accedido: 26 de abril de 2025. [En línea]. Disponible en: https://community.rhasspy.org/t/fully-support-the-hermes-or-hermod-protocol/41

[42] «La API Whisper de OpenAI facilita la conversión de voz a texto». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.datacamp.com/tutorial/converting-speech-to-text-with-the-openAI-whisper-API

[43] «Presentamos Whisper». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://openai.com/es-ES/index/whisper/

[44] E. S. Zawadzki, «Whisper, la herramienta de inteligencia artificial de OpenAI para convertir archivos de audio a texto», Entrepreneur. Accedido: 27 de abril de 2025. [En línea].

Disponible en: https://www.entrepreneur.com/es/tecnologia/whisper-la-herramienta-de-inteligencia-artificial-de/444821

[45] «Experience Lightning-Fast Realtime AI Voice with Piper TTS on Windows». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.toolify.ai/ai-news/experience-lightningfast-realtime-ai-voice-with-piper-tts-on-windows-1117947

[46] «Piper Tts Text-to-Speech Overview | Restackio». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.restack.io/p/piper-tts-answer-cat-ai

[47] «Piper Text-to-Speech - Voices». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.openhab.org/addons/voice/pipertts/

[48] «Tts — pipecat-ai documentation». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://pipecat-docs.readthedocs.io/en/stable/api/pipecat.services.piper.tts.html

[49] «KLIM™ Voice Micrófono USB con Base para Ordenador - Micro de Escritorio, Micrófono para Jugadores - Verde y Negro - Nueva Versión : Amazon.es: Informática». Accedido: 21 de junio de 2025. [En línea]. Disponible en: https://www.amazon.es/dp/B06VVHMLBL

[50] «reSpeaker 4-Mic Array for Raspberry Pi - AC108 Audio Codec, 4 Analog Microphones, 12 Programable RGB LEDs, 2 Grove ports, attached with NLU software algorithms, VAD,DOA, KWS». Accedido: 21 de junio de 2025. [En línea]. Disponible en: https://www.seeedstudio.com/ReSpeaker-4-Mic-Array-for-Raspberry-Pi.html

[51] «Sony Cámara Playstation Eye ps3 : Amazon.es: Videojuegos». Accedido: 21 de junio de 2025. [En línea]. Disponible en: https://www.amazon.es/Sony-C%C3%A1mara-playstation-eye-ps3/dp/B000W3YQ1Y

[52] «STJF Mini micrófono USB para raspberry pi 3B + estudio conversación de voz canto KTV micrófono con soporte para PC portátil micrófono - AliExpress 502», aliexpress. Accedido: 21 de junio de 2025. [En línea]. Disponible en: //es.aliexpress.com/item/1005008502358266.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=&aff_short_key=

[53] «Altavoces gaming MS1 - Mars Gaming». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://es.marsgaming.eu/es/audio/altavoces-gaming-ms1_ms1

[54] «Almacenamiento mínimo SD». Accedido: 3 de mayo de 2025. [En línea]. Disponible en: https://www.kingston.com/es/blog/personal-storage/choosing-storage-for-raspberry-pi

[55] E. Catalina Vasile, «Interfaces del hogar digital mediante pantallas y espejos inteligentes», Proyecto de Fin de Grado, Universidad Politécnica de Madrid, Madrid, 2023.

[56] «Audio Output - Rhasspy». Accedido: 4 de mayo de 2025. [En línea]. Disponible en: https://rhasspy.readthedocs.io/en/latest/audio-output/

[57] «Raspberry Pi 4 Modelo B (4GB) : Amazon.es: Informática». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.amazon.es/Raspberry-Pi-4595-Modelo-

GB/dp/B09TTNF8BT?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&smid=A1AT7YVPFBWXBL&gQT=2&th=1

[58] «Sandisk Ultra MicroSDHC 32GB UHS-I U1 A1 Clase 10 | PcComponentes.com», PcComponentes. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.pccomponentes.com/sandisk-ultra-microsdhc-32gb-uhs-i-u1-a1-clase-10

[59] «Raspberry Fuente de Alimentación USB-C para Raspberry Pi 4 5V 3A 15W Blanca | PcComponentes.com», PcComponentes. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.pccomponentes.com/raspberry-fuente-de-alimentacion-usb-c-para-raspberry-pi-4-5v-3a-15w-blanca

[60] «KLIM™ Voice Micrófono USB con Base para Ordenador - Micro de Escritorio, Micrófono para Jugadores - Verde y Negro - Nueva Versión : Amazon.es: Informática». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.amazon.es/KLIM-Voice-Micr%C3%B3fono-Base-Ordenador/dp/B06VVHMLBL?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&psc=1&smid=A1AO1PX8J2IHMT&gQT=2

[61] «Tacens Mars Gaming MS1 Altavoces 2.0 Negros | PcComponentes.com», PcComponentes. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.pccomponentes.com/tacens-mars-gaming-ms1-altavoces-20-negros

[62] *rhasspy/rhasspy*. (18 de junio de 2025). Shell. Rhasspy. Accedido: 18 de junio de 2025. [En línea]. Disponible en: https://github.com/rhasspy/rhasspy

[63] «Draw.io». Accedido: 18 de junio de 2025. [En línea]. Disponible en: https://app.diagrams.net/

[64] «¿Cuánto Cobra un Ingeniero de Telecomunicaciones? (Sueldo 2025) | Jobted.es». Accedido: 18 de junio de 2025. [En línea]. Disponible en: https://www.jobted.es/salario/ingeniero-telecomunicaciones

Anexo I: Manual de Usuario

En el presente anexo se explicará de forma guiada el uso y utilización de los elementos software y plataformas empleados en el desarrollo del proyecto y confección del asistente virtual. En concreto, se detallarán los siguientes apartados:

1. Configuración inicial Raspberry Pi
2. Funcionamiento servicio Whisper AI
3. Funcionamiento servicio Piper TTS
4. Funcionamiento reconocimiento Intents
5. Adición nuevos Intents
6. Funcionamiento servicio MonitorizacionRespuestaVoz

1. Configuración Raspberry Pi

En este primer apartado se describen los pasos iniciales necesarios tras la instalación satisfactoria del sistema operativo en la Raspberry Pi para poder trabajar de manera adecuada y establecer la configuración necesaria para el correcto funcionamiento de la máquina que albergará el asistente de voz.

1.1 Conexión a la red

La primera y más sencilla opción para conectar la Raspberry Pi a internet de manera rápida y sin necesidad de configuración añadida es a través de cable Ethernet. De esta manera, a la máquina se le asignará directamente una dirección IP a través de DHCP y dispondrá de conexión a la red de manera inmediata. Esta opción está limitada a la cercanía al router o punto de acceso del hogar y la disponibilidad de un cable de red para poder conectarlo.

Por esto, y para ofrecer la capacidad de instalar la Raspberry Pi en el lugar más conveniente o deseado de la casa, se configurará el acceso a la red a través de la conexión WiFi inalámbrica.

Dentro del escritorio principal del SO de la Raspberry Pi, se selecciona el icono de red en la esquina superior derecha y se escoge la red deseada entre las disponibles. El sistema solicitará que se introduzca la contraseña de la red. Una vez introducida, se dispondrá de acceso a la red.

Para verificar que la conexión está establecida, y visualizar la dirección IP asignada por DHCP, ejecutar el comando *ifconfig* o *ip a* en la consola de comandos.

