Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

ÁREA DE INGENIERÍA TELEMÁTICA

TRABAJO FIN DE MÁSTER

CONTROL DE UN HOGAR DOMOTIZADO A TRAVÉS DE ASTERISK

D. CASTRO GONZÁLEZ, Dailenis
TUTOR: D. NUÑO HUERGO, Pelayo

FECHA: Julio 2022

AGRADECIMIENTOS

A **Dios**, por ser la fortaleza de mi vida, porque es bueno y su misericordia es eterna, sin Él nada somos.

A mi **esposo**, por su amor, su paciencia, su apoyo incondicional y sobre todo porque ha tenido que soportar todos mis cambios de humor.

A mis **padres**, por ser tan buenos conmigo, por ese amor tan bonito que solo ellos saben dar, por sus regaños, por sus consejos, su preocupación. ¡Por todo!

A mis tres **hermanas** por tanto cariño, por estar al pendiente de mí aunque estemos lejos, y a mi **sobrina** Valeria por transmitirme tanta alegría.

A mis **suegros**, a toda mi **familia**, **amigos**, **compañeros** de clases por la ayuda que me brindaron en el momento que la necesité.

A los **profesores** que contribuyeron en mi formación durante estos años y en especial a mi **tutor** por guiarme, corregirme y estar siempre disponible para mí.

A todos, MUCHAS GRACIAS por formar parte de este largo proceso.

RESUMEN

Este trabajo propone una alternativa para el control domótico utilizando tecnologías de software libre como Asterisk. El sistema permite a un usuario ejercer el control por voz de sus dispositivos electrodomésticos mediante una llamada a la centralita de Asterisk. Para ello, se ha utilizado una maqueta comercial de *Internet of Things* (IoT) basada en la plataforma de Arduino la cual permite crear un escenario que simula un *Smart Home*. En este trabajo también intervienen otras tecnologías como la API *Speech to Text* (STT) de Google *Cloud*, Apache, Telegram, entre otras.

El prototipo propuesto es una solución sencilla y de bajo coste que puede servir de base para la búsqueda de alternativas tecnológicas que aporten comodidad, seguridad y ahorros en los hogares de los usuarios del sistema.

Palabras claves: Asterisk; *Smart Home*; *Internet of Things* (IoT); *Speech to Text* (STT).

ABSTRACT

This project proposes an alternative for home automation control using free software technologies such as Asterisk. The system allows users to perform voice control of their home appliances through a call to the Asterisk PBX. For this purpose, a commercial IoT mockup based on the Arduino platform has been used to create a scenario that simulates a Smart Home. This project also involves other technologies such as the Google Cloud STT API, Apache, Telegram, among others.

The proposed prototype is a simple and low-cost solution that can serve as a basis for the search for technological alternatives that provide comfort, security and savings in the homes of users of the system.

Key Words: Asterisk; Smart Home; Internet of Things (IoT); Speech to Text (STT).

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.- INTRODUCCIÓN

El Internet de las cosas (IoT) está experimentando un crecimiento vertiginoso en los últimos años debido a los continuos avances tecnológicos. Estos avances incluyen la ampliación de la conectividad a Internet, junto con el abaratamiento de los sensores, procesadores y el propio acceso a Internet. También han impulsado este auge los avances producidos en las comunicaciones inalámbricas y en protocolos de comunicación como IPv6. Básicamente el propósito de IoT es transformar objetos tradicionales (pasivos) en inteligentes (activos) y que a su vez se encuentren interconectados.

Dentro de la amplia gama de aplicaciones de IoT, la domótica es una de sus principales tendencias, surgiendo el término conocido como *Smart H*ome. Cuando se habla de un hogar inteligente, se refiere a la utilización de la domótica para generar una interacción dinámica entre el hogar y el usuario, tratando de automatizar y monitorizar diversos aspectos de una vivienda como la seguridad doméstica, el encendido y apagado de luces, de aparatos electrónicos, la calefacción, la climatización, etc. La implementación de la domótica proporciona mayor confort y seguridad de los habitantes, así como ahorros en los consumos de energía [1].

Debido a la tendencia de IoT en el ámbito doméstico surge la necesidad de crear prototipos que potencien e impulsen el desarrollo de soluciones en este sector, ya que sin lugar a duda va a formar parte de la vida cotidiana de cada individuo en los próximos años. Para el futuro de la domótica en el hogar los aspectos claves son el uso de asistentes de voz y además aprovisionar de inteligencia a la mayor cantidad de dispositivos del entorno [2]. Actualmente, ya existen soluciones comerciales que integran el reconocimiento por voz para llevar a cabo el control domótico del hogar. Sin embargo, los altos costes de adquisición pueden dificultar el uso extendido de estas herramientas.

En este contexto, el presente Trabajo de Fin de Máster (TFM) propone un prototipo de hogar domotizado controlado por comandos de voz, donde se integran tecnologías de software y hardware libres como Asterisk y Arduino, lo que permite reducir los costes. En este sentido, se utiliza Asterisk como la pasarela de comunicación entre las órdenes de voz dadas por el usuario y el entorno doméstico, mientras que la plataforma de Arduino interviene como el elemento controlador de los dispositivos finales en el hogar.

En base a lo anterior, un usuario puede acceder al control de sus electrodomésticos marcando a una extensión de la centralita de Asterisk desde su *softphone* (teléfono móvil, *tablet*, ordenador). Una vez se efectúa la llamada empieza la interacción con el servicio a través de un IVR (*Interactive Voice Response*) donde el usuario puede ordenar a viva voz la acción que desea ejecutar sobre sus equipos, bien sea de forma inmediata o planificada en una fecha y hora futuras. Para posibilitar dicha interacción, Asterisk se comunica con la API (*Application Programming Interface*) STT de Google *Cloud* para traducir los comandos de voz a texto. Según el texto recibido, se compara con una serie de frases predefinidas y si se encuentra alguna coincidencia se envía la opción especificada por el

usuario al hogar domotizado. Dicho hogar se ha simulado con el uso de una maqueta comercial basada en Arduino, la cual integra múltiples sensores y actuadores que representan las distintas funcionalidades de un entorno doméstico convencional. También es posible que el hogar envíe notificaciones al usuario a través de Telegram o tome decisiones automáticas en función de umbrales establecidos.

En la Figura 1.1 se muestra la arquitectura en bloques del prototipo, la cual permite tener una visión global del proyecto.

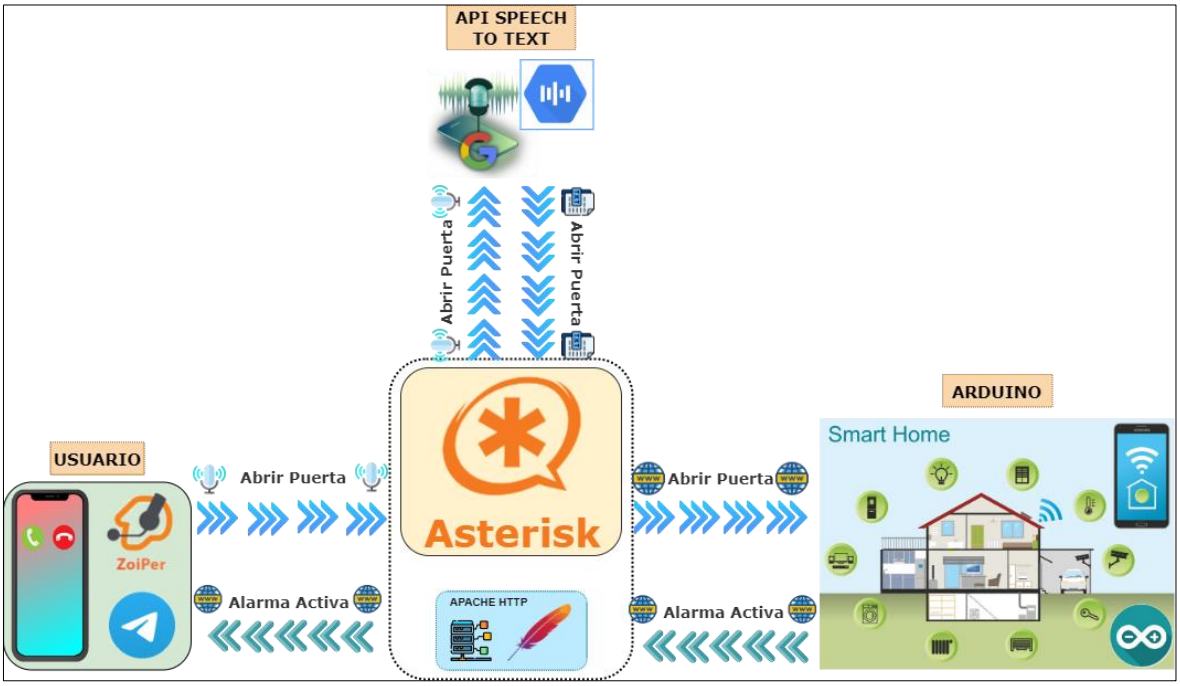


Figura 1.1.-Arquitectura en bloques del prototipo.

1.1.- Objetivos y Alcance

Teniendo en cuenta lo anterior, este TFM plantea como **objetivo general**: Diseñar e implementar un prototipo de bajo coste para el control domótico del hogar basado en el reconocimiento de la voz y utilizando software libre.

Para dar cumplimiento al objetivo general del proyecto se marcan los siguientes **objetivos específicos**:

1. Implementar una centralita basada en Asterisk como pasarela de comunicación entre el usuario y el hogar domotizado.
2. Configurar un IVR en la centralita de Asterisk que sirva de interfaz con el usuario.
3. Integrar los servicios de la API STT de Google *Cloud* con la centralita para efectuar el reconocimiento de voz.

4. Implementar el control domótico de los sensores y actuadores presentes en el hogar domotizado. Dicho control se basa en la plataforma de Arduino como interfaz de hardware.
5. Capturar información de los dispositivos IoT para la toma de decisiones automáticas en función de los valores recogidos.
6. Programar un conjunto de *scripts* que permita el intercambio de datos entre todos los elementos del sistema.

El alcance del TFM es conseguir un sistema domótico que sea capaz de integrar plataformas de hardware y software libres como Arduino y Asterisk (y en menor medida Telegram), logrando un intercambio de datos bidireccional a través de Internet. Además, que el sistema tenga la capacidad de ejecutar las acciones ordenadas por el usuario inmediatamente o pueda planificarlas en el futuro indicando la fecha y hora de ejecución. Asimismo, se debe notificar al usuario sobre eventos producidos en el hogar.

1.2.- Distribución de Tareas

La metodología seguida es la siguiente: en primer lugar, se realiza una búsqueda de artículos científicos, así como de diversos trabajos relacionados con el presente trabajo. Con la información recopilada se realiza un estudio del estado del arte actual de la temática en cuestión. De los trabajos anteriores se detectan algunos aspectos que se pueden mejorar y se decide buscar una alternativa más eficiente para la traducción de las órdenes del usuario, sin penalizar la calidad de la traducción. Para ello, se decide hacer uso de la API STT de Google. Al mismo tiempo, se determina que todas las comunicaciones entre la centralita y el hogar domotizado se realicen mediante el protocolo HTTP (*Hypertext Transfer Protocol*), con el propósito de que el usuario pueda utilizar el sistema desde cualquier localización a través de Internet.

El siguiente paso fue el desarrollo e implementación del prototipo de tal manera que se cumplan los objetivos propuestos inicialmente. Luego se somete a prueba el sistema para comprobar el correcto funcionamiento del mismo. En esta fase se detectan los errores y se corrigen. Finalmente, se presentan los resultados que arrojan las pruebas y se extraen las conclusiones.

En Figura 1.2 se muestra el diagrama de Gantt del proyecto, donde se ve reflejado la división de las tareas y los tiempos que consume la realización de dichas tareas.

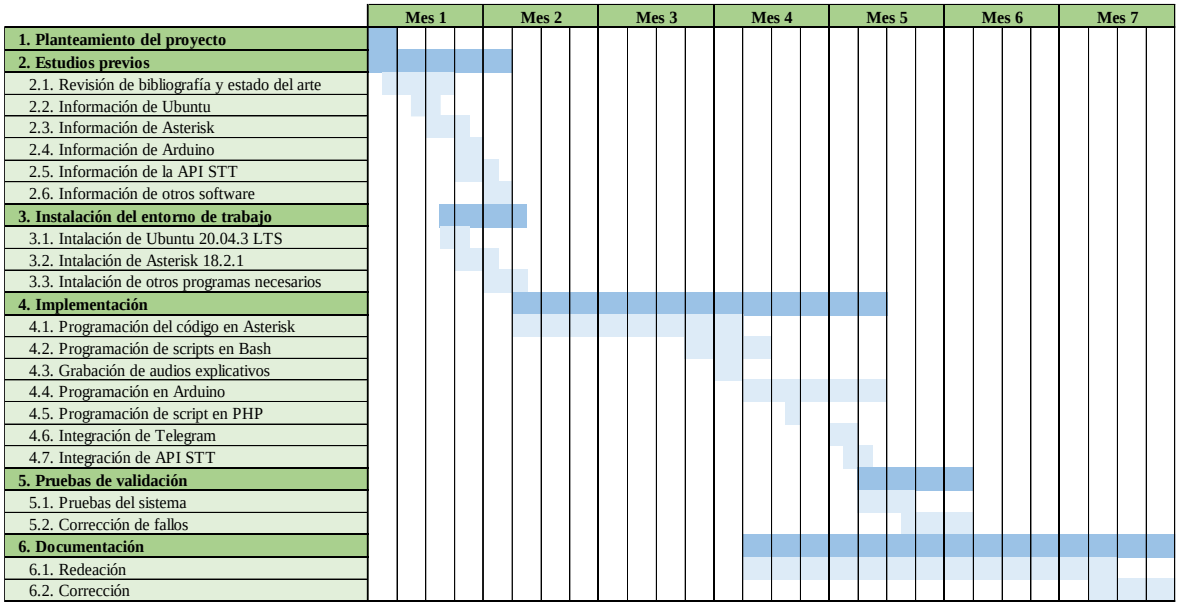


Figura 1.2.-Diagrama de Gantt.

1.3.- Organización del Documento

La estructura de este trabajo es la siguiente:

1. **Introducción:** Se realiza una reseña donde se define la necesidad, motivación, actualidad e importancia del tema que se aborda y se mencionan los elementos principales del diseño técnico.
2. **Marco Teórico. Tecnologías:** Se realiza un estudio de todos los conceptos teóricos necesarios para comprender el funcionamiento del prototipo. Se expone las distintas tecnologías empleadas.
3. **Estado del Arte:** En este capítulo se estudia el estado del arte de IoT en el ámbito de la domótica.
4. **Requisitos del Sistema:** En este se definen los requisitos funcionales y no funcionales del sistema.
5. **Diseño del Sistema:** Se detalla las principales fases del diseño, se propone la arquitectura con las funciones de los diferentes componentes del sistema, etc. También se describen los casos de uso.
6. **Descripción técnica del Sistema:** Este capítulo se describen los aspectos técnicos requeridos para llevar a cabo la implementación del sistema.
7. **Validación y Resultados:** Se describen las pruebas realizadas una vez finalizado e implementado el diseño propuesto, con el fin de validar los requisitos del sistema.
8. **Conclusiones y Líneas Futuras:** Se mencionan las conclusiones obtenidas de todo el trabajo realizado y se proponen las líneas de trabajo futuro en función de los aspectos que no han sido cubiertos con el presente proyecto.