1.2 Asginación de IP fija

Sea cual sea la forma escogida de conexión a la red, de manera predeterminada DHCP habrá asignado una ip libre aleatoria dentro del rango disponible del router. Dado que Rhasspy, durante el uso habitual del asistente de voz, ejecutará ciertos servicios sobre la máquina local y que ésta debe mantener una conexión constante con la Raspberry Pi donde se encuentra instalado OpenHAB en el HDA, será necesario que la Raspberry Pi mantenga una dirección IP estática para mantener una conexión estable en el tiempo con estos componentes aunque se produzcan reinicios de las máquinas.

Esto se puede llevar a cabo de manera sencilla a través de la consola de comandos editando el fichero */etc/dhcpcd.conf* mediante el comando: *sudo nano /etc/dhcpcd.conf*

Dentro del fichero, buscar las líneas donde venga el ejemplo de configuración de una dirección IP estática. A continuación, descomentar las líneas que hacen referencia a la conectividad IPv4 e incluir los valores de los parámetros correspondientes en función de la red local a la que se conecta. En concreto, se debe indicar la IP deseada, la IP del router o punto de acceso, la IP del DNS a utilizar y el nombre de la interfaz a la que se le asignará la IP estática en cuestión. El contenido del fichero respecto a este debe ser similar a este:

```
Interface eth0
Static ip_address=192.168.1.100/24
Static routers= 192.198.1
Static domain_name_servers=192.168.1.1
```

1.3 Activación de acceso remoto (SSH)

Para facilitar la operación y administración remota de la Raspberry Pi sin necesidad de depender de un monitor o pantalla, se habilita el servicio de acceso remoto SSH.

Desde el escritorio de la Raspberry Pi, abrir el menú de inicio y seleccionar la opción "Configuración de Raspberry Pi" dentro del menú "Preferencias". En la pestaña interfaces, activar la opción SSH y confirmar la selección en la ventana emergente.

De forma inmediata, el servicio queda activo y se puede acceder mediante consola de comandos a la Raspberry Pi de forma remota. Para ello, en la máquina remota ejecutar el siguiente comando: *ssh pi@<ip_raspberry_pi>*. Tras introducir la contraseña del usuario, se habrá accedido a la Raspberry Pi.

2. Funcionamiento servicio Whisper AI

Como se explicó anteriormente en la descripción de la solución propuesta, Whisper AI es un sistema de reconocimiento automático de voz desarrollado por la empresa OpenAI. En este proyecto, se utiliza como motor del servicio de Speech-to-Text (STT) para realizar sus funciones de reconocimiento y conversión de las órdenes de audio a texto dentro del asistente de voz para poder ser procesadas por Rhasspy.

Aunque Whisper AI cuenta con un modelo basado en computación en la nube, el modelo empleado en este proyecto es el de ejecución en local en la Raspberry Pi.

Tras la detección de la Wake Word, el asistente procede a grabar todo el audio con el contenido de la orden del usuario y se lo envía a Whisper AI en formato WAV para su procesamiento. Whisper AI analiza los fragmentos de audio y los transcribe a texto. El texto resultante es devuelto a Rhasspy para ser interpretado y ejecutar el reconocimiento de intents.

En el Anexo II. "Whisper_server.py" se describe el script que crea y ejecuta el servidor que emplea la instalación local de Whisper AI para llevar a cabo las transcripciones de audio a texto.

En él se expone el servidor a través del puerto 5050 de la Raspberry Pi para poder ser ejecutado por Rhasspy a través de la URL *http://localhost:5050/stt*.

Para mantener el continuo funcionamiento del servidor, dado el carácter asíncrono del asistente de voz, en el Anexo II. "Whisper-stt.service" se incluye el código de la configuración del servicio local que mantiene en ejecución constante el servidor para ser accesible desde la URL indicada.

Entrando en detalle del funcionamiento del servidor *whisper_server.py*, el modelo instalado de Whisper se configura para ejecutarse con el modo de cómputo *int8* para reducir el uso de CPU del modelo. El archivo de audio recibido se transforma en un buffer de bytes para ser procesado y, una vez transcrita, la respuesta se modela en formato JSON para ser recibida de manera transparente por Rhasspy.

3. Funcionamiento servicio Piper TTS

Al igual que se detalló anteriormente en la descripción de la solución propuesta, Piper TTS es un sistema de generación y síntesis de voz neuronal diseñado para generar audio hablado a partir de texto de forma local. En este proyecto, se utiliza como motor del servicio Text-to-Speech (TTS) para transformar las respuestas o feedback del asistente de voz en audio de alta calidad de manera eficiente para que el usuario reciba la información a través de los altavoces.

Los modelos descargables (*onnx*) y los ficheros configurables (*JSON*) permiten que el servicio se ejecute completamente de forma local en la Raspberry Pi sin conexión a la red. Además, está especialmente desarrollado para operar en dispositivos de bajo consumo y recursos limitados como este.

Una vez Rhasspy reconoce el intent en la orden del usuario y ejecuta el procesado y la lógica correspondiente, si está activado, se genera un mensaje de respuesta en forma de texto para ser reproducido al usuario. Este contenido se envía a Piper TTS en formato JSON para su conversión a audio. El archivo de audio resultante es devuelto a Rhasspy para ser reproducido a través de los altavoces conectados y terminando el ciclo de vida de una ejecución del asistente de voz.

En el Anexo II. "Piper_server.py" se incluye el script que crea y lanza el servidor que emplea la instalación local de Piper TTS, especificando el modelo de voz utilizado, idioma y otras configuraciones de Piper TTS necesarias para su correcto funcionamiento. En este caso se utiliza el modelo de voz en español *es_Es-davefx-medium*, por la relación entre calidad, naturalidad de la voz y corto tiempo de respuesta.

En el script se expone el servidor a través del puerto 4040 de la Raspberry Pi para poder ser ejecutado por Rhasspy de manera sencilla a través de la URL *http://localhost:4040/tts*. Para garantizar que el servidor se mantenga funcionando continuamente y esté disponible para las solicitudes del asistente, se ha definido un servicio local que lo mantiene ejecutado constantemente. La definición del script puede consultarse en el Anexo II. "Piper-tts.service".

En detalle, el funcionamiento del servidor *piper_server.py* consiste en recibir una cadena de texto, convertirla a audio por medio del modelo preentrenado seleccionado, a través de un comando por consola, y devolver un archivo de audio en formato WAV que Rhasspy reproduce directamente a través de los altavoces instalados.

4. Funcionamiento reconocimiento Intents

De una orden transcrita del audio del usuario, el servicio Intent Recognition analiza la presencia de alguno de Intents predefinidos y asociados a las acciones sobre elementos del HDA mediante comparación con el archivo *sentences.ini*. En este archivo de configuración de Rhasspy, detallado en el Anexo II. Sentences.ini, se encuentran definidos los distintos items y el conjunto de frases o sentencias, y sus múltiples posibles variaciones, asociadas a cada Intent y por ende a cada orden o acción que se puede realizar sobre los dispositivos monitorizados a través de OpenHAB.

En estas sentencias, vienen definidas una serie de parámetros o variables (ubicación, acción, etc) que serán recogidas por el Intent Recognition, junto con el nombre del Intent definido entre corchetes [] para poder procesar su lógica correctamente.

La interfaz web de Rhasspy ofrece la posibilidad de testear el reconocimiento de Intents desde la pantalla principal introduciendo ordenes manualmente como texto. En el cuadro "Text to recognize" se escribe la orden del usuario simulando que STT la ha transcrito. Al pulsar el botón *Recognize* se ejecutará el servicio Intent Recognition. Si la orden introducida se corresponde con alguna de las predefinidas en el *sentences.ini* para actuar sobre algún dispositivo, Rhasspy mostrará el nombre del Intent definido y el nombre y valor de las variables o parámetros definidos en esa sentencia.