9. **Presupuesto:** En este capítulo se detallan los costes del desarrollo e implementación del sistema por concepto de recursos hardware, software y recursos humanos.
10. **Anexos:** Se muestra presenta el manual del instalador, el manual de usuario y otros aspectos de importancia no incluidos en los capítulos de este documento.
11. **Referencias bibliográficas:** Se conforma un listado de toda la bibliografía que se ha consultado.

2.- MARCO TEÓRICO. TECNOLOGÍAS

Este capítulo aborda todas las temáticas teóricas necesarias para comprender el funcionamiento del prototipo de bajo coste propuesto. El objetivo de este capítulo es familiarizar al lector con las distintas tecnologías empleadas, mencionando los conceptos básicos, especificaciones técnicas, y otros temas pertinentes. Si el lector ya se encuentra familiarizado con los fundamentos de IoT, VoIP (*Voice Over Internet Protocol*), Asterisk, Arduino y las otras tecnologías que se abordan, se recomienda una lectura superficial o ahorrarse la lectura de este capítulo.

2.1.- Internet de las Cosas

El Internet de las Cosas, o IoT, es una tecnología aplicada a escenarios donde los objetos, sensores y artículos de uso diario se conectan a Internet, de manera que estos generen, intercambien y procesen información sobre un entorno físico para proporcionar servicios de valor añadido a los usuarios finales, con una mínima intervención humana [3]. En la Figura 2.1 se observa un esquema de los tres elementos claves que intervienen en el IoT.

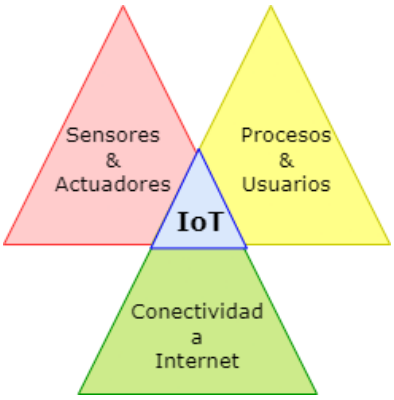


Figura 2.1.- Elementos claves de IoT.

Estos sensores y dispositivos se conectan a Internet a través de redes cableadas o inalámbricas, creando así una malla de conexiones a nivel mundial que logra alcanzar a una multitud de “cosas” de uso cotidiano que reaccionan de forma autónoma e inteligente ante eventos o cambios en el entorno.

Aunque el término IoT es relativamente nuevo, fue empleado por primera vez en 1999 por Kevin Ashton para describir un sistema en el cual los objetos del mundo físico se podían conectar a Internet por medio de sensores. Además, durante varias décadas se han hecho intentos de combinar ordenadores y redes para monitorizar y controlar diferentes dispositivos, lo que en ese tiempo fueron soluciones basadas en redes dedicadas para un propósito concreto y no en redes basadas en el Protocolo IP (*Internet Protocol*) y los

estándares de Internet. No obstante, en el año 1990 ya se había presentado en una conferencia el primer dispositivo conectado a Internet vía IP, y en los años posteriores se fueron conectando otras "cosas" también vía IP.

Si bien no es nueva la idea de conectar objetos entre sí a través de Internet, no es hasta hoy, que convergen todas las tecnologías que han disparado el auge del IoT. Por un lado, está el proceso continuo para dar conectividad ubicua a Internet y la adopción generalizada de redes basadas en el protocolo IP. Por otro, están las diferentes innovaciones en las áreas de la electrónica, la informática y las TICs (Tecnologías de la Información y la Comunicación), que han permitido la miniaturización de los microprocesadores y un aumento de la capacidad de cómputo de los sistemas. Además, se ha producido una reducción sustancial de los costes de fabricación y comercialización de todo el hardware que interviene en esta tecnología.

En la actualidad, la popularización del IoT ha sido tal que una amplia gama de sectores de la industria está incorporando esta tecnología a sus productos, servicios y operaciones. Entre ellos, está el sector de la automoción, la salud, entornos urbanos, de oficinas y también del hogar, siendo este último el sector de interés para este proyecto. En la Figura 2.2 se presenta un esquema de un hogar domotizado.

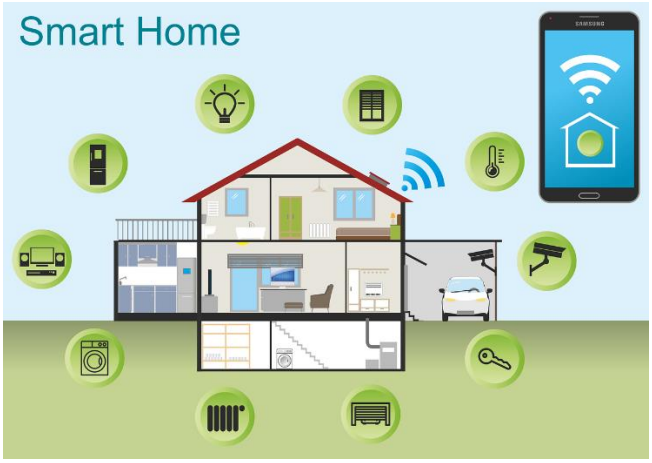


Figura 2.2.-Hogar inteligente o domotizado.

En la figura anterior se puede observar que una gran cantidad de dispositivos están conectados a Internet, lo que permite gestionar y controlar el hogar de forma remota y en cualquier momento. En pocas palabras, la domótica trata de integrar la tecnología existente en todos los aspectos posibles dentro de un hogar tradicional.

A pesar del entusiasmo generalizado en torno al IoT en todos los sectores de la vida cotidiana, lamentablemente, el IoT también plantea importantes desafíos como la seguridad, la privacidad, etc., que pueden dificultar la realización de sus potenciales beneficios. Sin embargo, estos puntos no son abordados en el presente trabajo.

2.1.1.- Modelos de Comunicación

En el año 2015 se publicó la RFC 7452 [4] donde se describe un marco de cuatro modelos de comunicación comunes que utilizan los dispositivos IoT. A continuación, se mencionan las principales características de cada uno.

2.1.1.1.- Comunicación: Dispositivo a dispositivo

Es un modelo representado por dos o más dispositivos que se conectan y se comunican directamente entre sí, sin un servidor de aplicaciones intermediario. Estos dispositivos se comunican sobre muchos tipos de redes, ya sean de ámbito local, o global como Internet. Sin embargo, para establecer comunicaciones directas entre dispositivos, muchas veces se utilizan otros protocolos como *Bluetooth*, *ZigBee* o *Z-Wave* (ver Figura 2.3).

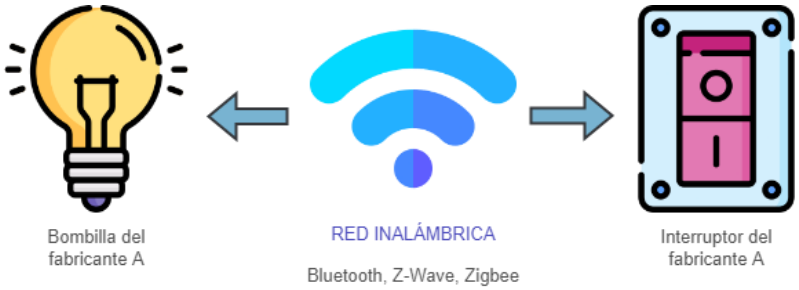


Figura 2.3.-Modelo de comunicación: Dispositivo a dispositivo.

2.1.1.2.- Comunicación: Dispositivo a la nube

En este caso, el dispositivo IoT se conecta directamente a un servicio en la nube, como por ejemplo un proveedor de servicios de aplicaciones para intercambiar datos. Este enfoque suele aprovechar la infraestructura existente como las conexiones WiFi o redes Ethernet para establecer una conexión entre el dispositivo y la red IP, para luego conectarse con el servicio en la nube. Este modelo se ilustra en la Figura 2.4.

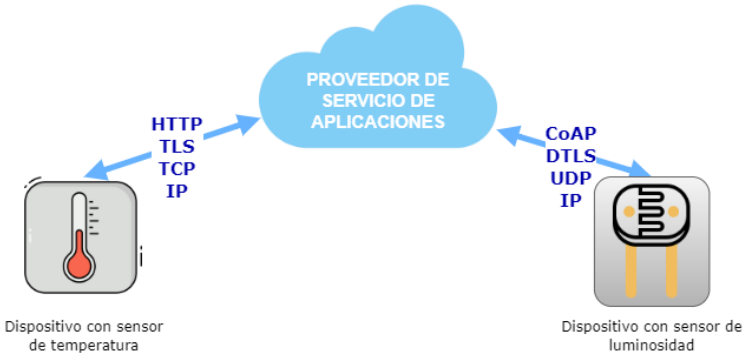


Figura 2.4.- Modelo de comunicación: Dispositivo a la nube.

2.1.1.3.- Comunicación: Dispositivo a puerta de enlace

El dispositivo IoT, en este modelo, se conecta a través de un servicio ALG (*Application Layer Gateway*) para llegar a un servicio en la nube. Es decir, que hay un software de aplicación corriendo en un dispositivo de puerta de enlace local, que actúa como intermediario entre el dispositivo y el servicio en la nube. Este servicio provee también seguridad y otras funcionalidades como traducción de protocolos o datos. En la Figura 2.5 se puede observar este modelo.

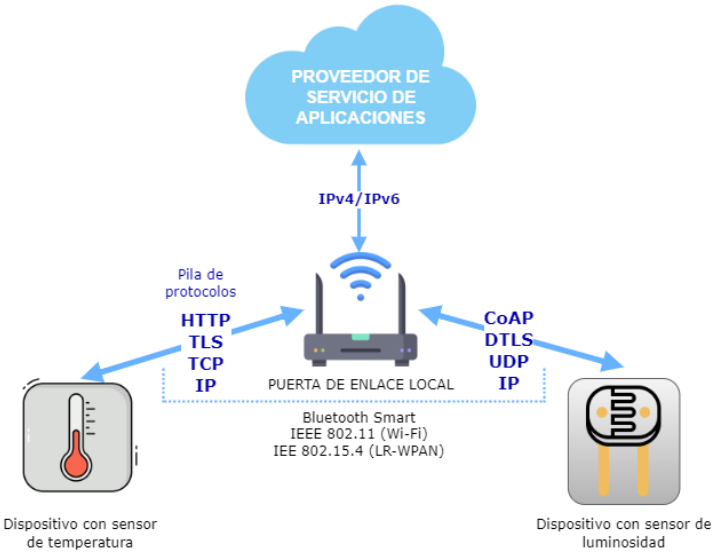


Figura 2.5.- Modelo de comunicación: Dispositivo a la nube.

2.1.1.4.- Intercambio de datos a través del *back-end*

Este modelo hace referencia a una arquitectura de comunicación que permite que los usuarios, mediante la comunicación con un servicio en la nube y en combinación con datos de otras fuentes, envíen los datos de sus objetos inteligentes para que estos sean analizados. Con esta arquitectura el usuario permite que los datos subidos por sus sensores sean accedidos por terceros. La Figura 2.6 muestra una representación de esta arquitectura.

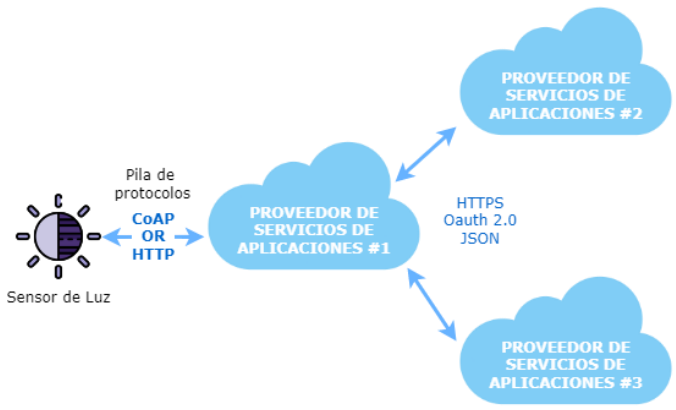


Figura 2.6.- Modelo de comunicación: Intercambio de datos a través del *back-end*.

2.1.2.- Dispositivos usados en IoT

Los sensores y actuadores son uno de los pilares principales del IoT ya que son los encargados de recopilar información del entorno e interactuar con él. Estos dos términos no deben ser confundidos. Los sensores permiten conocer el valor de variables físicas de un medio y las transforma en una señal eléctrica, que puede ser interpretada por un microcontrolador que tiene programada ciertas directrices de actuación en base a las mediciones recibidas. Es aquí donde entran los actuadores, que se encargan de convertir la señal eléctrica, en energía que es capaz de activar un determinado proceso.

Los microcontroladores o microprocesadores también son piezas claves, ya que son el cerebro del sistema en cuestión. Un microcontrolador se comporta como un intermediario, ya que "escucha" a los sensores y "habla" con los actuadores, en función de las órdenes que tenga programada en su memoria.

2.1.2.1.- Sensores

Como se ha mencionado, los sensores típicamente convierten estímulos físicos en señales analógicas o digitales, para ser entregadas a un dispositivo de gestión y control que las procesa para tomar decisiones. Existen gran cantidad de sensores y se pueden clasificar atendiendo a varios criterios:

- Según el principio de funcionamiento:
  - **Activos:** generan la señal de salida sin la necesidad de una fuente de alimentación externa.
  - **Pasivos:** generan la señal de salida por intermedio de una fuente auxiliar.
- Según el tipo de señal que generan:
  - **Digitales:** frente a un estímulo cambian de estado (binario) y los valores de tensiones obtenidos son 0V y 5V.
  - **Analógicos:** entregan niveles de tensiones variable en el tiempo.
- Según el tipo de variable física medida:
  - **Magnéticos:** se sirve del efecto *Hall* para la medición de campos magnéticos o corrientes.
  - **Mecánicos:** son utilizados para medir desplazamiento, posición, movimiento, tensión, etc.
  - **Térmicos:** se usan para la medición de la temperatura.
  - **Acústicos:** convierten una señal acústica en eléctrica, como por ejemplo un micrófono.

La clasificación de los tipos de sensores es mucho más extensa de lo que se ha abordado en este trabajo, pero no es objetivo hacer una amplia disertación en esta área de la electrónica, simplemente hacer mención de algunos de ellos.

2.1.2.2.- Actuadores

Se denominan actuadores a aquellos elementos que, en función de una señal recibida, pueden provocar un efecto directamente sobre el mundo físico. Estos generaran el movimiento de los elementos según las órdenes transmitidas por el microcontrolador, transformando la energía de entrada en energía de salida utilizable para realizar una acción final. Los actuadores no suelen ser capaces de procesar datos.

Se pueden clasificar en tres grupos según la fuente de procedencia de la fuerza que provocan:

- **Eléctricos:** Son aquellos capaces de ser movilizados por energías eléctricas. Son los actuadores más comunes en el mercado. Ejemplo de ellos son los motores de corriente continua, motores paso a paso, los servomotores, los relés, electroválvulas, etc.
- **Hidráulicos:** Son los que requieren de una fuerza que proviene de un líquido, para la realización de sus movimientos. Imprimen gran velocidad y se emplean cuando se necesita mucha potencia. Ejemplos de este tipo de actuadores son las válvulas, los motores y cilindros hidráulicos, etc.
- **Neumáticos:** Son aquellos capaces de transformar la energía proveniente por el aire en movimiento, es decir, que convierten la energía eólica en energía mecánica. El rango de presión es mayor que en los hidráulicos ya que el aire comprimido puede generar mayor velocidad. Estos se agrupan a su vez, en cilindro simple y doble.

2.1.2.3.- Microcontroladores

Un microcontrolador es un chip o circuito que integra un gran número de componentes en un solo encapsulado y tiene la característica de ser programable. Es decir, que es capaz de ejecutar de forma autónoma una serie de instrucciones previamente definidas por el programador [5].

Por definición, un microcontrolador ha de incluir en su interior tres elementos básicos: la CPU (*Central Processing Unit*), diferentes tipos de memorias para alojar las instrucciones y datos que necesita la CPU, y diferentes pines de E/S (entrada/salida). Los pines son los encargados de comunicar el microcontrolador con el exterior, bien sea para recibir datos de sensores provenientes del entorno o para enviar órdenes a los actuadores e interactuar con el medio físico. En resumen, un microcontrolador está especializado en ejecutar constantemente un conjunto de instrucciones predefinidas, en base a la información recibida y enviada por sus pines de E/S.

2.2.- Plataforma Arduino

Arduino es una plataforma abierta de electrónica para la creación de prototipos flexibles y fáciles de usar, basada en software y hardware libre.

El software de Arduino es gratis y multiplataforma (ya que funciona en Linux, MacOS y Windows). Se publica bajo la licencia GPL (*General Public Licence*), esto quiere decir que cualquier persona con conocimientos puede formar parte del desarrollo del software Arduino, contribuyendo con mejoras continuas, sugiriendo ideas, funcionalidades, características, compartiendo soluciones a posibles errores existentes, etc. Esto provoca una comunidad espontánea que colabora mutuamente a través de Internet para optimizar las prestaciones de esta plataforma. El entorno de desarrollo Arduino está basado en el entorno *Processing*, aunque internamente el lenguaje de Arduino se basa en código C/C++ y no en Java como *Processing*.

La placa hardware de Arduino incorpora un microcontrolador de tipo Atmel AVR reprogramable y una serie de pines-hembra, los cuales están unidos internamente a los pines de E/S del microcontrolador que permiten conectar una amplia gama de sensores y actuadores de forma muy sencilla y cómoda [5]. Estas placas se programan mediante un ordenador con el IDE (*Integrated Development Environment*) de Arduino instalado o a través del navegador web, usando comunicación serie.

En la actualidad, existen multitud de microcontroladores y plataformas (hardware más software) de otros fabricantes que ofrecen funcionalidades más o menos similares a las que ofrece Arduino. No obstante, el uso de placas Arduino brinda una serie de ventajas respecto a sus competidores, las cuales se mencionan a continuación y son las que justifican la selección de esta plataforma para este proyecto.

- **Precio:** El coste de las placas Arduino es menor respecto al resto de placas.
- **Multiplataforma:** Se puede instalar y ejecutar en sistemas operativos Windows, MacOS y Linux.
- **Facilidad de uso:** El IDE ofrece un sistema de gestión de librerías y placas muy práctico. Es un software sencillo que carece de funciones avanzadas típicas de otros IDEs, pero suficiente para programar, lo que lo convierte en una buena herramienta para principiantes.
- **Libre y extensible:** Cualquiera que desee ampliar y mejorar tanto el diseño hardware de las placas como el software y el propio lenguaje de programación, puede hacerlo. Esto permite que existan muchas variantes de placas no oficiales (que a su vez son más económicas) y librerías software de terceros, que pueden adaptarse mejor a las necesidades concretas de cada proyecto.
- **Librerías:** Una de las mayores ventajas de Arduino es que existen librerías para prácticamente cualquier dispositivo externo que se le quiera acoplar.

- **Documentación y tutoriales:** Internet contiene mucha documentación de esta plataforma. La gran comunidad que integra Arduino está continuamente enriqueciendo la documentación y compartiendo sus ideas a través de tutoriales.
- **Reutilizables y versátiles:** Son reutilizables porque se puede aprovechar la misma placa para varios proyectos (ya que es muy fácil de desconectarla, reconectarla y reprogramarla), y versátiles porque las placas Arduino son compatibles con infinidad de periféricos como teclados, *displays*, sensores, actuadores, etc.

Por último, mencionar que actualmente hay diferentes modelos de placas oficiales de Arduino. La versión estándar es Arduino UNO, pero existen otras como Arduino Leonardo, Arduino Mega 2560, Arduino Nano, Arduino YUN, Arduino DUE, entre otras. Todas ellas están especializadas en trabajar dentro de situaciones específicas donde la placa UNO estándar no ofrece soluciones a las necesidades que puedan surgir.

Para el proyecto en cuestión se ha adquirido la placa Mega2560 de OSOYOO. Los dos próximos apartados están dedicados a este modelo concreto.

2.2.1.- Placa Mega2560 de OSOYOO

La placa Mega2560 de OSOYOO [6] es fabricada y ensamblada por la compañía OSOYOO, no es una placa original de Arduino. Al tratarse de hardware libre, es perfectamente legal, de ahí que sea más económica que si se tratara de la versión original.

Esta placa es totalmente compatible con Arduino Mega2560 rev.3 e incluye un microcontrolador basado en el ATmega2560. Tiene 54 pines de E/S digital, de los cuales 15 se pueden usar como salidas PWM (*Pulse Width Modulation*), 16 entradas analógicas, 4 puertos serie de hardware (UART, *Universal Asynchronous Receiver-Transmitter*), un oscilador de cristal de 16 MHz, una conexión USB (*Universal Serial Bus*) tipo B, un conector de alimentación, un encabezado ICSP (*In Circuit Serial Programming*) y un botón de reinicio. En la Figura 2.7 se muestra una imagen de esta placa.

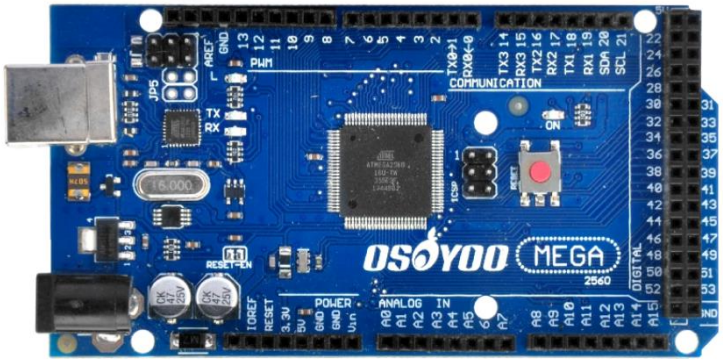


Figura 2.7.- Placa Mega2560 de OSOYOO.

A continuación, se presenta la Tabla 2.1Tabla 2.1 con las especificaciones técnicas de esta placa:

Microcontrolador	**ATmega2560**
Tensión de funcionamiento	5 V
Voltaje de entrada (recomendado)	7-12 V
Voltaje de entrada (límite)	6-20 V
Pines de E/S digitales	54 (15 salidas PWM)
Pines de entrada analógica	16
Corriente CC por pin de E/S	20 mA
Corriente CC para pin de 3.3V	50 mA
Memoria *flash*	256 KB (8 KB usados por el gestor de arranque)
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz
Dimensiones (mm)	101.52 x 53.3
Peso (g)	37

Tabla 2.1.- Especificaciones técnicas de la Placa Mega2560 de OSOYOO.

2.2.1.- *Shield* Mega-IoT de OSOYOO

El *shield* MEGA-IoT de OSOYOO [7] es una placa de expansión que integra el módulo ESP12S, construido con el chip ESP8266 para proporcionar una conexión WiFi de bajo coste a la placa Arduino Mega2560. Este *shield* es PnP (*Plug and Play*), siendo muy cómodo ya que no se necesitan soldaduras ni cableados adicionales, simplemente se debe unir el *shield* a la placa Mega2560 como se muestra en la Figura 2.8. Luego, se seleccionan los pines para la comunicación en serie con mini puentes o *jumpers* y la conexión del hardware está lista para comenzar a programarlo.

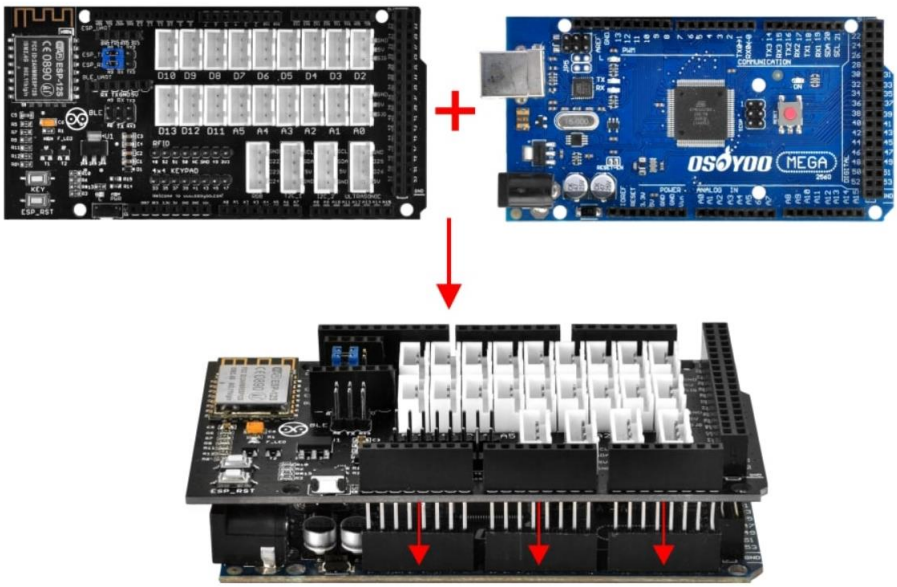


Figura 2.8.- Conexión de la placa Mega2560 con el *shield* Mega-IoT.

El módulo ESP12S viene de serie con un firmware de comandos AT (*Attention*) que son el conjunto de comandos que sirven para configurar y parametrizar este dispositivo. Por tanto, puede ser controlado a través de cualquier puerto UART/Serie.

En la Figura 2.9 se observa el módulo WiFi integrado en la placa, además de una señalización del resto de componentes, donde:

1. Puerto de depuración ESP8266: Estos 4 pines se pueden usar para *flashear* el firmware ESP8266 o depurar el comando AT a través del puerto serie.
2. Selector ESP WiFi: Si los *jumpers* se conectan al lado izquierdo, ESP se conecta al *software Serial* (A8, A9), si los *jumpers* están a la derecha, ESP8266 se conecta al *hardware Serial1* de Mega2560.
3. Conexión para el módulo *Bluetooth* HC-02 (HC-05, HC-10).
4. Ranuras para conectar módulos digitales de OSOYOO (sensores o actuadores).
5. Ranuras para conectar módulos analógicos de OSOYOO (sensores o actuadores).
6. Ranura para el módulo de sensor ultrasónico OSOYOO
7. Ranuras de 4 pines para conectar dispositivo I2C. Ejemplo: OSOYOO I2C LCD.
8. Ranura para el módulo RGB OSOYOO.
9. Conexión para el teclado 4x4 OSOYOO.
10. LED indicador de potencia.
11. Conexión para el módulo RFID OSOYOO.
12. Botón de reinicio de Arduino.
13. LED Arduino (D13).
14. Botón de reinicio ESP8266 (para actualizar el firmware).
15. Botón de llave ESP8266 (para actualizar el firmware).
16. Convertidor de nivel integrado de 5V-3.3V.

17. Selector de serie *Bluetooth*: si los *jumpers* se conectan al lado izquierdo, ESP se conecta al *softwareSerial* (A8, A9), si están a la derecha, ESP8266 se conecta al *hardware Serial1* de Mega2560.

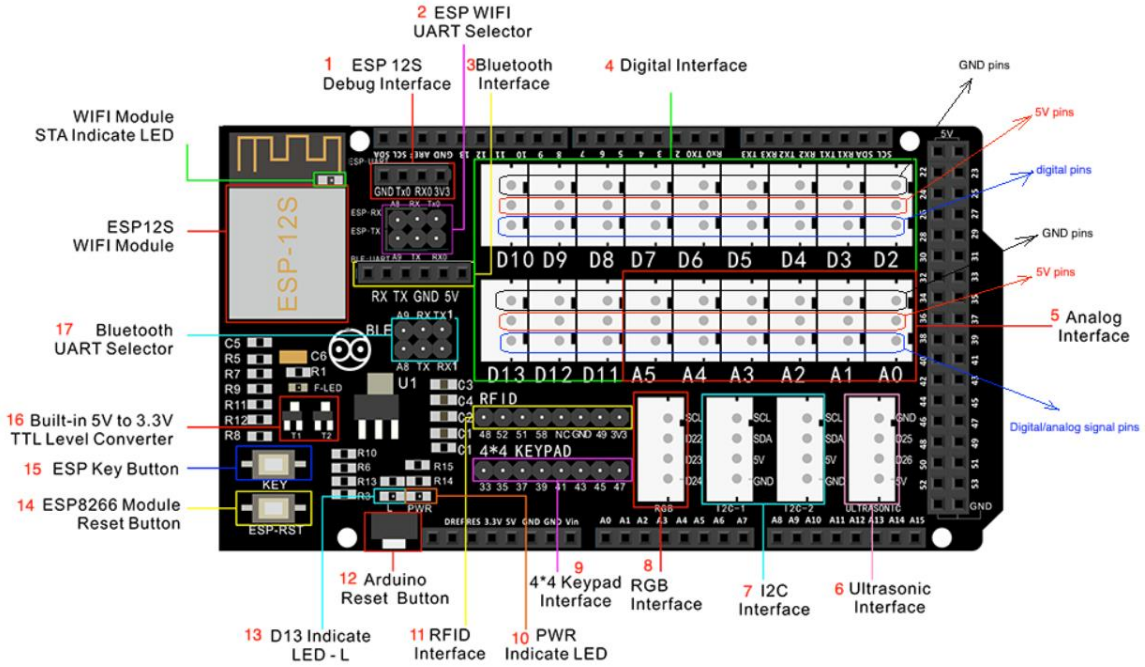


Figura 2.9.- Señalización de componentes del *shield* Mega-IoT de OSOYOO.

2.2.2.- Dispositivos Empleados

En este subapartado se realiza una breve descripción de los dispositivos empleados para simular los diferentes equipos de un hogar. Se mencionan las especificaciones técnicas de cada uno y se hace alusión a algunos conceptos básicos. Estos *kits* de dispositivos se han adquirido de los fabricantes OSOYOO [8] y Keyestudio [9]. Todos los módulos son PnP, ya que en ellos viene incorporada la circuitería necesaria (resistencias, condensadores, diodos, transistores, etc.) para su correcto funcionamiento. No requieren de ningún elemento adicional, solo hay que conectarlos a la placa de Arduino y ya están listos para ser usados.

2.2.2.1.- Módulos LEDs

Un LED (*Light Emitting Diode*) es un dispositivo que emite luz cuando se encuentran en polarización directa, es decir el ánodo (pata larga) está conectado a un polo positivo y el cátodo (pata corta) a un polo negativo.