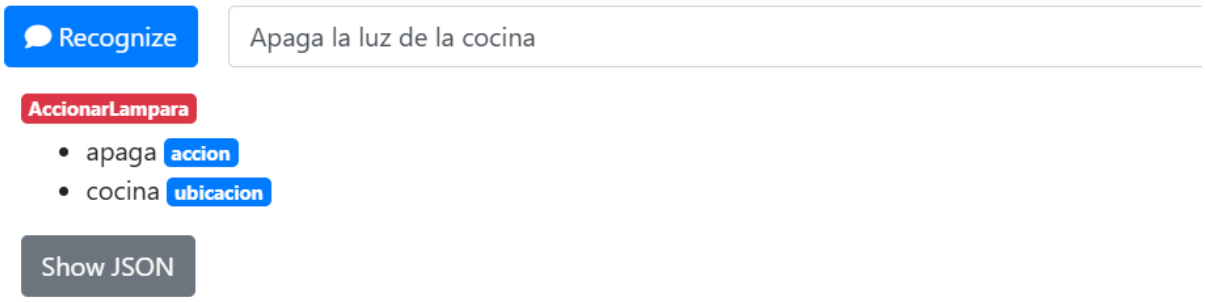

Figura 26. *Reconocimiento de Intents Manual*

Como respuesta o salida del servicio Intent Recognition, Rhasspy genera un JSON con la información recogida de la detección de Intent realizada, donde se incluye el nombre del Intent y las variables o parámetros reconocidos en la orden. Estas variables se agrupan dentro de un mismo objeto del JSON denominado "Slots". Pulsando el botón *Show JSON*, se muestra el contenido completo del archivo. Este JSON es lo que Rhasspy envía al servicio Intent Handling finalizado el reconocimiento de Intents para poder procesar la lógica.

Para poder simplificar y estandarizar el script *intent_handler_v1.py*, la información adicional de configuración de OpenHAB necesaria para el procesado se ha externalizada a través del JSON de configuración *configuracionOpenHAB.json*, cuyo contenido está descrito en el Anexo II. "ConfiguracionOpenHAB.json".

En él, en primer lugar, están definidos los nombre de los items de OpenHAB, necesarios para poder interactuar directamente con ellos, clasificados por tipo de dispositivo, ubicación en la que se encuentra o parámetro de medida en caso de los sensores.

También se ha incluido, para cada tipo de dispositivo, las órdenes definidas a enviar a OpenHAB para actuar sobre el dispositivo. Para ello, en todas las sentencias del *sentences.ini* se recoge mediante la variable "Accion" las distintas posibles palabras que indican el tipo de orden a ejecutar. Estas opciones se incluyen en el JSON, parseando cada una con la orden de OpenHAB correspondiente.

```
"acciones_generales":{
    "lamparas":{
        "encender": "ON",
        "enciende": "ON",
        "apagar": "OFF",
        "apaga": "OFF"
    },
```

Figura 27. *Parseo de Ordenes OpenHAB*

De la figura anterior, encender, enciende, apagar y apaga son las posibles palabras que se recogerán sí o sí en la variable "Accion" de las sentencias definidas para actuar sobre lámparas. Para cada opción, se asocia el comando o orden que se enviará al item de OpenHAB, ON o OFF en este caso.

Otro elemento estandarizado para simplificar el procesamiento e incluido en el JSON son los mensajes de respuesta que el asistente transformará en audio con Piper TTS para transmitirlo al usuario como feedback en las acciones u ordenes ejecutadas satisfactoriamente. Estos mensajes están ordenados en función del tipo de dispositivo y la acción enviada a OpenHAB sobre el item del mismo. Siguiendo el ejemplo de las lámparas:

```
"mensajes_respuesta": {
    "lamparas": {
        "ON": "Luz de {ubicacion} encendida",
        "OFF": "Luz de {ubicacion} apagada"
    },
```

Figura 28. *Mensajes de Respuesta por Item*

Como se observa en la figura, los mensajes están estandarizados, empleando la variable "{ubicacion}" en este caso, para hacerlos compatibles con cualquier instancia de lámparas del HDA. Con este formato, en el script de *intent_handler_v1.py* se reemplaza el valor de la ubicación por el valor proporcionado por los slots para esa variable de manera sencilla si necesidad de tratar de manera individual cada instancia de lámparas.

Finalmente, están asociadas también la función diseñada a ejecutar para cada uno de los Intents definidos permitiendo así reutilizar código de manera simple y sencilla.

```
"funciones": {
    "AccionarLampara": "accionarLampara",
    "CambiarColorLampara": "cambiarColorLampara",
    "CambiarTemperaturaLampara": "cambiarTemperaturaLampara",
    "AccionarPersiana": "accionarPersiana",
    "ConsultarPersiana": "consultarPersiana",
    "ConsultarTemperatura": "consultarParametro",
    "ConsultarHumedad": "consultarParametro",
    "ConsultarPresencia": "consultarParametro",
    "ConsultarInundacion": "consultarParametro",
    "ConsultarLuminosidad": "consultarParametro",
    "ConsultarGas": "consultarParametro",
    "HabilitarReglas": "habilitarReglas",
    "EjecutarReglas": "ejecutarReglas",
    "ActivarVoz": "activarVoz"
},
```

Figura 29. *Relación Intents-Funciones*

Por último, están incluidos el parseo a texto de las unidades de medida devueltas por OpenHAB en la consulta de algunos parámetros para incluirlas en las respuestas de voz. Y la variable "respuesta_voz_activa", ya mencionada en apartados anteriores, para disponer del estado del item MonitorizacionRespuestaVoz sin necesidad de realizar una consulta activa del item en cada ejecución.

```
"unidades_medida": {
    "ºC": "grados centígrados",
    "%": "por ciento",
    "lx": "lúmenes",
    "ppm": "partes por millón"
},
"respuesta_voz_activa": true
```

Figura 30. *Parseo a Texto de Unidades de Medida*

Este JSON es abierto y consultado por el script en cada ejecución.

Retomando el flujo de la ejecución, el servicio Intent Handling ejecuta únicamente el script *intent_handler_v1.py*. Este esta diseñado con la intención de particionar el código en la medida de lo posible a través de funciones para simplificar la ejecución y promover la reutilización de código. El contenido completo del mismo se encuentra en el Anexo II. "Intent_handler_v1.py".

En primer lugar, se lee y obtiene la información tanto del JSON generado por Intent Recognition como del *configuracionOpenHAB.json* para disponer de ella y se ejecuta la función general "*procesar_orden_rhasspy()*" que analizará la función correspondiente a ejecutar en base al Intent que se ha reconocido.

Como punto intermedio sea cual sea la función a ejecutar, una serie de funciones obtienen y recogen información necesaria para el procesamiento concreto de la función (nombre item, acción a enviar a OpenHAB…) del JSON de configuración. Finalmente, la función *"procesar_item"*, en base al tipo de dispositivo con el que interactuar y la acción a realizar se ejecuta la función correspondiente.

```
if item:
    if tipo_dispositivo == "voz":
        procesar_activacion_voz(item, tipo_dispositivo, accion, acciones_disponibles_dispositivo)
    elif medida:
        mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, "")
        mensaje_error = "No he podido obtener la información solicitada"
        procesar_consulta_parametro(item, medida, ubicacion, mensaje_exito, mensaje_error)
    elif tipo_dispositivo == "posicion_persianas":
        mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, {})
        mensaje_error = "No he podido obtener la información solicitada"
        procesar_consulta_persiana(item, ubicacion, mensaje_exito, mensaje_error)
    elif nombre_regla and accion in acciones_disponibles_dispositivo:
        mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, {}).get(acciones_disponibl
        mensaje_error = "No he podido confirmar que se haya habilitado o ejecutado la regla"
        procesar_envio_accion(item, acciones_disponibles_dispositivo[accion], mensaje_exito, mensaje_error)
    elif accion in acciones_disponibles_dispositivo:
        mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, {}).get(acciones_disponibl
        mensaje_error = "No he podido obtener el valor de la medida"
        procesar_envio_accion(item, acciones_disponibles_dispositivo[accion], mensaje_exito, mensaje_error)
else:
```

Figura 31. *Elección de Función en Base a Dispositivo o Acción*

Cabe destacar que las interacciones con OpenHAB se realizan siempre vía API Rest por sencillez y comodidad, tanto para ejecutar acciones como consultar parámetros.