Se usan cuatro módulos de LEDs de OSOYOO de color amarillo, blanco, verde y rojo. Estos módulos incorporan las resistencias para limitar la corriente que circula por los diodos LEDs. Tienen tres pines de salida: VCC (*Voltage Common Collector*) tensión

positiva de alimentación, GND (*ground*) referencia negativa del circuito y SIG (*signal*). Pueden conectarse tanto a un pin digital como analógico. El voltaje de funcionamiento es de 3.3 ~ 5 V_{DC}. En la Figura 2.10 se muestran estos componentes. Se utilizan, además, otros dos módulos LEDs del otro fabricante con características similares.

Figura 2.10.- Módulos LEDs.

2.2.2.2.- Módulo RGB

El módulo RGB de OSOYOO (ver Figura 2.11), cuyas siglas hacen referencia a los colores primarios (*red*, *green*, *blue*), reproduce una amplia gama de colores mezclando los tres primarios en el rango de valores de 0 a 255. Este módulo consta de un LED SMD (*Surface Mounted Device*) modelo RGB KY-009 y tres resistencias para limitar el paso de la corriente. Tiene cuatro pines: R, G, B y GND (cátodo común). Los colores se ajustan a través de los pines R, G, B mediante señales PWM. El voltaje de trabajo es de 3.3 ~ 5V_{DC}.

Figura 2.11.- Módulo RGB.

2.2.2.3.- Sensor de movimiento

El sensor de movimiento o PIR (*Passive InfraRed*) es un dispositivo con propiedades piroeléctricas usado para la detección de presencia o movimiento de animales y personas. Cualquier objeto emite calor en forma infrarroja y este sensor se basa en ese principio para detectar la luz infrarroja que irradia dichos objetos en su rango de visión.

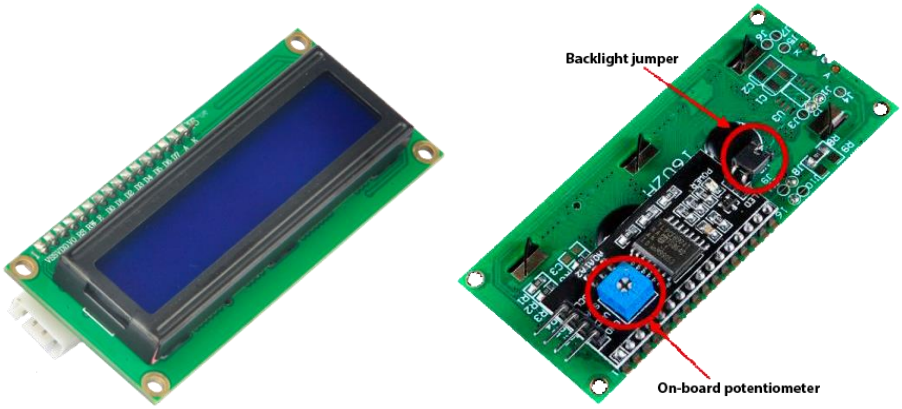
El dispositivo tiene tres pines: GND, VCC y SIG, este último es el que emite una señal eléctrica repentina una vez que detecta que hay radiación infrarroja dentro de un cono de 110° desde su centro y una distancia máxima de hasta 7 metros. El voltaje de entrada es de 3.3 ~ 18V_{DC}. Una imagen de dicho sensor se muestra en la Figura 2.12.

Figura 2.12.- Sensor infrarrojo pasivo (PIR).

2.2.2.4.- LCD 1602

LCD 1602 (*Liquid-Crystal Display*) es una pantalla de visualización que puede mostrar hasta dos líneas de dieciséis caracteres cada una de color blanco sobre fondo azul. Este modelo tiene incorporado un adaptador LCD a I2C (*Inter-Integrated Circuit*) que trae toda la circuitería necesaria para realizar la conversión. Los pines de salida de este módulo son los que se conectan al Arduino y son los siguientes: SDA (*System Data*) para trasmitir los datos, SLC (*System Clock*) es el reloj asíncrono que indica cuándo leer los datos, más los pines GND y VCC. El voltaje de alimentación es 5V.

La luz de fondo se puede controlar mediante el firmware (desde el IDE de Arduino) o quitando el *jumper* que tiene el módulo adaptador. Igualmente se puede controlar el contraste del LCD ajustando el potenciómetro que también tiene integrado (ver Figura 2.13b).



(a) (b)

Figura 2.13.- LCD 1602: (a) Parte delantera, (b) Parte trasera.

2.2.2.5.- Servomotor

El servomotor es un actuador rotativo similar a los motores de corriente continua, pero con la diferencia de que pueden ejercer un control preciso de la posición dentro de su rango de operación. Para ello, tienen integrado un sensor de posición. Todos tienen tres cables de conexión que típicamente suelen distinguirse por los colores marrón, rojo y

naranja. El marrón es para GND, el rojo para VCC y el naranja para la señal. El voltaje de operación es de 4.8 ~ 6V_{DC}. En la Figura 2.14 se observa una imagen de los dos micro servomotores que se utilizan.

Figura 2.14.- Micro Servomotores.

El ángulo de rotación del servomotor se controla regulando el ciclo de trabajo de la señal PWM. El ciclo estándar de la señal PWM es de 20 ms (50 Hz). El ancho se distribuye entre 500 µs - 2500 µs, correspondiendo al ángulo de rotación de 0° a 180°, esto se representa en la Figura 2.15.

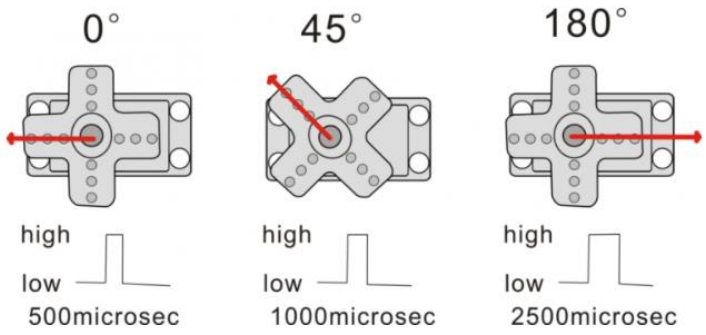


Figura 2.15.- Correspondencia entre el ciclo de trabajo y el ángulo de rotación.

Hay dos formas de controlar un servomotor con Arduino. Una es usar un puerto digital común de Arduino para producir ondas cuadradas con diferentes ciclos de trabajo (simulando la señal PWM) y utilizar esa señal para controlar la posición del motor. La otra forma es usar la función *Servo* del Arduino para controlar el motor. Esta solución es más sencilla, pero tiene la limitación de que solo se pueden controlar dos motores, a través de los pines digitales 9 y 10 exclusivamente. Esta variante es la que se aplica en el proyecto ya que se necesitan controlar dos servomotores solamente.

2.2.2.6.- *Buzzer* activo y pasivo

Un *buzzer*, o zumbador, es un dispositivo capaz de emitir sonidos. Puede ser mecánico, electromecánico o piezoeléctrico, siendo este el más conocido y habitual. El último de los mencionados incorpora un elemento piezoeléctrico al que, al aplicarle una

señal eléctrica, se deforma y produce un sonido. A su vez, los *buzzer* piezoeléctricos se dividen en dos grandes grupos: activos y pasivos. Físicamente pueden ser muy parecidos, o incluso idénticos, la diferencia radica en su funcionamiento interno.

Los *buzzer* activos contienen un oscilador que provoca la emisión de un sonido a frecuencia fija tan pronto como se conectan a la alimentación. Esto quiere decir que no se puede variar el tono del sonido emitido. En oposición, los *buzzer* pasivos carecen del oscilador, por lo que su funcionamiento depende de una señal cuadrada externa que tenga diferentes frecuencias. Una forma de identificarlos es aplicarle una tensión de corriente continua. Si suena, entonces se trata de un *buzzer* activo, de lo contrario, se trata de un *buzzer* pasivo.

Los *buzzer* solo tienen dos pines (positivo y negativo), pero no es recomendable conectarlos directamente a Arduino, ya que le puede suministrar hasta 40 mA y se necesita limitar la corriente que pasa a través de este dispositivo. Además, se recomienda usar un transistor con función de interruptor para que no se desperdicie energía cuando el *buzzer* no emite sonido. Por ello, lo más fácil es adquirir un módulo donde viene integrada toda la electrónica: resistencia, transistor y el *buzzer*. En la Figura 2.16 se observan los dos módulos empleados para este proyecto, uno activo y otro pasivo.

(a) (b)

Figura 2.16.- *Buzzer* : (a) activo, (b) pasivo.

Como se observa en la figura anterior, cuando se emplean los módulos se utilizan tres pines: GND, VCC y SIG. Controlar el *buzzer* activo es muy sencillo, basta con enviar niveles de señal alto (5V) y bajo (0V) por el pin SIG. En el caso del *buzzer* pasivo se suele utilizar la librería *Tone*, propia de Arduino, donde se le pasa como parámetro la frecuencia y la duración del tono (este último es opcional). También se pueden utilizar librerías de terceros que permiten programar *beeps* totalmente configurables para generar mejores efectos de sonido.

2.2.2.7.- *Fan* Motor

El módulo de *Fan* utiliza el *driver* de control del motor L9110. Tiene cuatro pines: GND, VCC, y dos pines de señal INA y INB. Mediante pulsos PWM se puede controlar la

velocidad del motor a través del pin INA, mientras que con el pin INB, se puede variar la dirección de rotación del motor: nivel bajo (0V) para un sentido, y nivel alto (5V) para el sentido contrario. El voltaje de operación es de $5V_{DC}$. En la Figura 2.17 se muestra este dispositivo.

Figura 2.17.- *Fan* motor L9110.

2.2.2.8.- Sensor DHT11

El DHT11 es un sensor que proporciona medidas de temperatura y humedad en el rango de 0-50 ºC con un error de ± 2 °C y humedad relativa de 20-90% con un error de ± 5% respectivamente. No obstante, la medida de humedad carece de valor para el presente proyecto. Este sensor tiene 4 pines: VCC, DATA (Datos), NC (*Not Connected*) y GND. Sin embargo, es muy habitual encontrarlo integrado en una PCB (*Printed Circuit Board*) con una resistencia de *pull-up*. En tal caso, el módulo se reduce a 3 pines: VCC, GND y DATA o SIG, este último se conecta a cualquiera de las entradas digitales de Arduino. La tensión de alimentación es de +5V. En la Figura 2.18 se muestra un módulo de este dispositivo.

Figura 2.18.- Sensor DHT11.

2.2.3.- Entorno de Desarrollo (IDE)

El IDE de Arduino es un editor de texto y compilador que permite a los programadores escribir y probar sus propios programas para después transferir el contenido de las instrucciones a la placa de Arduino en su lenguaje máquina. Es una aplicación multiplataforma (disponible para Windows, MacOS y Linux).

En la Figura 2.19 se muestra una ilustración de la interfaz gráfica del IDE Arduino, señalando las partes principales que la componen.

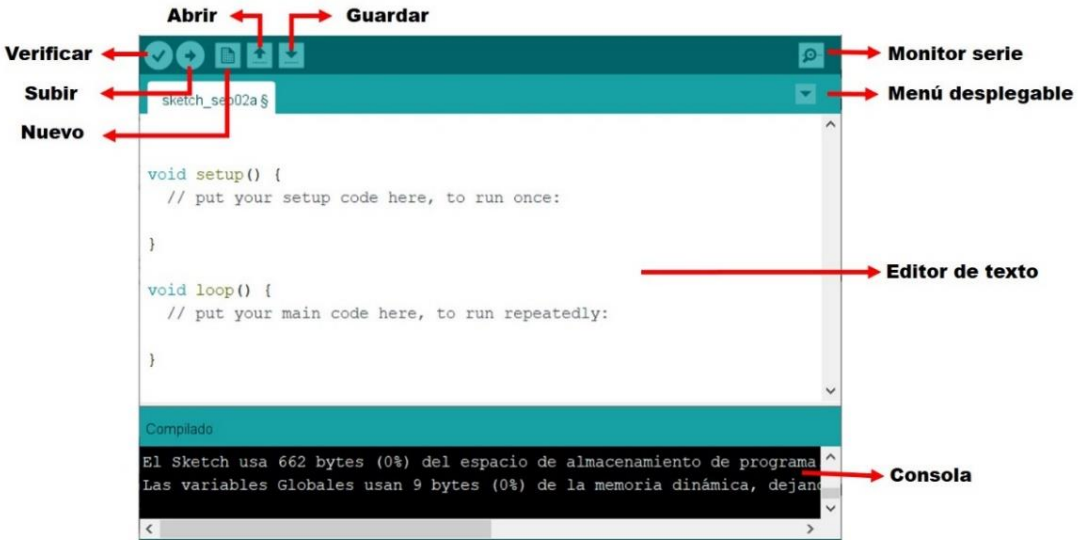


Figura 2.19.- Interfaz gráfica del IDE de Arduino.

A continuación, se describe brevemente la función de cada una.

- Barra de botones: Compuesta a su vez por los siguientes elementos:
  - **Verificar:** Comprueba la sintaxis del programa. Si el código escrito es correcto entonces lo compila.
  - **Subir:** Si la verificación ha sido correcta, se presiona este botón para cargar en la memoria del microcontrolador de la placa Arduino el programa compilado.
  - **Nuevo:** Crea un nuevo programa vacío.
  - **Abrir:** Para abrir los programas disponibles en otros directorios.
  - **Guardar:** Guarda el código del programa en un fichero con extensión "*ino*" en el directorio que se especifique.
  - **Monitor serie:** Abre el monitor serie para visualizar los datos transferidos.
  - **Menú desplegable:** Permite crear nuevas pestañas, borrarlas, renombrarlas, etc. Es útil cuando el programa es muy extenso, ya que es posible programar cada función en una pestaña diferente.
- Editor de texto: Es la parte principal de la interfaz del IDE y está compuesto por tres secciones:
  - **Sección de declaración de variables globales:** ubicada directamente al inicio del programa y está reservada para declarar las variables que pueden ser utilizadas desde cualquier parte del mismo.
  - **Sección *void setup*():** Delimitada por llaves de apertura y cierre, es donde se define la configuración del programa, se inicializan variables, la comunicación serie y otros parámetros. Es la primera función que se ejecuta en el programa.

- **Sección *void loop*():** Delimitada por llaves de apertura y cierre. Esta función se ejecuta inmediatamente después de *void setup* infinitas veces hasta que la placa se apague (o se resetee).

- Consola de Notificaciones: Es la parte de depuración donde se notifica al programador sobre errores de sintaxis, comunicación, etc.

2.2.3.1.- Librerías

Los ficheros que forman una librería en Arduino son esencialmente dos, con extensión "*.h*" (cabeceras) y con extensión "*.cpp*" (código fuente en C/C++). En el primero se especifican las propiedades y métodos que utilizan, las relaciones de herencia con otras librerías, etc. El segundo es el que contiene la implementación de los métodos y funciones definidos en el fichero "*.h*". Opcionalmente puede incluir un archivo *Keywords.txt*, que contiene las palabras claves que se resaltan en el IDE y una carpeta con programas de ejemplo.

Arduino incluye una serie de librerías oficiales en la instalación del IDE. Además, en Internet se puede encontrar un gran número de librerías desarrolladas por terceros que se adaptan a casi la totalidad de los proyectos.

Para una información más detallada de todo lo relacionado con Arduino (librerías, IDE, software, hardware, documentación asociada, etc.) consultar [10].