Adicionalmente, existen dos funciones para complementar el funcionamiento del script. Por un lado, *"detectar_cambio_estado"* permite comprobar que, ante una acción que suponga un cambio de estado de un item, se verifique que se ha realizado correctamente en OpenHAB. Por otro lado, la función "*reproducir*" permite lanzar la reproducción del mensaje de respuesta ejecutando directamente el servicio TTS de la Raspberry Pi a través de la URL expuesta.

5. Adición de nuevos Intents

El asistente de voz está desarrollado con el objetivo de simplificar y facilitar la ampliación y escalabilidad del mismo para controlar y monitorizar cualquier otro dispositivo que se incorpore al HDA. Los pasos a seguir para configurar e implementar la lógica de un nuevo dispositivo, suponiendo que su integración se realiza a través de OpenHAB como plataforma de monitorización son:

En primer lugar, definir en el archivo de configuración *sentences.ini* el nombre del Intent a asociar con ese dispositivo así como las diversas sentencias que se quiera para reconocer dicho Intent en los comandos de voz del usuario. Recordar añadir y definir las variables correspondientes necesarias como ubicación, acción etc.

En segundo lugar, debe modificarse el JSON *configuracionOpenHAB* para incluir los datos y parámetros del nuevo dispositivo.

- Item: En caso de ser un tipo de dispositivo ya existente, simplemente añadir el nuevo objeto dentro del dispositivo asociando la ubicación donde se encuentra con el nombre del item en OpenHAB. En caso de ser un dispositivo nuevo, añadir los datos bajo el objeto "items".
- Acciones: De igual manera que con el item, deben añadirse, de no existir, las órdenes a enviar a OpenHAB asociadas a cada una de las posibles variables "acción" definidas en el *sentences.ini*.
- Mensaje respuesta: En relación con las acciones a lanzar sobre el item de OpenHAB, definir los mensajes de éxito para cada una de ellas para ser reproducidas por el altavoz del asistente de voz.
- Funciones: Por último, añadir el parseo entre el nombre del Intent definido en *sentences.ini* con el nombre de la función creada en el script para procesar la lógica de este dispositivo.

En caso de ser necesario, se pueden incluir en el JSON datos o parámetros asociados con el manejo del dispositivo con el fin de externalizarlos y simplificar su operación en el script.

Por último, queda modificar o adaptar el script *intent_handler_v1.py* para tratar los eventos del nuevo dispositivo. Dado el diseño modular y de reutilización de código, simplemente habrá que definir la nueva función básica para este dispositivo, ya añadida al JSON de configuración, con la misma estructura que las que se muestran en la siguiente figura:

```
#Funciones de cada accion o interacción
def accionarLampara(slots):
    return procesar_item("lamparas", slots, obtener_acciones("lamparas"))

def cambiarColorLampara(slots):
    return procesar_item("color_lamparas", slots, obtener_acciones("color_lamparas"))

def cambiarTemperaturaLampara(slots):
    return procesar_item("temperatura_lamparas", slots, obtener_acciones("temperatura_lamparas"))

def accionarPersiana(slots):
    return procesar_item("persianas", slots, obtener_acciones("persianas"))

def consultarPersiana(slots):
    return procesar_item("posicion_persianas", slots, obtener_acciones("posicion_persianas"))

def consultarParametro(slots,intent):
    return procesar_item("sensores", slots, obtener_acciones("sensores"), intent.replace("Consultar", "").lower())

def habilitarReglas(slots):
    return procesar_item("reglas_habilitar", slots, obtener_acciones("reglas_habilitar"))

def ejecutarReglas(slots):
    return procesar_item("reglas_ejecutar", slots, obtener_acciones("reglas_ejecutar"))

def activarVoz(slots):
    return procesar_item("voz", slots, obtener_acciones("voz"))
```

Figura 32. *Funciones Básicas para el Procesado de Cada Dispositivo*

Dentro de la función se llamará a *procesar_item*, incluyendo el tipo de dispositivo, los Slots de Intent Recognition y las acciones definidas para dicho dispositivo, para que actúe de manejador y ejecute la función general para el procesado de las acciones correspondientes.

De forma general, las acciones a ejecutar sobre un item de OpenHAB serán las ya definidas de *procesar_envio_accion*, para enviar un comando o acción concreta al item, o *procesar_consulta_parametro*, para obtener el dato de algún sensor.

En caso de precisar otra operativa sobre el item de OpenHAB, será necesario desarrollar la nueva lógica en una función y posteriormente anidarla dentro del bucle if-else de la función *procesar_item* mostrado en la Figura 28.

6. Funcionamiento servicio MonitorizacionRespuestaVoz

Como parte final del flujo de funcionamiento del asistente de voz, una vez procesada y ejecutada una orden del usuario, se genera una respuesta para ser procesada por Piper TTS y reproducida como audio a través de los altavoces.

Como se vio en apartados anteriores, esta parte del flujo se ejecuta siempre que el item de OpenHAB "*ActivadorRespuestaAsistenteVoz*" esté activo. Dado que consultar constantemente un item de OpenHAB introduce un retardo significativo en la ejecución, el valor de este item está parseado al objeto "*ActivarVoz*" del JSON *configuracionOpenHAB* para simplificar las consultas.

El servicio MonitorizacionRespuestaVoz se encarga precisamente de supervisar en todo momento el item de OpenHAB y actualizar y sobrescribir su estado en el JSON de configuración. De esta manera se asegura que, en el momento de ejecución del asistente, el objeto del JSON disponga siempre de la última información disponible actualizada del item. Esta integración está implementada como una solución ligera que permite monitorizar el estado del item en segundo plano de manera sencilla.

En el Anexo II. "MonitorizacionVoz_server.py" está descrito el contenido del script que ejecuta la comprobación del item. Para que la comprobación se realice de manera constante y se actualice en tiempo real el estado del item, se ha definido un servicio local que mantiene ejecutando el script constantemente. La definición de este servicio está disponible en el Anexo II. "MonitorizacionVoz.service".

Sobre el funcionamiento del script, éste se conecta a OpenHAB a través de un WebSocket al topic de eventos del item *ActivadorRespuestaAsistenteVoz* a través de la URL para detectar cualquier cambio de estado que se produzca. Al detectarse un cambio, abre el JSON *configuracionOpenHAB* y actualiza el estado del item.

Este componente no forma parte del núcleo de Rhasspy ni de los motores de los principales servicios, pero es un elemento fundamental para dotar de autonomía al usuario y hacer que la interacción por voz sea completa dando respuesta a las órdenes indicadas y aportando capacidad de diagnóstico ante fallos o errores.