2.3.- Asterisk

Asterisk es un software de código abierto, ofrecido por la empresa Digium bajo la licencia GPL. Una vez instalado en un ordenador de propósito general, Asterisk proporciona todas las funcionalidades de centralitas telefónicas como Cisco, Alcatel, Siemens, etc. Asterisk es ampliamente utilizado en entornos empresariales, sin embargo, debido a su flexibilidad también puede utilizarse en entornos domésticos, siendo este sector el de mayor interés para el proyecto.

Algunas de las funcionalidades básicas que provee Asterisk son: desvíos, capturas y transferencias de llamadas, identificación del número llamante, música en espera, autenticación, fax, conexión con líneas de telefonía tradicional, configuración de bases de datos, etc. Otras más avanzadas incluyen buzones de voz, IVR, CDR (*Call Detail Record*), ACD (*Automatic Call Distribution*), sistema de audioconferencias, integraciones con reconocimiento de voz, entre otras.

Además de las características anteriores los administradores del sistema pueden añadir nuevas funcionalidades de varias formas: programando *scripts* en el lenguaje propio de Asterisk, añadiendo módulos personalizados escritos en C o escribiendo *scripts* AGI (*Asterisk Gateway Interface*) en Perl, Bash (*Bourne-again Shell*), PHP (*Hypertext*

Preprocessor) u otros lenguajes de su preferencia. Todo esto convierte a Asterisk en la principal alternativa a día de hoy dentro de las plataformas de telefonía de código abierto.

Originalmente se creó para el sistema operativo GNU/Linux, pero actualmente cuenta con versiones para otros sistemas operativos; aunque Linux, como plataforma nativa, es la más estable y en la que existe un mejor soporte. Por esta razón, se decide instalar la versión en Linux para este proyecto.

2.3.1.- Protocolos de Señalización y Transporte

Un protocolo de señalización tiene como finalidad crear y liberar el canal de comunicación para permitir el paso de la información de usuario cuando se inicia y termina una llamada. SIP (*Session Initiation Protocol*) es uno de estos protocolos.

Dos dispositivos SIP (teléfonos IP) pueden comunicarse directamente, sin embargo, es habitual hacer uso de algunos elementos adicionales llamados *proxies* para facilitar el establecimiento de las llamadas. Un *proxy* (por ejemplo, una centralita telefónica como Asterisk) opera como un intermediario encargándose de negociar entre las dos partes.

Una vez establecida la comunicación, para el intercambio del tráfico de datos entre ambos extremos es necesario un protocolo de transporte que envíe la información desde el origen hacia el destino. En VoIP el protocolo de transporte típico es RTP (*Real-time Transport Protocol*). Este se encarga de fragmentar, re-ensamblar los paquetes y proveer mecanismos para reducir el *jitter*[1], el retardo, etc.

En la Figura 2.20 se muestra el esquema general del proceso de registro entre clientes y el servidor *proxy* y las conversaciones entre ellos. Se puede apreciar que la señalización (SIP) y las conversaciones de voz (RTP) viajan por caminos diferentes.

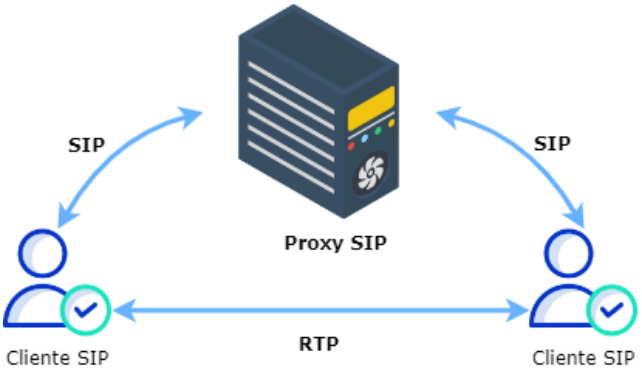


Figura 2.20.- Relación entre los protocolos SIP y RTP.

[1] Variación del tiempo entre la llegada de distintos paquetes.

2.3.1.1.- SIP

La versión más reciente del protocolo SIP se encuentra descrita en el RFC 3261 [11]. Es un protocolo de señalización a nivel de aplicación encargado de la iniciación, modificación y terminación de sesiones multimedia (aplicaciones de mensajería instantánea, vídeos, audios, conferencias, entre otras), las cuales se llevan a cabo de manera interactiva [12]. SIP puede ser transportado sobre TCP (*Transmission Control Protocol*) o sobre UDP (*User Datagram Protocol*).

Según [13], las tareas principales llevadas a cabo por SIP son las siguientes:

1. Autenticación.
2. Negociación de la calidad de una llamada telefónica.
3. Intercambio de las direcciones IP y puertos que se van utilizar para enviar y recibir las comunicaciones.

La premisa de SIP es que cada extremo de una conexión es un par. Es similar a HTTP en que está basado en texto y se asemeja a SMTP (*Simple Mail Transfer Protocol*) en la forma de especificar las direcciones SIP. Estas identifican a un usuario de un determinado dominio. A las direcciones SIP habitualmente se les llama URI (*Uniform Resource Identifier*) y se pueden especificar de la siguiente manera:

sip:usuario@dominio[:port]	*sip:dailenis@uniovi.es[:5060]*
sip:usuario@direcciónIP[:port]	*sip:dailenis@192.168.1.14[:5060]*

El dominio representa el nombre del *proxy* SIP que conoce la dirección IP del terminal identificado por el usuario de dicho dominio. El puerto por defecto para SIP es 5060, aunque si es necesario se puede modificar [12].

Durante una comunicación SIP en una arquitectura P2P (*Peer to Peer*) se intercambian dos tipos de mensajes: Peticiones o métodos SIP y las respuestas SIP. Los métodos SIP más utilizados son los siguientes:

- INVITE: Enviada a un usuario para iniciar la comunicación.
- BYE: Finaliza la comunicación previamente establecida entre usuarios.
- CANCEL: Usada para cancelar una petición antes de que se haya atendido.
- OPTIONS: Solicita información sobre las capacidades de un servidor.
- REGISTER: Peticiones enviadas al servidor para registrar nuevos usuarios.
- ACK: Enviada para confirmar la recepción de otros mensajes dentro de un diálogo SIP.

Cada método o petición SIP tiene asociado una respuesta identificada por un código. El código de la respuesta consta de tres dígitos y están asociadas a grupos de respuestas tales como:

- **1xx:** Respuestas informativas.
- **2xx:** Respuestas de éxito.

- **3xx:** Respuestas de re-direccionamiento.
- **4xx:** Respuestas de error en la petición.
- **5xx:** Respuestas de error en el servidor SIP.
- **6xx:** Respuestas de errores globales.

2.3.1.2.- RTP

Para el intercambio de paquetes entre ambos extremos de la comunicación se requiere de un protocolo de transporte. Los protocolos de transporte de Internet (TCP y UDP) no se diseñaron originalmente pensando en la transmisión de medios en tiempo real. TCP, aunque garantiza la entrega de datos añade un retraso significativo y UDP, aunque consigue que los paquetes lleguen al destino en un lapso de tiempo más corto, no garantiza la entrega.

Por ello, estos mecanismos no sirven en una conversación de voz típica debido a que, si se pierden paquetes, o se produce un retraso superior a 150 milisegundos, habrá dificultades para mantener una conversación clara y fluida.

Teniendo en cuenta estas limitaciones, se crea el protocolo RTP para la transmisión confiable de voz y vídeo en tiempo real a través de Internet. La primera versión fue publicada en 1996 en el documento RFC 1889 [14] y fue reemplazado por el estándar RFC 3550 [15] en 2003.

El protocolo RTP generalmente usa UDP como protocolo de transporte. Para llevar a cabo su función hace uso de números de secuencia, marcas de tiempo, envío de paquetes sin retransmisión, identificación de contenido, sincronización, etc., lo que permite poder continuar con la reproducción del flujo de paquetes en presencia de pérdidas, *jitter* o retardos. Esto quiere decir que, si bien no puede garantizar que la entrega de tráfico se haga exactamente en tiempo real, sí garantiza que se haga de forma sincronizada [12].

2.3.2.- Estructura de Directorios y Ficheros

Cuando se instala Asterisk se crean un gran número de directorios, cada uno con un propósito específico. A continuación, se mencionan dichos directorios y una breve descripción de la función de cada uno:

- **/etc/asterisk/:** El directorio más importante de Asterisk ya que contiene todos los ficheros de configuración. Seguidamente se mencionan los utilizados en este proyecto.
  - ***extensions.conf*:** Contiene la secuencia del *dialplan*.
  - ***pjsip.conf*:** Contiene la configuración permitida para cada uno de los usuarios registrados en el servidor Asterisk usando el protocolo de señalización SIP.

    - ***asterisk.conf*:** Indica la ubicación de los demás directorios. Si se desea, se puede modificar las rutas de los directorios en este fichero.
- **/usr/lib/asterisk/modules/:** Contiene los ficheros binarios de los módulos de Asterisk que han sido compilados. Normalmente no se interactúa con esta carpeta. Sin embargo, ocasionalmente es útil saber dónde se encuentran los módulos, por ejemplo, para borrar módulos incompatibles al actualizar la versión anterior de Asterisk.
- **/var/lib/asterisk/:** Contiene diversos ficheros importantes para Asterisk en distintos subdirectorios.
    - **/var/lib/asterisk/agi-bin/:** Contiene los *scripts* AGI que pueden ser ejecutados desde el *dialplan* con la aplicación AGI.
    - **/var/lib/asterisk/astdb/:** La base de datos de Asterisk guarda información de configuración, de registro de usuarios, etc. Este archivo nunca se modifica a mano, para ello, debe usarse el comando "*database*" desde la línea de comandos de Asterisk.
    - **/var/lib/asterisk/images/:** Contiene imágenes que pueden transmitirse por canales que lo soporten.
    - **/var/lib/asterisk/sounds/:** Este directorio contiene los distintos sonidos que Asterisk es capaz de reproducir. Al utilizar aplicaciones como *Playback* o *Background*, si no se indica la ruta absoluta al fichero, se busca en este directorio.
- **/var/spool/asterisk/:** Contiene diversos subdirectorios, relacionados con la entrada/salida de ficheros. Este directorio se utiliza para almacenar elementos transitorios como mensajes de voz, grabaciones de llamadas, *callfiles*, etc.
    - **/var/spool/asterisk/outgoing/:** Asterisk lee periódicamente este directorio en busca de *call files* que son ficheros que permiten generar llamadas automáticamente.
    - **/var/spool/asterisk/system/:** Si se utiliza la aplicación *System*, Asterisk guarda en esta carpeta los ficheros temporales generados.
    - **/var/spool/asterisk/tmp/:** Contiene ficheros temporales que Asterisk puede necesitar antes de mover un fichero.
    - **/var/spool/asterisk/voicemail/:** Almacena todos los ficheros con los mensajes de los buzones de voz.
- **/var/log/asterisk/:** Contiene varios archivos de registros que son útiles para depurar y además consultar eventos, errores y otros resultados relacionados con el funcionamiento de la centralita.

2.3.3.- Canal *pjsip.conf*

Un canal en Asterisk se establece cuando dos extremos cualesquiera de la conexión se comunican (ver Figura 2.21). Dichos canales pueden ser de tipo DAHDI, IAX, H.323,

SIP, PJSIP. En este proyecto se decide utilizar el canal PJSIP debido a que es el que se ha incorporado en las últimas versiones de Asterisk.

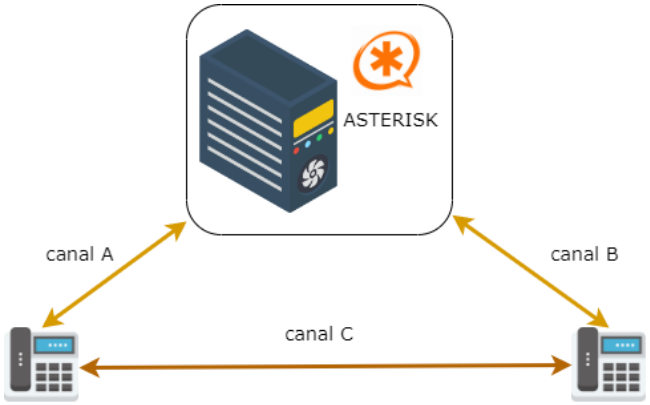


Figura 2.21.- Canales en Asterisk.

En el fichero *pjsip.conf* es donde se define la configuración de un canal PJSIP. No es más que un archivo de texto plano compuesto por secciones. Cada sección se identifica por nombres entre corchetes y tiene una o varias opciones de configuración a las que se les puede asignar un valor. En la Figura 2.22 se observa la sintaxis y un ejemplo del fichero *pjsip.conf*.

[Nombre_de_la_sección]

Opción_de_Configuración = Valor

```
[tipo_de_transporte]
type=transport
protocol=udp ;udp,tcp,tls,ws,wss,flow
bind=0.0.0.0

[Ana]
type=endpoint
context=llamada_interna
disallow=all
allow=ulaw
allow=gsm
transport=tipo_de_transporte
auth=Ana_pass
aors=Ana

[Ana]
type=aor
max_contacts=1

[Ana_pass]
type=auth
auth_type=userpass
password=A137*
username=Ana
```

Figura 2.22.- Ejemplo de fichero *pjsip.conf*.

En la figura anterior se puede observar que cada sección tiene una opción *type* que define el tipo de sección que se está configurando. Existen varios tipos de secciones que

pueden ser consultadas en [16]. A continuación, se describe los cuatro tipos de secciones que se emplean en el fichero *pjsip.conf* del presente proyecto.

- **Transport:** Define la capa de transporte utilizada por PJSIP, admite opciones como TCP, UDP, etc. Si varias extensiones tienen el mismo método de transporte, se puede compartir una sola sección *transport* entre todas ellas.
- **Endpoint:** El identificador de esta sección es el número o el nombre asociado a la extensión. En ella se define el contexto donde se atienden las llamadas efectuadas por esta extensión, los tipos de códecs de voz utilizados y se vincula con las secciones de tipo *transport*, *aor* y *auth*.
- **Aor:** Permite definir el número de extensiones diferentes que se pueden registrar con el mismo nombre de extensión. Usualmente se configura el valor del parámetro *max_contacts* = 1 como se muestra en la figura, pero se puede registrar más de un teléfono IP con el mismo número de extensión.
- **Auth:** Define los parámetros de autenticación de la extensión durante el registro, habitualmente mediante un par usuario y contraseña.

En Asterisk también existe otro tipo de canal denominado local, dicho canal es de gran utilidad ya que permite hacer llamadas internas para ejecutar acciones sin involucrar directamente a los usuarios de la centralita. En este proyecto esta funcionalidad se aprovecha para darle un destino a las llamadas producidas por los *call files* cuando se desea planificar una acción futura. En este caso, no se establece la llamada con el usuario cuando llega el momento de la ejecución de la acción.

2.3.4.- Fichero *extensions.conf*

El plan de marcado o *dialplan* es el corazón de Asterisk. Cuando un usuario efectúa una llamada, el *dialplan* define el flujo de acciones a realizar para manejar dicha llamada. A diferencia de los sistemas telefónicos tradicionales, el *dialplan* de Asterisk es completamente personalizable y se configura en el archivo *extensions.conf*.