Anexo II: Software y archivos de configuración

1. Intent_handler_v1.py

```
import sys
import json
import logging
import inspect
import requests
import time

ogging.basicConfig(
    level=logging.INFO,
    format = '%(asctime)s - %(levelname)s - %(message)s',
    datefmt = '%Y-%m-%d %H:%M:%S'
)

#Configuración General
CONFIG_JSON = "/profiles/es/rhasspy-files/configuracionOpenHAB.json"  #Ruta archivo JSON de
configuración OpenHAB
OPENHAB_URL = "http://192.168.0.100:8080" #Ajustar para cada ejecución
RHASSPY_URL = "http://192.168.0.189:12101/api/text-to-speech"
HEADERS = {'Content-Type': 'text/plain'}

#Variables Globales
RESPUESTA_VOZ_ACTIVA = True #Habilita la respuesta por voz del asistente

#Cargar archivo JSON
try:
    with open(CONFIG_JSON, "r", encoding="utf-8") as f:
        datos = json.load(f)
except Exception as e:
    logging.error(f"Error cargando el archivo de configuración: {e}")
    datos = {}

#Procesamiento de las órdenes de Rhasspy
def procesar_orden_rhasspy(intent, slots):
    global RESPUESTA_VOZ_ACTIVA
    #Se mapean los intents con sus funciones específicas. Se cargan los datos necesarios del
JSON primero.
    try:
        item = None
        RESPUESTA_VOZ_ACTIVA = datos.get("respuesta_voz_activa", True)
        funciones_config = datos.get("funciones", {})
        funciones = {k: globals().get(v) for k, v in funciones_config.items()}

        #Ejecución de la función correspondiente a cada Intent de Rhasspy
        if intent in funciones and funciones[intent]:
            funcion = funciones[intent]
            sig = inspect.signature(funcion)
```

```
            contador_parametros = len(sig.parameters)

            if contador_parametros == 2:
                item = funcion(slots, intent)
            elif contador_parametros == 1:
                item = funcion(slots)
        else:
            logging.warning(f"Intent no reconocido o definido: {intent}")

        #Comprobación item correcto o no
        if item is None:
            logging.warning(f"Parámetros incorrectos en el intent. Intent introducido:
{intent}")
        else:
            logging.info(f"Intent reconocido: {intent}. Item afectado: {item}")

    except Exception as e:
        logging.error(f"Error enviando la orden: {e}")

#############################################################################################
def procesar_item(tipo_dispositivo, slots, acciones_disponibles_dispositivo, medida=None):
    global RESPUESTA_VOZ_ACTIVA
    ubicacion = slots.get("ubicacion", "").lower()
    accion = slots.get("accion", "").lower()
    color = slots.get("color", "").lower()
    nombre_regla = slots.get("regla", "").lower()
    item = obtener_item(tipo_dispositivo, ubicacion, medida)
    logging.info(f"Item: {item}") #YA NO SE EJECUTA

    if item:
        if tipo_dispositivo == "voz":
            procesar_activacion_voz(item, tipo_dispositivo, accion,
acciones_disponibles_dispositivo)
        elif medida:
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, "")
            mensaje_error = "No he podido obtener la información solicitada"
            procesar_consulta_parametro(item, medida, ubicacion, mensaje_exito,
mensaje_error)
        elif tipo_dispositivo == "posicion_persianas":
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo, {})
            mensaje_error = "No he podido obtener la información solicitada"
            procesar_consulta_persiana(item, ubicacion, mensaje_exito, mensaje_error)
        elif nombre_regla and accion in acciones_disponibles_dispositivo:
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo,
{}).get(acciones_disponibles_dispositivo[accion], "").format(nombre_regla=nombre_regla)
            mensaje_error = "No he podido confirmar que se haya habilitado o ejecutado la
regla"
            procesar_envio_accion(item, acciones_disponibles_dispositivo[accion],
mensaje_exito, mensaje_error)
        elif accion in acciones_disponibles_dispositivo:
```

```
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo,
{}).get(acciones_disponibles_dispositivo[accion], "").format(ubicacion=ubicacion,
color=color)
            mensaje_error = "No he podido obtener el valor de la medida"
            procesar_envio_accion(item, acciones_disponibles_dispositivo[accion],
mensaje_exito, mensaje_error)
    else:
        logging.info(f"Item {item} no existente o no configurado")
    return item

def obtener_acciones(tipo_dispositivo):
    return datos.get("acciones_generales", {}).get(tipo_dispositivo, {})

def obtener_item(tipo_dispositivo, ubicacion, medida=None):
    tipo_item = datos.get("items", {}).get(tipo_dispositivo, {})
    if isinstance(tipo_item, dict):
        if medida:
            return tipo_item.get(ubicacion, {}).get(medida)
        return tipo_item.get(ubicacion)
    else:
        return tipo_item

#Funciones de cada accion o interacción
def accionarLampara(slots):
    return procesar_item("lamparas", slots, obtener_acciones("lamparas"))

def cambiarColorLampara(slots):
    return procesar_item("color_lamparas", slots, obtener_acciones("color_lamparas"))

def cambiarTemperaturaLampara(slots):
    return procesar_item("temperatura_lamparas", slots,
obtener_acciones("temperatura_lamparas"))

def accionarPersianaSwitch(slots):
    return procesar_item("persianas_switch", slots, obtener_acciones("persianas_switch"))

def accionarPersianaShutter(slots):
    return procesar_item("persianas_shutter", slots, obtener_acciones("persianas_shutter"))

def consultarPersiana(slots):
    return procesar_item("posicion_persianas", slots,
obtener_acciones("posicion_persianas"))

def consultarParametro(slots,intent):
    return procesar_item("sensores", slots, obtener_acciones("sensores"),
intent.replace("Consultar", "").lower())

def habilitarReglas(slots):
    return procesar_item("reglas_habilitar", slots, obtener_acciones("reglas_habilitar"))
```

```
def ejecutarReglas(slots):
    return procesar_item("reglas_ejecutar", slots, obtener_acciones("reglas_ejecutar"))

def activarVoz(slots):
    return procesar_item("voz", slots, obtener_acciones("voz"))

#FUNCIONES AUXILIARES
def procesar_activacion_voz(item, tipo_dispositivo, accion,
acciones_disponibles_dispositivo):
    global RESPUESTA_VOZ_ACTIVA
    if acciones_disponibles_dispositivo[accion] == "ON":
        if not RESPUESTA_VOZ_ACTIVA:
            RESPUESTA_VOZ_ACTIVA = True
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo,
{}).get(acciones_disponibles_dispositivo[accion], "")
            mensaje_error = "No he podido confirmar si se ha activado la respuesta de voz en
OpenHAB"
            procesar_envio_accion(item, acciones_disponibles_dispositivo[accion],
mensaje_exito, mensaje_error)
        else:
            reproducir("La respuesta de voz ya estaba activada")
            logging.info("Respuesta de voz ya activada")
    elif acciones_disponibles_dispositivo[accion] == "OFF":
        if RESPUESTA_VOZ_ACTIVA:
            mensaje_exito = datos.get("mensajes_respuesta", {}).get(tipo_dispositivo,
{}).get(acciones_disponibles_dispositivo[accion], "")
            mensaje_error = "No he podido confirmar si se ha desactivado la respuesta de voz
en OpenHAB"
            procesar_envio_accion(item, acciones_disponibles_dispositivo[accion],
mensaje_exito, mensaje_error)
            RESPUESTA_VOZ_ACTIVA = False
        else:
            logging.info("Repuesta de voz ya desactivada")

def procesar_consulta_persiana(item, ubicacion, mensaje_exito, mensaje_error):
    posicion = obtener_parametro(item)
    if posicion is not None:
        if posicion > 85:
            mensaje_exito = mensaje_exito.get("abierta", "")
        elif posicion < 15:
            mensaje_exito = mensaje_exito.get("cerrada", "")
        else:
            mensaje_exito = mensaje_exito.get("resto", "")
        reproducir(mensaje_exito.format(ubicacion=ubicacion))
    else:
        reproducir(mensaje_error)
        logging.info(f"Posicion persiana en {ubicacion}: No se ha podido medir")

def procesar_consulta_parametro(item, medida, ubicacion, mensaje_exito, mensaje_error):
    valor = obtener_parametro(item)
```

```
    if valor is not None:
        valor_str = str(valor)
        simbolo = valor_str[-2:].lower() if len(valor_str) > 1 else valor_str[-1].lower()
        unidad = datos.get("unidades_medida", {}).get(simbolo, simbolo)
        reproducir(mensaje_exito.format(medida=medida, ubicacion=ubicacion,
valor_str=valor_str, unidad=unidad))
        logging.info(f"{medida.capitalize()} en {ubicacion}: Correctamente medida")
    else:
        reproducir(mensaje_error)
        logging.info(f"{medida.capitalize()} en {ubicacion}: No se ha podido medir")

def obtener_parametro(item):
    url = f"{OPENHAB_URL}/rest/items/{item}/state"
    try:
        response = requests.get(url, headers=HEADERS)
        if response.status_code == 200:
            logging.info(f"Valor de {item} obtenido correctamente")
            return response.text
        else:
            logging.error(f"Error obteniendo valor de {item}: {response.status_code} -
{response.text}")
            return None
    except requests.exceptions.RequestException as e:
        logging.error(f"Error en la conexión con OpenHAB: {e}")
        return None

def procesar_envio_accion(item, accion, mensaje_exito, mensaje_error):
    if enviar_orden(item, accion):
        #Obtener confirmación cambio de estado del item
        if detectar_cambio_estado(item, accion, timeout=5):
            reproducir(mensaje_exito)
            logging.info(f"Orden {accion} enviada a {item} satisfactoriamente")
        else:
            reproducir(mensaje_error)
            logging.info(f"Orden {accion} enviada a {item}. No se ha obtenido confirmación
de OpenHAB")
    else:
        logging.info(f"Orden no enviada")

#Función auxiliar para enviar las órdenes de un item de OpenHAB.
def enviar_orden(item, orden):
    url = f"{OPENHAB_URL}/rest/items/{item}"
    enviada = False
    try:
        response = requests.post(url, data=orden, headers=HEADERS)
        if response.status_code == 200:
            enviada = True
            logging.info(f"Comando {orden} enviado a {item}")
        else:
```

```
            logging.info(f"Error enviado comando {orden} a {item}: {response.status_code} -
{response.text}")
        return enviada
    except requests.exceptions.RequestException as e:
        logging.error(f"Error en la conexión con OpenHAB: {e}")

#Función auxiliar para detectar que se han realizado los cambios de estado de los items vía
API REST
def detectar_cambio_estado(item, estado_esperado, timeout=5):
    url = f"{OPENHAB_URL}/rest/items/{item}/state"
    #Tiempo de inicio para el timeout
    start_time = time.time()
    while time.time() - start_time < timeout:
        try:
            response = requests.get(url, headers=HEADERS)
            if response.status_code == 200:
                estado_actual = response.text.strip()
                if estado_actual.upper() == estado_esperado.upper():
                    logging.info(f"Estado cambiado. {item} cambió a {estado_actual}")
                    return True
            else:
                logging.error(f"Error obteniendo el estado de {item}: {response.status_code}
- {response.text}")
        except requests.exceptions.RequestException as e:
            logging.error(f"Error en la conexión con OpenHAB: {e}")
        time.sleep(1)
    logging.error(f"No se pudo confirmar el cambio de estado de {item} a {estado_esperado}")
    return False

#Función auxiliar para el envío de las respuesta de las órdenes a Rhasspy para reproducirlas
por voz
def reproducir(texto):
    if not RESPUESTA_VOZ_ACTIVA:
        logging.info(f"Reproducción de voz desactivada. Mensaje omitido: {texto}")
        return
    try:
        response = requests.post(RHASSPY_URL, headers=HEADERS, data=texto.encode('utf-8'))
        if response.status_code == 200:
            logging.info(f"TTS: '{texto}' enviado a Rhasspy para su reproducción")
        else:
            logging.info(f"Error en el procesamiento TTS: {response.status_code} -
{response.text}")
    except requests.exceptions.RequestException as e:
        logging.error(f"Error en la conexión con Rhasspy: {e}")

##############################################################################################
#Función principal para leer el intent y los slots pasados por Rhasspy por la entrada
estándar
if __name__ == "__main__":
    try:
```

```
        rhasspy_data = json.loads(sys.stdin.read())     #Cargar el JSON de Rhasspy
        intent = rhasspy_data.get("intent", {}).get("name", "")
        slots = rhasspy_data.get("slots", {})
        procesar_orden_rhasspy(intent,slots)

    except json.JSONDecodeError:
        logging.error("Error: El formato JSON es inválido.")
    except Exception as e:
        logging.error(f"Error procesando la entrada estándar: {e}")
```