El *dialplan* se compone de cuatro partes principales [12] [17] [18]: los contextos, las extensiones, las prioridades y las aplicaciones. En la Figura 2.23 se presenta un ejemplo de un *dialplan* básico, donde se puede visualizar las partes principales.

```
;exten => nombre_ext, prioridad (etiqueta opcional), aplicación

[llamada_interna] ;CONTEXTO

exten => 100, 1, Answer()
same => n(hola), PlayBack(audioHola)
same => n, Hangup()

[otro_contexto]
exten => 100, 1, Answer()
;...
```

Figura 2.23.- Ejemplo de fichero *extensions.conf*.

En la Figura 2.22 se muestra el registro del usuario Ana a la centralita y se le asocia el contexto [llamada_interna]. Según la Figura 2.23, si Ana realiza una llamada a la extensión 100, se responde dicha llamada, se reproduce el audio del fichero "audioHola" y por último se cuelga la llamada. Al estar asociado exclusivamente con el contexto [llamada_interna], el usuario Ana no tiene acceso a la extensión 100 de [otro_contexto].

En los siguientes subapartados se realiza una descripción más detallada de la función de cada una de las partes de un *dialplan*.

2.3.4.1.- Contextos

El *dialplan*, al igual que el fichero *pjsip.conf*, se divide en secciones llamadas contextos y se definen colocando el nombre del contexto entre corchetes. El nombre puede contener letras mayúsculas y minúsculas, números, el guión y el guión bajo.

La función de los contextos es evitar que las diferentes partes del *dialplan* interactúen entre sí, a menos que la interacción sea permitida específicamente. Una extensión definida en un contexto está completamente aislada de las extensiones en cualquier otro contexto. Como se ha visto en el apartado 2.3.3, al definir un usuario en el archivo *pjsip.conf*, se le asocia un contexto, por lo que ese usuario sólo tiene acceso a las extensiones incluidas en su contexto. Esto provee de seguridad al sistema ya que personas no autorizadas no pueden acceder a las extensiones de otros contextos.

Todas las instrucciones colocadas después de la definición de un contexto forman parte de ese contexto, hasta que se defina el siguiente. No debe usarse como nombres de [general], [globals] y [default] ya que son contextos reservados para funciones específicas del *dialplan*.

2.3.4.2.- Extensiones

En Asterisk una extensión es mucho más que un identificador numérico, ya que una misma extensión puede contener una lista de acciones asociadas que son ejecutadas de manera secuencial, por orden de prioridad. Cada acción de esa lista de acciones contiene una aplicación. En la Figura 2.24 se puede apreciar la sintaxis para definir una extensión.

Las extensiones pueden ser de tipo literal, especial o estándar. Las primeras son cualquier combinación de números, letras y los caracteres (* y #). La extensión 1000 definida en la Figura 2.24 es un ejemplo de este tipo de extensión.

Las especiales son muy útiles para reducir el tamaño del *dialplan* y son conocidas como patrones. Cualquier patrón comienza con un guión bajo "_" para que Asterisk entienda que se trata de un patrón. Los siguientes caracteres pueden utilizarse para definir un patrón.

- **X** Cualquier dígito del 0 al 9.

- **Z** Cualquier dígito del 1 al 9.
- **N** Cualquier dígito del 2 al 9.
- **[]** Cualquier dígito que se encuentre entre los corchetes.
- **.** uno o más caracteres, ejemplo _7. implica cualquier número que empiece por 7 seguido de cualquier carácter.
- **!** ninguno o más caracteres, ejemplo 7 o 7 seguido de cualquier carácter.

Para una mejor comprensión de estos patrones, en la Figura 2.24 se muestran varios ejemplos.

```
[llamada_interna]
exten => 1000,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca la extensión 1000

exten => _100X,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca una extensión entre el 1000 y el 1009

exten => _100Z,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca una extensión entre el 1001 y el 1009

exten => _100N,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca una extensión entre el 1002 y el 1009

exten => _10[3 5-7]0,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca una extensión de {1030, 1050, 1060, 1070}

exten => _10.,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca 10 "+" uno o más caracteres

exten => _10!,1,Dial(PJSIP/Eva)
;Se llama al terminal de Eva si Ana marca 10 "+" ninguno o más caracteres
```

Figura 2.24.- Ejemplos de patrones de Asterisk.

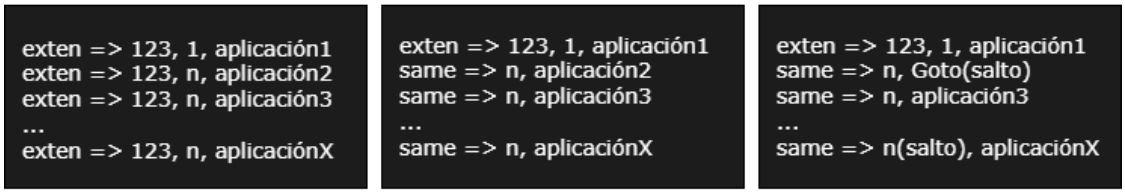
Por último, las extensiones estándares son las que están definidas por defecto, alguna de ellas son las siguientes:

- **s:** Inicio. Se utiliza para atender una llamada cuando no hay un número marcado.
- **t:** Tiempo de espera. Se utiliza para colgar una llamada cuando se agota el tiempo de espera.
- **h:** Colgar. Se llama después de que el usuario desconecta la llamada.
- **i:** Inválido. Se activa cuando se marca una extensión inexistente en el contexto del *dialplan*.

2.3.4.3.- Prioridades

Cada extensión puede tener varias acciones, llamadas prioridades. Las prioridades se numeran secuencialmente, empezando por el 1, y cada una ejecuta una aplicación específica. Además, las prioridades deben estar bien enumeradas, pues si falta alguna intermedia Asterisk aborta la ejecución de la llamada.

En la Figura 2.25 se presentan tres ejemplos de algunas simplificaciones que introduce Asterisk para evitar errores de enumeración cuando una extensión tiene muchas prioridades.



(a) (b) (c)

Figura 2.25.- Ejemplos de usos de las prioridades de Asterisk.

En el recuadro (a) se introduce la prioridad "*n*" que significa "siguiente". Internamente, Asterisk cada vez que encuentra una prioridad n, calcula la siguiente prioridad, tomando la anterior y sumándole 1. En el recuadro (b) se introduce el uso del operador "*same*", lo que evita reescribir la extensión completa mientras esta sea la misma. En el recuadro (c) aparece el uso de las etiquetas (*labels*). Esto soluciona el problema que aparece al sustituir las prioridades numéricas por la prioridad "n". Si se desea realizar un salto a una zona concreta del dialplan hay que etiquetar esa prioridad tal como se muestra en la figura.

2.3.4.4.- Aplicaciones

Las aplicaciones son una parte esencial del *dialplan*. Cada aplicación realiza una acción específica en el canal actual, como reproducir un sonido, buscar en una base de datos, colgar una llamada, etc. Algunas aplicaciones no necesitan que se le pase ningún argumento para realizar su función, ellas por sí solas ya tienen predefinido lo que tienen que hacer. Otras, sin embargo, requieren de información adicional que se le pasa como argumentos separados por comas, y en función de esos argumentos se afecta la forma en que realiza las acciones.

Algunas aplicaciones básicas son:

- ***Answer()*:** Responde a un canal que está sonando. Esta aplicación no toma argumentos. No siempre es necesaria, pero es una forma efectiva de asegurar que un canal está conectado antes de realizar otras acciones.
- ***Hangup()*:** Cuelga el canal de llamada. Se suele usar sin pasarle ningún argumento.
- ***NoOp()*:** No hace nada, generalmente se utiliza como una herramienta de depuración. El texto que se pase como parámetro se imprime en la consola de Asterisk, no es necesario poner el texto entre comillas.

- ***Playback()*:** Reproduce un archivo de sonido pregrabado a través de un canal. Las entradas DTMF (*Dual Tone Multi Frequency*) del usuario son ignoradas. Para utilizar *Playback*, hay que especificar el nombre de archivo sin la extensión como argumento, Por ejemplo, *Playback*(audio) reproduce el archivo de sonido llamado audio.wav, asumiendo que se encuentra en el directorio de sonidos por defecto (normalmente /var/lib/asterisk/sounds).

Otros ejemplos de aplicaciones con un modo de operación más avanzado son las siguientes:

- ***Goto()*:** Envía una llamada a otra parte específica del *dialplan*. Requiere que se pasen como argumentos el contexto de destino, la extensión y la prioridad. Si la zona del *dialplan* donde se desea realizar el salto está en el mismo contexto, se pasa como argumentos extensión y prioridad. Si se encuentra en la misma extensión solo se le pasa la prioridad. La sintaxis es la siguiente:

  Goto(contexto, extensión, prioridad)

  Goto(extensión, prioridad)

  Goto(prioridad)

- ***GotoIf()*:** Esta aplicación hace que la ejecución del *dialplan* continúe en la ubicación especificada por las etiquetas, según el resultado de la evaluación de la condición dada. La sintaxis es la siguiente:

  GotoIf (condición? LabelifTRUE:LabelifFALSE)

  Envía la llamada a la etiqueta *LabelifTRUE* si la condición es verdadera o a *LabelifFALSE* si la condición es falsa. Cualquiera de estas etiquetas puede ser omitida, pero nunca las dos a la vez. En caso de que no se utilice una etiqueta es necesario especificar [contexto, extensión, prioridad].
- ***Gosub()*:** El propósito de esta aplicación es llamar a un bloque de funcionalidad del *dialplan*, al cual se le puede pasar argumentos y luego regresa a la misma dirección desde donde fue llamada la subrutina (opcionalmente con un valor de retorno). La sintaxis es:

  GoSub(contexto, extensión, prioridad [arg1,…, argN])

- ***Background()*:** Al igual que *Playback*, reproduce un archivo de sonido grabado, pero cuando el usuario que está llamando presiona una tecla en su teléfono, interrumpe la reproducción y pasa la llamada a la extensión que corresponde con el dígito o dígitos pulsados.
- **WaitExten():** Esta aplicación espera a que el usuario introduzca dígitos DTMF durante un número específico de segundos que se le pasa como argumento. Se utiliza directamente después de la aplicación *Background*.

El catálogo completo de aplicaciones disponibles en Asterisk se encuentra en [19].

2.3.5.- AGI (*Asterisk Gateway Interface*)

El *dialplan* de Asterisk ha evolucionado hasta convertirse en una interfaz de programación sencilla y potente para la gestión de llamadas. Sin embargo, en ocasiones es necesario extender las funcionalidades de Asterisk integrándolo con otros sistemas, como un Arduino, por ejemplo. Es aquí donde AGI juega su papel.

AGI [20] es una interfaz de entrada a Asterisk que permite al desarrollador el uso de lenguajes de alto nivel como Perl, PHP, Python, Bash para realizar el control de llamadas del *dialplan*, ya sea porque se tiene gran experiencia en dichos lenguajes de programación o porque se quiere utilizar códigos ya existentes para integrar otros sistemas con Asterisk.

Todos los *scripts* AGI deben estar ubicados en /var/lib/asterisk/agi-bin ya que Asterisk espera encontrarlos en ese directorio cuando son ejecutados. Esta aplicación también admite que se le pasen varios argumentos separados por coma.

2.3.6.- Variables

Las variables de Asterisk se pueden utilizar para añadir lógica y dinamismo al *dialplan*. Una variable es un contenedor con nombre que tiene asociado un valor. Su contenido puede cambiar, pero su nombre no, lo que significa que siempre se puede referenciar el nombre de la variable, sin importar el valor que tenga en ese momento.

Hay tres tipos de variables que se pueden usar en un *dialplan*: variables globales, de canal y de entorno. Las globales son visibles para todos los canales en todo momento, deben ser declaradas en el contexto [globals] al principio del *dialplan* y las variables de entorno son una forma de acceder desde Asterisk a las variables de entorno de Unix.

En cambio, las variables de canal están asociadas únicamente a una llamada en particular. A diferencia de las variables globales, se definen sólo para la duración de la llamada actual y están disponibles solamente para los canales que participan en esa llamada. Las variables de canal se establecen con la aplicación *Set*. Seguidamente, se propone un ejemplo donde se inicializa una variable con un valor numérico y a continuación se reproduce la locución del valor almacenado en dicha variable.

```
exten => 123, 1, Set(variable=40)
same => n, SayNumber(${variable})
```

2.3.7.- *Call File*

Un archivo de llamada, o *call file*, es un archivo de texto con una sintaxis específica que es muy útil para planificar una llamada futura. Esta funcionalidad es fundamental para el desarrollo de este proyecto.

Una vez creado el archivo basado en pares <clave>:<valor>, se mueve al directorio de cola /var/spool/asterisk/outgoing. Como Asterisk sondea este directorio cada segundo, cuando detecta un nuevo archivo automáticamente procesa la llamada según el contenido de este archivo. En este proyecto la tarea de crear los *call files* se ha automatizado mediante la programación de un *script* en Bash. En la Figura 2.26 se muestra un ejemplo de un *call file* que ya ha sido ejecutado correctamente.

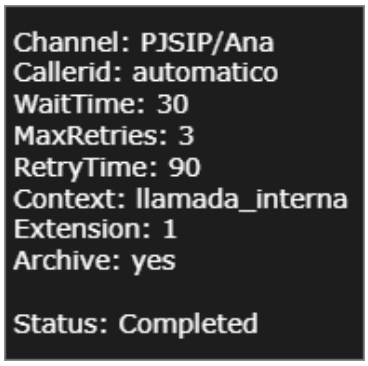


Figura 2.26.- Ejemplo de un *call file* completado.

A continuación, se describe las opciones que se pueden especificar en un *call file*:

- **Channel:** El canal que se utiliza para la nueva llamada, en la forma Tecnología/Recurso como en la aplicación Dial. Este valor es obligatorio.
- **Callerid:** El identificador de llamadas que se utiliza.
- **WaitTime:** Especifica el tiempo de espera en segundos para una respuesta antes de abandonar una llamada saliente. El valor predeterminado es 45.
- **MaxRetries:** Especifica cuántas veces Asterisk reintenta la llamada antes de abortarla. Si no se especifica, el valor por defecto es 0.
- **RetryTime:** Periodo de tiempo de espera antes de reintentar una nueva llamada. El valor predeterminado es 300 (5 minutos).
- **Archive:** Los *call files* se eliminan inmediatamente después de su ejecución de forma predeterminada. Si se define esta opción con *yes* se almacena en el directorio /var/spool/asterisk/outgoing_done/ y Asterisk le agrega una línea donde pone el estado del *call file*, es decir, si ha caducado, se ha completado o ha dado error. Este mensaje es útil en tareas de comprobación.

Cuando se responde la llamada, hay dos opciones: ejecutar una sola aplicación, o ejecutar el *dialplan* especificando el contexto, la extensión y la prioridad.

Para ejecutar una aplicación:

- **Application:** La aplicación a ejecutar.
- **Data:** Los argumentos de la aplicación.

Para comenzar a ejecutar aplicaciones en el *dialplan*:

- **Context:** El contexto en el *dialplan*.

- **Extension:** La extensión en el contexto especificado.
- **Priority:** La prioridad de la extensión especificada (numérico o etiqueta).
- **Setvar:** También puede asignar valores a las variables que están disponibles para el canal, como si hubiera realizado un par (var = valor) en el *dialplan*.

Para más información de los *call files* de Asterisk, se puede consultar [21].