2. configuracionOpenHAB.json


```
{
    "items": {
        "lamparas": {
            "baño": "Interruptor_Aplique_Bano",
            "cocina": "Interruptor_LamparaMesa_Cocina",
            "encimera": "Interruptor_LuzEncimera_Cocina",
            "salon": "Interruptor_Aplique_Salon",
            "pie": "Interruptor_LamparaPie1_Salon",
            "dormitorio": "Interruptor_LamparaMesa1_Dormitorio"
        },
        "color_lamparas": {
            "salon": "Color_Aplique_Salon",
            "cocina": "Color_LuzEncimera_Cocina"
        },
        "temperatura_lamparas": {
            "salon": "CTemperatura_Aplique_Salon"
        },
        "persianas_switch": {
            "entrada": "Interruptor_Persiana_Entrada"
        },
        "persianas_shutter": {
            "dormitorio": "Interruptor_Persiana_Dormitorio",
            "salon": "Interruptor_Persiana_Salon",
        },
        "posicion_persianas": {
            "dormitorio": "Posicion_Persiana_Dormitorio",
            "salon": "Posicion_Persiana_Salon"
        },
        "sensores": {
            "cocina": {
                "temperatura": "Temperatura_MultiSensorAmbiental_Cocina",
                "humedad": "Humedad_MultiSensorAmbiental_Cocina",
                "gas": "Gas_MultiSensorAmbiental_Cocina"
            },
            "salon": {
                "temperatura": "Temperatura_MultiSensorAmbiental_Salon",
                "humedad": "Humedad_MultiSensorAmbiental_Salon"
            },
            "dormitorio": {
                "temperatura": "Temperatura_MultiSensorAmbientalyPresencia_Dormitorio",
                "humedad": "Humedad_MultiSensorAmbientalyPresencia_Dormitorio",
                "luminosidad": "Luminosidad_MultiSensorAmbientalyPresencia_Dormitorio",
                "presencia": "Presencia_MultiSensorAmbientalyPresencia_Dormitorio"
            },
            "baño": {
                "temperatura": "Temperatura_MultiSensorAmbientalyPresencia_Bano",
                "humedad": "Humedad_MultiSensorAmbientalyPresencia_Bano",
                "presencia": "Presencia_MultiSensorAmbientalyPresencia_Bano",
```

```
            "inundacion": "Inundacion_MultiSensorAmbientalyPresencia_Bano"
        }
    },
    "reglas_habilitar": {
        "amanecer": "ActivadorRule_Amanecer",
        "anochecer": "ActivadorRule_Anochecer",
        "apagar todo": "ActivadorRule_ApagarTodo",
        "control luz baño": "ActivadorRule_ControlLuzBano",
        "general": "ActivadorRule_GLOBAL"
    },
    "reglas_ejecutar": {
        "amanecer": "EjecutaRule_Amanecer",
        "anochecer": "EjecutaRule_Anochecer",
        "apagar todo": "EjecutaRule_ApagarTodo",
        "control luz baño": "EjecutaRule_ControlLuzBano"
    },
    "voz": "ActivadorRespuestaAsistenteVoz"
},
"acciones_generales":{
    "lamparas":{
        "encender": "ON",
        "enciende": "ON",
        "apagar": "OFF",
        "apaga": "OFF"
    },
    "color_lamparas":{
        "rojo": "255,0,0",
        "verde": "0,255,0",
        "azul": "0,255,0",
        "amarillo": "255,255,0",
        "naranja": "255,165,0",
        "blanco": "255,255,255"
    },
    "temperatura_lamparas":{
        "calida": "20",
        "calido": "20",
        "neutra": "50"
        "fria": "80",
        "frio": "80"
    },
    "persianas_switch":{
        "sube": "ON",
        "subir": "ON",
        "levanta": "ON",
        "levantar": "ON",
        "abre": "ON",
        "abrir": "ON",
        "baja": "OFF",
        "bajar": "OFF",
        "cierra": "EFF",
```

```
            "cerrar": "EDA",
        },
        "persianas_shutter":{
            "sube": "UP",
            "subir": "UP",
            "levanta": "UP",
            "levantar": "UP",
            "abre": "UP",
            "abrir": "UP",
            "baja": "DOWN",
            "bajar": "DOWN",
            "cierra": "DOWN",
            "cerrar": "DOWN",
            "para": "STOP",
            "parar": "STOP",
            "deten": "STOP",
            "detener": "STOP"
        },

        "posicion_persianas": {

        },
        "sensores":{

        },
        "reglas_habilitar":{
            "habilita": "ON",
            "habilitar": "ON",
            "deshabilita": "OFF",
            "deshabilitar": "OFF"
        },
        "reglas_ejecutar":{
            "ejecuta": "ON",
            "ejecutar": "ON",
            "lanza": "ON",
            "lanzar": "ON"
        },
        "voz": {
            "habilita": "ON",
            "habilitar": "ON",
            "activa": "ON",
            "activar": "ON",
            "deshabilita": "OFF",
            "deshabilitar": "OFF",
            "desactiva": "OFF",
            "desactivar": "OFF"
        }
    },
    "mensajes_respuesta": {
        "lamparas": {
```

```
            "ON": "Luz de {ubicacion} encendida",
            "OFF": "Luz de {ubicacion} apagada"
        },
        "color_lamparas": {
            "rojo": "Lámpara {ubicacion} cambiada a color {color}",
            "azul": "Lámpara {ubicacion} cambiada a color {color}",
            "amarillo": "Lámpara {ubicacion} cambiada a color {color}",
            "verde": "Lámpara {ubicacion} cambiada a color {color}",
            "naranja":"Lámpara {ubicacion} cambiada a color {color}",
            "blanco":"Lámpara {ubicacion} cambiada a color {color}"
        },
        "temperatura_lamparas": {
            "HIGH": "Temperatura de la lámpara de {ubicacion} más cálida",
            "DOWN": "Temperatura de la lámpara de {ubicacion} más fría"
        },
        "persianas": {
            "UP": "Persiana {ubicación} levantada",
            "DOWN": "Persiana {ubicacion} bajada",
            "STOP": "Persiana {ubicacion} parada"
        },
        "posicion_persianas": {
            "100": "Persiana de {ubicacion} cerrada",
            "0": "Persiana de {ubicacion} abierta",
            "resto": "Persiana de {ubicacion} semiabierta"
        },
        "sensores": "{medida.capitalize()} en {ubicacion} de {valor_str} {unidad}",
        "reglas_habilitar": {
            "ON": "Regla {nombre_regla} habilitada",
            "OFF": "Regla {nombre_regla} deshabilitada"
        },
        "reglas_activar": {
            "ON": "Regla {nombre_regla} ejecutada"
        },
        "voz":{
            "ON": "Respuesta de voz activada. Hola de nuevo",
            "OFF": "Respuesta de voz desactivada. Hasta la próxima"
        }
    },
    "funciones": {
        "AccionarLampara": "accionarLampara",
        "CambiarColorLampara": "cambiarColorLampara",
        "CambiarTemperaturaLampara": "cambiarTemperaturaLampara",
        "AccionarPersiana": "accionarPersiana",
        "ConsultarPersiana": "consultarPersiana",
        "ConsultarTemperatura": "consultarParametro",
        "ConsultarHumedad": "consultarParametro",
        "ConsultarPresencia": "consultarParametro",
        "ConsultarInundacion": "consultarParametro",
        "ConsultarLuminosidad": "consultarParametro",
        "ConsultarGas": "consultarParametro",
```

```
        "HabilitarReglas": "habilitarReglas",
        "EjecutarReglas": "ejecutarReglas",
        "ActivarVoz": "activarVoz"
    },
    "unidades_medida": {
        "°C": "grados centígrados",
        "%": "por ciento",
        "lx": "lúmenes",
        "ppm": "partes por millón"
    },
    "respuesta_voz_activa": true
}
```