2.3.8.- IVR (*Interactive Voice Response*)

Un IVR es una tecnología de telefonía automatizada que permite la interacción de los usuarios con el sistema sin la intervención de un operador. Estos sistemas orientan al usuario mediante audios pregrabados, sobre las distintas opciones preestablecidas que existen en el menú. El usuario selecciona una de las opciones mediante entradas DTMF y el sistema ejecuta lo que esté definido en el *dialplan* para ese dígito o dígitos.

En los últimos años se han incorporado tecnologías más avanzadas en los sistemas IVR, donde los usuarios pueden responder a una indicación u ordenar algo a través de la voz. Esta alternativa resulta interesante para el desarrollo de este proyecto ya que es más fluida e intuitiva que los sistemas por teclado DTMF. Sin embargo, los sistemas DTFM aportan rapidez a la interacción ya que, si el usuario conoce de antemano la combinación de dígitos que debe marcar, se interrumpe la reproducción del audio pregrabado y el *dialplan* continúa la ejecución. Por el contrario, si no la conoce tiene que escuchar el audio completo para saber las indicaciones, mientras que la interacción por voz puede resultar más intuitiva en ese sentido. En la Figura 2.27 se muestra dos ejemplos básicos de un IVR por DTMF y otro por reconocimiento de voz.

```
[home]
exten => 100,1,Answer()
same => n(bienvenida),BackGround(es/bienvenida)
same => n,WaitExten(7)

exten => 1,1,NoOp(No 1 pulsado. Acción
Inmediata)
same => n,Goto(inmediata-ivr,s,1)

exten => 2,1,NoOp(No 2 pulsado. Acción Futura)
same => n,Goto(futura-ivr,s,1)

exten => i,1,NoOp(Número inválido)
exten => i,n,Goto(100,bienvenida)

exten => t,1,NoOp(Agotado tiempo espera)
exten => t,n,Goto(100,bienvenida)

[inmediata-ivr]
exten => s,1,BackGround(es/acciones)
same => n,WaitExten(7)

exten => 1,1,NoOp(No 1 pulsado. Encender)
same => n,Goto(encender-ivr,s,1)
;...

[futura-ivr]
;...
```

(a)

```
[home]
exten => 100,1,Answer()
same => n(bienvenida),Playback(custom/bienvenida)
same => n,Gosub(google_stt,s,1)
same => n,Set(textoGoogle=${utterance})
same => n,GotoIf($["${textoGoogle}" = "ahora"]?inmediata-ivr,s,1)
same => n,GotoIf($["${textoGoogle}" = "después"]?futura-ivr,s,1)
same => n,Playback(custom/invalid)
same => n, Goto(bienvenida)

[inmediata-ivr]
exten => s,1,NoOP(IVR de acciones inmediatas)
same => n(inmediata),Playback(custom/action)
same => n,Gosub(google_stt,s,1)
same => n,Set(accionGoogle=${utterance})
same => n,GotoIf($["${accionGoogle}" = "encender luz"]?
acciones,1,1)
same => n,GotoIf($["${accionGoogle}" = "apagar luz"]?
acciones,2,1)

;...

[futura-ivr]
exten => s,1,NoOP(IVR de acciones futuras)
;...
```

(b)

Figura 2.27.- Ejemplo de un IVR: (a) DTMF, (b) Reconocimiento de voz.

Tomando como referencia el ejemplo de la Figura 2.27a, el funcionamiento es el siguiente: cuando un usuario llama a la extensión 100 se reproduce un *background* que da la bienvenida e indica al usuario las dos posibles opciones disponibles. Si marca la tecla 1 el IVR se dirige al contexto [inmediata-ivr], si marca la tecla 2 se dirige al contexto [futura-ivr] para continuar la interacción con el menú. Si marca otra tecla diferente se activa la extensión "*i*" y si se agota el tiempo de espera de una entrada DTMF se activa la extensión "*t*" regresando en ambos casos a la línea del *dialplan* etiquetada como "bienvenida".

El flujo del ejemplo de la Figura 2.27b es un poco diferente: cuando un usuario llama a la extensión 100 se reproduce el fichero de audio suministrado como parámetro a la aplicación *background*, el cual da las indicaciones al usuario. Este responde por voz en lugar de presionar una tecla del teléfono. En ese instante se llama a la subrutina que se conecta con los servicios de STT de Google. Si el resultado de la traducción es "ahora" el IVR se dirige al contexto [inmediata-ivr], si la traducción es "después" se dirige al contexto [futura-ivr] para continuar navegando por los distintos contextos del *dialplan* en función de las órdenes de voz del usuario.

2.4.- *Softphone*

Según [17] existen tres tipos de puntos finales que se utilizan para establecer la comunicación de los usuarios a través de la red. Se les conoce como *hardphones* (dispositivo físico), *softphones* y adaptadores de terminales analógicos (ATA), siendo los *softphones* los de mayor interés para este proyecto.

Un *softphone* es una aplicación de software que proporciona funcionalidad telefónica en un ordenador portátil, de escritorio, un teléfono inteligente u otro dispositivo informático. El audio debe pasar por el sistema de sonido de estos dispositivos. Actualmente se espera que la interfaz de los *softphones* tenga el mismo aspecto y sonido que los teléfonos tradicionales, aunque esta concepción puede cambiar en un futuro muy cercano.

Sin embargo, para el presente trabajo, se define un *softphone* como cualquier dispositivo que se ejecuta en un ordenador personal, presenta la apariencia de un teléfono y ofrece como función principal la capacidad de realizar y recibir comunicaciones de audio *full-duplex*, antes conocidas como llamadas telefónicas.

2.4.1.- Zoiper

Zoiper es un software multiplataforma que ofrece llamadas de voz, vídeo y mensajería instantánea, disponible para los sistemas operativos Windows, MacOS y Linux, además de iOS y Android. Está diseñado para trabajar con sistemas de comunicación IP

basado en el protocolo SIP. Este software dispone de una versión gratuita y otra comercial con más funcionalidades habilitadas.

La versión gratuita *Free* Zoiper5 permite hacer o recibir hasta dos llamadas de manera simultánea, iniciar y conducir videoconferencias, proporciona la opción de llamada en espera y también facilita la transferencia de llamadas, dichas funcionalidades son suficientes para llevar a cabo las tareas de este proyecto.

2.5.- Servidor HTTP Apache

Apache es un servidor web gratuito y libre, que cuenta con una gran cantidad de desarrolladores voluntarios que colaboran a través de Internet en su código fuente, configuraciones, parches de seguridad y en la documentación relacionada con este.

La arquitectura utilizada es cliente/servidor, es decir, el equipo cliente inicia una solicitud al equipo servidor, y este responde con el recurso solicitado. El servidor y el cliente se comunican a través del protocolo HTTP utilizando el puerto 80 por defecto para escuchar peticiones y aceptar las conexiones.

También, el servidor Apache tiene las siguientes ventajas [22]:

- **Flexible y eficiente:** Es capaz de trabajar con el estándar HTTP/1.1 y con la mayor parte de las extensiones web que existen en la actualidad, como los módulos PHP, SSL (*Secure Sockets Layer*), SSI (*Server Side Includes*), proxy, etc. Además, mantiene la compatibilidad con HTTP/1.0.
- **Extensible y personalizable:** Como su arquitectura es modular, permite a los administradores del servidor activar y desactivar funcionalidades adicionales.
- **Multiplataforma:** Disponible para diferentes plataformas como GNU/Linux, Windows, MacOS.
- **Licencia:** Es de código abierto lo que permite el uso comercial y no comercial.

La función del servidor Apache en este proyecto es únicamente actuar como un intermediario de la comunicación, cuando el hogar inteligente (Arduino) precisa notificar algunos eventos al usuario.

2.6.- Programación en *Shell*: Bash

Shell es el término usado en informática para referirse a un intérprete de comandos. Actualmente, Bash es el *shell* predeterminado de los sistemas operativos basados en el *kernel* GNU/Linux, aunque también proporciona otros *shells*.

Bash ejecuta línea por línea las instrucciones introducidas por el usuario o contenidas en un *script* y devuelve los resultados. Su función es proporcionar una interfaz en modo texto entre los usuarios o programas y el núcleo (*kernel*) para que este ejecute las operaciones.

Cuando se escriben comandos directamente en un terminal de Linux se está usando Bash a nivel básico. Sin embargo, para automatizar tareas repetitivas es necesario utilizar Bash *scripting*, que consiste en emplear las instrucciones de bash en un *script* usando expresiones matemáticas, condicionales, bucles, puertas lógicas, variables, etc.

Para comenzar a programar un *script* hay que poner en la primera línea la ruta y el intérprete que se utiliza. Bash por defecto está en la carpeta /bin del sistema operativo, por tanto la primera línea de los *scripts* en Bash debe ser #!/bin/bash.

Para mayor información de este intérprete consultar el manual de referencia [23].

2.7.- Telegram

Telegram es una aplicación gratuita de mensajería instantánea y llamadas VoIP, disponible para Android, iOS y que además permite el acceso vía web. Una particularidad de Telegram es la posibilidad de interactuar con los *bots* que vienen integrados en sus grupos.

Esta característica es la que aporta interés para el presente proyecto, ya que se puede hacer uso de dichos *bots* creados por terceros o se pueden crear nuevos *bots* que respondan a las necesidades particulares de un proyecto, siendo este último el caso en cuestión.

Se ha creado e integrado un *bots* al proyecto, con el propósito de notificar al usuario de los eventos ocurridos en el hogar inteligente. Para más información sobre la integración de esta tecnología en este trabajo, se puede consultar el manual del instalador (ver Anexo 10.1.5).

2.8.- API *Speech to Text*

Speech to Text es la API de Google que permite transcribir el contenido de voz en formato de texto.

Esta API cuenta con un porcentaje de precisión elevado respecto a otras alternativas, ya que aplica los algoritmos de la red neuronal de aprendizaje profundo más avanzado de Google para reconocer la voz automáticamente. Asimismo, admite más de 125 idiomas y variantes lingüísticas.

Entre las características más relevantes de esta API, se pueden citar las siguientes:

- Procesa las señales de audio captadas directamente por el micrófono en tiempo real o puede procesar un archivo de audio previamente grabado.
- La tecnología avanzada de *Google Cloud* ayuda a reconocer términos específicos de dominio, convirtiendo automáticamente los números dichos de viva voz en años, horas, monedas, direcciones y otros tipos de clase.
- Implementa mecanismos de cancelación del ruido, siendo capaz de procesar archivos de audio de multitud de entornos ruidosos.
- Permite el filtrado de contenido inadecuado o poco profesional en los datos de audio y ayuda a descartar las palabras inapropiadas en los resultados de texto.
- Proporciona una interfaz de usuario fácil de usar para experimentar con audio de voz y permite una integración sencilla con los proyectos de los desarrolladores.

Sin embargo, todo no son ventajas ya que la API STT de Google no es del todo gratuita. El precio por usar esta API se determina cada mes según la cantidad de audio procesado correctamente por el servicio, y se mide en incrementos redondeados a los siguientes 15 segundos. Las solicitudes que produzcan un error no cuentan como procesadas correctamente y, por tanto, no tienen coste alguno. Las que devuelvan una respuesta, significa que el audio se ha procesado correctamente.

Por ejemplo, el uso en teléfonos, *tablets*, portátiles u ordenadores, es gratuito el reconocimiento de voz para audios de menos de 60 minutos. Para transcripciones de audio de más de 60 minutos y hasta un millón de minutos al mes, cuesta 0,006 dólares por 15 segundos. Para el uso y la autorización en dispositivos integrados como coches, televisores, electrodomésticos o altavoces, es necesario ponerse en contacto con la compañía.

Para mayor información y para utilizar este servicio de Google *Cloud* acceder a [24].

3.- IoT EN LA DOMÓTICA. ESTADO DEL ARTE

La tendencia actual es la conexión a Internet de "cualquier cosa". A esta tendencia se le conoce como IoT y tiene un amplio espectro de aplicaciones, entre ellas la domótica. El objetivo de la domótica es cubrir necesidades en el hogar como la seguridad, el ahorro, la comunicación y el confort de sus habitantes. En este capítulo, se aborda el estado del arte actual de la tecnología IoT aplicada al ámbito doméstico.

3.1.- Investigaciones Científicas

Desde hace varios años se han publicado investigaciones de carácter científico para controlar distintos dispositivos en el hogar integrando tecnologías de código abierto. A continuación, se mencionan algunos de los trabajos más destacados según las fuentes bibliográficas consultadas.

En Potamitis et al. [25], se lleva a cabo un asistente electrónico de voz instalado en un ordenador que puede gestionar la interacción de un interlocutor con sus dispositivos de entretenimiento y electrodomésticos. Los retos a los que se enfrenta el *front-end* del sistema están relacionados con la adversidad acústica de los entornos domésticos que utilizan micrófonos a distancia (en este caso manos libres). El asistente es capaz de recibir órdenes e interactuar con el interlocutor mientras camina en presencia de interferencias de altavoces y música que compiten entre sí, así como de ruidos estacionarios procedentes de los electrodomésticos. La naturaleza de la aplicación implica que el motor de reconocimiento esté activo continuamente, sin embargo, el localizador de palabras excluye el mando accidental de los aparatos por interlocutores de fondo, mediante la búsqueda constante de una palabra específica. Mientras no se detecte la palabra clave, no se permite la propagación de la señal al módulo de reconocimiento de voz.

En Deka et al. [26], se diseña y desarrolla una plataforma IVR basada en Asterisk que permite a los habitantes acceder y controlar el estado de sus electrodomésticos de forma remota. En este prototipo hay un teléfono móvil conectado por *Bluetooth* al sistema informático que actúa como servidor Asterisk. Por tanto, los habitantes hacen una llamada al teléfono móvil con *Bluetooth* y es redirigida al servidor para ofrecer la plataforma basada en IVR, a través del cual se eligen las acciones a realizar sobre los aparatos. Se implementa un mecanismo de autenticación basado en un PIN de usuario, el cual es almacenado, junto con otra información, en una base de datos del servidor, de modo que sólo los usuarios registrados puedan acceder al sistema. Una vez que un usuario registrado introduce sus comandos a través de la plataforma IVR, el sistema envía los comandos al

microcontrolador 8051 utilizando la comunicación a través del puerto serie y se realizan las acciones en consecuencia sobre los actuadores.

Dias et al. [27], describe una plataforma multiusuario que permite el control remoto y concurrente de sistemas de automatización usando un Arduino como controlador principal y como interfaz de usuario un IVR de Asterisk. El funcionamiento de la plataforma es el siguiente: el usuario inicia una llamada dirigida al servidor Asterisk utilizando un teléfono IP (hardware o software), se autentica ante la centralita con un par usuario-contraseña y accede al menú IVR seleccionando uno de los dispositivos previamente configurados a través de DTMF. La elección del usuario se dirige a una acción, según lo que se haya configurado en el *dialplan*. En respuesta a lo anterior se ejecuta un *script* PHP a través de AGI que envía comandos al Arduino, y este a su vez envía señales de control a los dispositivos finales. Para validar la plataforma se realizan dos experimentos de laboratorio. En el primero, el usuario realiza una llamada telefónica al servidor Asterisk y elige una de las opciones a través del menú IVR. El segundo consiste en un sistema de riego temporizado, donde el usuario determina el tiempo en minutos que el sistema debe permanecer activo.