3. whisper_server.py

```
from flask import Flask, request, jsonify, Response
from faster_whisper import WhisperModel
import wave
import io
import json

# Configuración
MODEL_PATH = "/home/pi/whisper-models/whisper-base"  # Modelo whisper
model = WhisperModel(MODEL_PATH, device="cpu", compute_type="int8")  # INT8 reduce uso
CPU/RAM

# Crear servidor Flask
app = Flask(__name__)

@app.route("/stt", methods=["POST"])
def stt():
    if not request.data:
        return jsonify({"error": "No se recibio audio"}), 400

    try:
        # Convertir datos en un buffer de bytes
        audio_buffer = io.BytesIO(request.data)
        segments, _ = model.transcribe(audio_buffer, language="es")

        # Construir la transcripción y eliminar el punto final para el procesamiento en
Rhasspy
        transcription_text = " ".join([segment.text for segment in
segments]).strip().lower()
        if transcription_text.endswith("."):
            transcription_text = transcription_text[:-1]

        #Construir la respuesta
        response_data = json.dumps(
            {"text": transcription_text},
            ensure_ascii=False,
            indent=2
        )
        return Response(response_data, mimetype="application/json; charset=utf-8")

    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5050, debug=True)
```

4. sentences.ini

```
[AccionarLampara]
(enciende | activa | apaga | desactiva){accion} la (luz | lámpara) (del | de la| de)
$ubicaciones_lamparas{ubicacion}

[CambiarColorLampara]
(cambia | pon) la (luz | lámpara | bombilla) (del | de la)
$ubicaciones_lamparas_color{ubicacion} [(de | al)] color $colores_lampara{color}

[CambiarTemperaturaLampara]
pon la (luz | lámpara) (del | de la) $ubicaciones_lamparas_temperatura{ubicacion}
(más | menos){diferencia} (cálida | fría){temperatura}

[AccionarPersiana]
(sube | baja | para | abre | cierra | levanta | detén){accion} la persiana (del | de
la) $ubicaciones_persiana{ubicacion}

[ConsultarPersiana]
(cómo | como) está la persiana del $ubicaciones_persiana{ubicacion}
(está | esta) (subida | bajada) la persiana del $ubicaciones_persiana{ubicacion}

[ConsultarPosicionPersiana]
en qué posición está la persiana del $ubicaciones_persiana{ubicacion}
qué posición tiene la persiana del $ubicaciones_persiana{ubicacion}

[ConsultarTemperatura]
(qué | que) temperatura hace en (el | la) $ubicaciones_temperatura{ubicacion}
(cuánto | cuanto) (frío | calor) hace en (el | la)
$ubicaciones_temperatura{ubicacion}

[ConsultarHumedad]
(cuánta | cuanta | qué | que) humedad (hace | hay) en el
$ubicaciones_humedad{ubicacion}
hay mucha humedad en el $ubicaciones_humedad{ubicacion}

[ConsultarPresencia]
[dime si] (hay | está | esta) alguien en el $ubicaciones_presencia{ubicacion}

[ConsultarInundacion]
[dime si] (está | esta) el $ubicaciones_inundacion{ubicacion} inundado

[ConsultarLumninosidad]
(cuánta | cuanta | qué | que) luz (hace | hay) en el
$ubicaciones_luminosidad{ubicación}

[ConsultarGas]
(cuál es | dime | en cuanto está | a cuanto está) el nivel de gas en la
$ubicaciones_gas{ubicacion}
(hay gas | cuando gas hay) en la $ubicaciones_gas{ubicacion}
```

```
[HabilitarReglas]
(habilita |hHabilitar | deshabilita | deshabilitar){accion} [la] regla [de]
(amanecer | anochecer | apagar todo | control luz baño){regla}

[EjecutarReglas]
(ejecuta | ejecutar | lanza | lanzar) [la] regla [de] (amanecer | anochecer | apagar
todo | control luz baño){regla}

[ActivarVoz]
(habilita | habilitar | activa | activar | deshabilita | deshabilitar | desactiva |
desactivar){accion} [la] respuesta (de | por) voz
```

5. piper-tts.service

```
[Unit]
Description=Piper TTS Service
After=network.target

[Service]
User=pi
ExecStart=/home/pi/piper-venv/bin/python /home/pi/piper_server.py
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

6. piper_server.py

```
from flask import Flask, request, send_file
import subprocess
import os

app = Flask(__name__)

MODEL_PATH = "/home/pi/piper-models/es_ES-davefx-medium/es_ES-davefx-medium.onnx"
CONFIG_PATH = "/home/pi/piper-models/es_ES-davefx-medium/es_ES-davefx-medium.json"
OUTPUT_WAV = "/home/pi/piper-audio-files/output.wav"
FIXED_WAV = "/home/pi/piper-audio-files/output_fixed.wav"

@app.route("/tts", methods=["POST"])
def tts():
    text = request.get_data(as_text=True).strip()
    if not text:
        return {"error": "No text provided"}, 400

    try:
        command = [
            "/home/pi/piper-venv/bin/piper",
            "-m", MODEL_PATH,
            "-c", CONFIG_PATH,
            "-f", OUTPUT_WAV
        ]

        # Ejecutar Piper y generar el archivo de audio
        process = subprocess.run(command, input=text.encode(), capture_output=True)

        if process.returncode != 0:
            return {"error": "Piper execution failed", "details": process.stderr.decode()}, 500

        convert_command = ["sox", OUTPUT_WAV, "-r", "48000", FIXED_WAV]
        convert_process = subprocess.run(convert_command, capture_output=True)

        if convert_process.returncode != 0:
            return {"error": "Failed to convert audio", "details":
convert_process.stderr.decode()}, 500

        return send_file(FIXED_WAV , mimetype="audio/wav")

    except Exception as e:
        return {"error": str(e)}, 500

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=4040, debug=True)
```

7. whisper-stt.service

```
[Unit]
Description = Whisper STT Server
After=network.target

[Service]
User=pi
ExecStart=/home/i/whisper-venv/bin/python /home/pi/whisper_server.py
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

8. monitorizacionVoz_server.py

```
import json
import websocket
import logging
import threading
import time
import os

logging.basicConfig(
    level=logging.INFO,
    format = "%(asctime)s - %(levelname)s - %(message)s",
    datefmt = "%Y-%m-%d %H:%M:%S"
)

#Configuración General
CONFIG_JSON = "home/pi/.config/rhasspy/profiles/es/configuracionOpenHAB.json"
OPENHAB_URL = "ws://192.168.1.100:8080" #Ajustar para cada ejecución
TOKEN_API =
"oh.tokenMonitorizacionVoz.SRq9Ulyi14D2FQtuWheZyDmqVg7KjS6iTsiPfYmsSYc80lC1jT7QO7XPvG4aqI69
BFjBdJI7gUciwKu1ikwYjQ"
ITEM_VOZ = "ActivadorRespuestaAsistenteVoz"

#Leer y modificar el estado de la respuesta de voz en el JSON
def escribirEstadoVoz(estado):
    try:
        with open(CONFIG_JSON, 'r', encoding='utf-8') as f:
            file = json.load(f)
        file["respuesta_voz_activa"] = estado
        with open(CONFIG_JSON, "w", encoding='utf-8') as f:
            json.dump(file, f, indent=4, ensure_ascii=False)

    except json.JSONDecodeError:
        logging.error("Error al leer el JSON. Puede haberse corrompido el archivo")
    except IOError as e:
        logging.error(f"Error al intentar acceder al archivo {CONFIG_JSON}: {e}")
    except Exception as e:
        logging.error(f"Error inesperado al escribir en el JSON: {e}")

#Hearbeat para mantener la conexión abierta
def send_heartbeat(ws):
    while True:
        heartbeat_msg = {"type": "WebSocketEvent", "topic": "openhab/websocket/heartbeat",
"payload": "PING", "source": "WebSocketTestInstance"}
        ws.send(json.dumps(heartbeat_msg))
        logging.info("Enviado mensaje de heartbeat")
        time.sleep(5)

#Manejo de los mensajes entrantes vía webSocket
def on_message(ws, message):
```

```
    try:
        data = json.loads(message)
        if data.get("type") == "ItemStateEvent" and
data.get("topic").endswith(f"{ITEM_VOZ}/state"):
            payload = json.loads(data["payload"])
            estado = payload.get("vaue", "").lower() == "on"
            escribirEstadoVoz(estado)
            logging.info(f"Estado de voz actualizado: {'ON' if estado else 'OFF'}")
        else:
            logging.info("No he podido recoger los datos de la respuesta webSocket de
OpenHAB")
    except json.JSONDecodeError:
        logging.error("Error al decodificar el mensjae WebSocket")

#Manejo error de conexión del webSocket
def on_error(ws, error):
    logging.error(f"Error WebSocket: {error}")

#Manejo cierre conexión webSocket
def on_close(ws, close_status_code, close_msg):
    logging.warning("Conexión WebSocket cerrada. Reintentando...")

#Manejo apertura conexión webSocket
def on_open(ws):
    subscription_msg = {"type": "subscribe","topic": f"openhab/items/{ITEM_VOZ}/state"}
    ws.send(json.dumps(subscription_msg))
    logging.info(f"Suscrito a {ITEM_VOZ}")

if __name__ == "__main__":
    if not os.path.exists(CONFIG_JSON):
        logging.error(f"El archivo {CONFIG_JSON} no existe o no se encuentra. Terminando
ejecución")
        exit(1)

    ws = websocket.WebSocketApp(f"{OPENHAB_URL}", on_message=on_message, on_error=on_error,
on_close=on_close)
    ws.on_open = on_open

    #Lanzar thread paralelo para el envio del heartbeat cada 5 segundos.
    heartbeat_thread = threading.Thread(target=send_heartbeat, args=(ws,))
    heartbeat_thread.daemon = True
    heartbeat_thread.start()
    #Iniciar websocket
    ws.run_forever(time_interval=5)
```