Melo et al. [28], proponen otro enfoque desde una perspectiva de un Edificio Inteligente. El proyecto presenta un esquema de calentamiento de agua inteligente, controlado por tecnologías IoT, para grandes edificios en entornos urbanos. El objetivo principal es garantizar el mínimo desperdicio de agua, así como el ahorro de energía en los sistemas de bombeo y calefacción de edificios multifamiliares. El sistema consiste en un servidor en la nube que ejecuta una PBX (*Private Branch Exchange*) basada en Asterisk y un servicio web que se conecta mediante una aplicación móvil. Se utiliza un temporizador programable para establecer los periodos del día en los que es más probable que se utilice el agua caliente. Fuera de estos periodos los usuarios pueden activar la recirculación mediante una llamada telefónica a un IVR, o a través de una aplicación móvil.

En Hernández et al. [1], presenta un proyecto que proporciona la base para buscar una alternativa que facilite el control y monitorización de luces y dispositivos electrónicos a través de una llamada telefónica, mediante el uso de una centralita IP. Para ello, se utilizan soluciones tecnológicas como Raspberry y Arduino, utilizando la primera como servidor que aloja una centralita basada en Asterisk y la segunda como entidad de gestión del encendido/ apagado de luces y dispositivos que el usuario desea controlar. El funcionamiento del prototipo consiste en que el usuario realice una llamada desde cualquier lugar, a través de Internet, conectándose a una centralita que le indica los dispositivos gestionados. Además, le permite seleccionar un número para el control de encendido, apagado o regulación de los mismos, cuyo servidor se conecta a una base de datos que es administrada por una Raspberry Pi 3 para conectarse al Arduino y ejecutar físicamente la acción deseada.

Spahic et al. [2], expone un sistema simplificado de hogar inteligente, que utiliza tecnologías de código abierto para la transmisión, el reconocimiento y el análisis de los comandos basados en la voz. La contribución de este trabajo consiste en la modificación de

las soluciones existentes aplicadas para el idioma inglés para que funcionen para los idiomas eslavos del sur. El sistema está implementado en la plataforma Raspberry Pi, que funciona como centralita privada basada en Asterisk y a su vez sirve como pasarela IoT, proporcionando el control sobre los dispositivos y electrodomésticos del hogar. El usuario hace una llamada a la casa inteligente, contesta el IVR que le pide autenticarse con una frase de seguridad, y en caso de ser correcta continúa la interacción entre el usuario y el IVR. En cada llamada, Asterisk graba el discurso del usuario, y la aplicación IVR lo transmite a servicios externos (*Speech to Text*, *Dialogflow*) que procesan el discurso adquirido, lo convierten en texto y luego recogen la intención del usuario para ser ejecutada inmediatamente.

3.2.- Investigaciones Docentes

Por otra parte, también existen varios trabajos docentes a los que se considera oportuno hacer referencia debido a la similitud con la temática de estudio de este trabajo.

En Rueda et al. [29], se implementa un diseño de un sistema domótico apoyado en las tecnologías libres de Asterisk y Arduino, para el desarrollo de una alternativa en la seguridad del hogar. Este sistema funciona integrando Asterisk y Arduino, donde este último se encuentra monitoreando dos sensores ubicados dentro de una vivienda. El primero es un sensor infrarrojo para detectar movimientos y el segundo es un sensor magnético para supervisar la apertura de una puerta, ventana, cajón, etc. Estos sensores envían una señal de activación a los pines de entrada de Arduino, el cual reacciona enviando una señal a través de la red LAN al servidor de Asterisk, que se encarga de realizar automáticamente una llamada *Ethernet* informando al usuario del evento producido. El usuario para recibir la alerta enviada por el sistema, debe llevar consigo el *softphone* con acceso a Internet, que puede estar ejecutado en el ordenador del usuario o en su dispositivo móvil. El usuario puede activar un electroimán o actuador marcando una tecla del *softphone* con la finalidad de cerrar una puerta definitivamente para dejar encerrado al intruso.

Barahona et al. [30], desarrollan un sistema que permite acceder a un menú IVR, a través de una llamada desde un *smartphone* Zoiper a una extensión definida en la PBX de Asterisk instalada en una Raspberry Pi. Los mensajes pregrabados en el IVR anuncian las opciones de interacción con el sistema domótico y el usuario selecciona a través del teclado del *softphone* la tarea deseada, controlando así el encendido y apagado de luces, aparatos eléctricos dentro del hogar desde cualquier lugar con acceso a Internet. Se utiliza la librería Asterisk-Java para establecer un puente de comunicación entre la centralita y el código escrito en Java que interpreta las órdenes del usuario y activa o desactiva los puertos GPIO (*General Purpose Input/Output*) de la Raspberry Pi donde está conectada una tarjeta de relés para las luces y aparatos eléctricos.

Veintimilla et al. [31], confeccionan un modelo que puede resultar de utillidad para personas con movimientos limitados ya que el control depende en su mayoría de la voz. El prototipo es de bajo coste y permite controlar los actuadores colocados en los diferentes ambientes de una maqueta de un hogar como son el baño, el comedor, el dormitorio y el salón, por medio de una aplicación web desde el navegador. El navegador recibe los comandos de voz, los convierte a texto, los compara con órdenes previamente establecidas y controla los eventos como: encendido/apagado de luces, apertura/cierre de puertas y ventanas, encendido/apagado del ventilador o calefactor.

Tomala [32], presenta un prototipo similar a la referencia anterior, mostrando un sistema controlado por voz para minimizar las dificultades de personas con discapacidad en sus extremidades superiores a la hora de realizar tareas cotidianas. El módulo central de este proyecto es una tarjeta Raspberry Pi 3, conectada a diversos actuadores tales como el seguro de puerta eléctrica o cerradura eléctrica, electroválvulas para manejo de ducha y lavadero, sistema de iluminación para el salón, comedor, dormitorio, cocina y baño. En este módulo se instala un servidor basado en LAMP (Linux, Apache, MySQL y PHP). El usuario debe efectuar las órdenes por medio de un micrófono inalámbrico, que las envía a la tarjeta que contiene una base de datos para comparar si la orden está contemplada o no, y luego se envía a la librería '*annyang'* la cual transcribe las pronunciaciones a texto para posteriormente ejecutar la acción correspondiente.

3.3.- Conclusiones Parciales

Después de una amplia revisión bibliográfica sobre el tema en cuestión se puede apreciar que, si bien existen estudios a nivel internacional que han analizado e implementado la integración de tecnologías libres como Arduino y Asterisk en el área de la domótica, en el ámbito nacional y local no está ampliamente difundido. Además, Se considera que las principales contribuciones de este proyecto son las siguientes:

La integración de la API STT de Google con la centralita de Asterisk mejora algunas de las soluciones propuestas ya que la precisión de las traducciones de la API de Google es superior a la de otras herramientas empleadas en las investigaciones analizadas.

El uso de protocolos como IP y HTTP hacen que sea posible técnicamente configurar la centralita para tener acceso a ella a través de Internet. Algunas de las soluciones analizadas están limitadas a la presencia física del usuario en el hogar para poder realizar el control sobre los dispositivos.

La alternativa propuesta además de responder a las órdenes dictadas por el usuario, es capaz de realizar acciones automáticas en respuesta a ciertos eventos del hogar.

Como inconvenientes de la solución que se propone, cabe destacar que la API es de pago a partir de los 60 minutos de audios procesados al mes y requiere de una conexión permanente a Internet.

4.- REQUISITOS DEL SISTEMA

En este capítulo se detallan los requisitos software del sistema de control de un hogar domotizado a través de Asterisk. Estos se dividen en dos tipos: los requisitos funcionales que definen lo que puede hacer el sistema, y los requisitos no funcionales que es lo que requiere el sistema para poder funcionar.

Cada uno de los requisitos está definido por los siguientes campos:

1. **ID:** Identificador único del requisito. Presenta la siguiente sintaxis:
   - RF-XX para requisitos funcionales
   - RNF-XX para requisitos no funcionales

   Siendo XX el número secuencial del requisito.
2. **Nombre:** Nombre del requisito.
3. **Descripción:** Breve descripción del requisito.
4. **Prioridad:** Indica la prioridad asignada a cada requisito, esta puede ser alta (A), media (M) o baja (B).

4.1.- Requisitos Funcionales

En la Tabla 4.1 se especifican los requisitos funcionales, donde se describen el conjunto de acciones que debe realizar el sistema:

REQUISITOS FUNCIONALES			
ID	**Nombre**	**Descripción**	**Prioridad**
Acciones Genéricas			
RF01	Reproducir audio explicativo	El sistema reproduce un audio que va guiando al usuario durante la navegación dentro del IVR.	A
RF02	Reconocimiento de voz	El sistema traduce cada comando de voz dicho por el usuario en texto, para procesarlo y permitirle que pueda navegar por el IVR.	A
RF03	Reproducir audio de error	El sistema reproduce un audio de "comando inválido" cuando el usuario no dice correctamente alguna de las acciones predefinidas en el prototipo.	A
RF04	Reproducir audio de silencio	El sistema reproduce un audio de "respuesta vacía" cuando el usuario no responde a una indicación del IVR.	A
RF05	Verificar el reconocimiento de voz	El sistema permite que el usuario pueda comprobar el reconocimiento de su voz.	B

RF06	Notificar resultado de la verificación	El sistema notifica al usuario el resultado de la traducción del reconocimiento de la voz y la probabilidad de que la traducción haya sido correcta.	B
Acciones Inmediatas			
RF07	Encender luces	El sistema permite encender las luces del hogar mediante la orden oral “encender luz”.	A
RF08	Apagar luces	El sistema permite apagar las luces del hogar mediante la orden oral “apagar luz”.	A
RF09	Encender luces de navidad	El sistema permite encender las luces de navidad del hogar mediante la orden oral “encender luces navideñas”.	M
RF10	Apagar luces de navidad	El sistema permite apagar las luces de navidad del hogar mediante la orden oral “apagar luces navideñas”.	M
RF11	Encender calefactor temperatura baja	El sistema permite encender el calefactor del hogar en temperatura baja mediante la orden oral “encender calefactor en uno”.	A
RF12	Encender calefactor temperatura media	El sistema permite encender el calefactor del hogar en temperatura media mediante la orden oral “encender calefactor en dos”.	A
RF13	Encender calefactor temperatura alta	El sistema permite encender el calefactor del hogar en temperatura alta mediante la orden oral “encender calefactor en tres”.	A
RF14	Apagar calefactor	El sistema permite apagar el calefactor del hogar mediante la orden oral “apagar calefactor”.	A
RF15	Encender aire acondicionado intensidad baja	El sistema permite encender el aire acondicionado del hogar en intensidad baja mediante la orden oral “encender aire en uno”.	A
RF16	Encender aire acondicionado intensidad media	El sistema permite encender el aire acondicionado del hogar en intensidad media la orden oral “encender aire en dos”.	A
RF17	Encender aire acondicionado intensidad alta	El sistema permite encender el aire acondicionado del hogar en intensidad alta mediante la orden oral “encender aire en tres”.	A
RF18	Apagar aire acondicionado	El sistema permite apagar el aire acondicionado del hogar mediante la orden oral “apagar aire”.	A
RF19	Encender aspersor	El sistema permite encender el aspersor del jardín mediante la orden oral “encender aspersor”.	A
RF20	Apagar aspersor	El sistema permite apagar el aspersor del jardín mediante la orden oral “apagar aspersor”.	A

RF21	Abrir puerta	El sistema permite abrir la puerta del garaje mediante la orden oral “abrir puerta”.	A
RF22	Cerrar puerta	El sistema permite cerrar la puerta del garaje mediante la orden oral “cerrar puerta”.	A
RF23	Enviar mensaje	El sistema permite enviar un mensaje recordatorio a un panel electrónico mediante la orden oral “enviar mensaje”.	A
RF24	Apagar panel	El sistema permite apagar el panel electrónico mediante la orden oral “apagar panel”.	A
RF25	Activar alarma	El sistema permite activar la alarma del hogar mediante la orden oral “activar alarma”.	M
RF26	Desactivar alarma	El sistema permite desactivar la alarma del hogar mediante la orden oral “desactivar alarma”.	A
RF27	Apagar todo	El sistema permite apagar todos los dispositivos encendidos simultáneamente mediante una única orden oral “apagar todo”.	A
Acciones Futuras			
RF28	Definir fecha	El sistema permite establecer la fecha en la que se desea programar una acción futura en formato (dd de mm de aaaa, ejemplo: 16 de junio de 2022).	A
RF29	Definir hora	El sistema permite establecer el horario en el que se desea programar una acción futura en formato (hh y mm).	A
RF30	Programar encender luz	El sistema permite encender las luces del hogar en un instante de tiempo futuro.	A
RF31	Programar apagar luz	El sistema permite apagar las luces del hogar en un instante de tiempo futuro.	A
RF32	Programar encender luces de navidad	El sistema permite encender las luces de navidad del hogar en un instante de tiempo futuro.	B
RF33	Programar apagar luces de navidad	El sistema permite apagar las luces de navidad del hogar en un instante de tiempo futuro.	B
RF34	Programar encender calefactor temperatura baja	El sistema permite encender el calefactor del hogar en temperatura baja en un instante de tiempo futuro.	A
RF35	Programar encender calefactor temperatura media	El sistema permite encender el calefactor del hogar en temperatura media en un instante de tiempo futuro.	A
RF36	Programar encender calefactor temperatura alta	El sistema permite encender el calefactor del hogar en temperatura alta en un instante de tiempo futuro.	A

RF37	Programar apagar calefactor	El sistema permite apagar el calefactor del hogar en un instante de tiempo futuro.	A
RF38	Programar encender aire acondicionado intensidad baja	El sistema permite encender el aire acondicionado del hogar en intensidad baja en un instante de tiempo futuro.	A
RF39	Programar encender aire acondicionado intensidad media	El sistema permite encender el aire acondicionado del hogar en intensidad media en un instante de tiempo futuro.	A
RF40	Programar encender aire acondicionado intensidad alta	El sistema permite encender el aire acondicionado del hogar en intensidad alta en un instante de tiempo futuro.	A
RF41	Programar apagar aire acondicionado	El sistema permite apagar el aire acondicionado del hogar en un instante de tiempo futuro.	A
RF42	Programar encender aspersor	El sistema permite encender el aspersor del jardín en un instante de tiempo futuro.	A
RF43	Programar apagar aspersor	El sistema permite apagar el aspersor del jardín en un instante de tiempo futuro.	A
RF44	Programar abrir puerta	El sistema permite abrir la puerta del garaje en un instante de tiempo futuro.	A
RF45	Programar cerrar puerta	El sistema permite cerrar la puerta del garaje en un instante de tiempo futuro.	A
RF46	Programar enviar mensaje	El sistema permite enviar un mensaje recordatorio a un panel electrónico en un instante de tiempo futuro.	A
RF47	Programar apagar panel	El sistema permite apagar el panel electrónico en un instante de tiempo futuro.	A
RF48	Programar activar alarma	El sistema permite activar la alarma del hogar en un instante de tiempo futuro.	A
RF49	Programar desactivar alarma	El sistema permite desactivar la alarma del hogar en un instante de tiempo futuro.	A
RF50	Programar apagar todo	El sistema permite apagar todos los dispositivos encendidos simultáneamente en un instante de tiempo futuro.	B
Acciones Automáticas			
RF51	Modo automático calefactor	El sistema opera con el calefactor en modo automático siempre que el usuario no actúe manualmente sobre este dispositivo ordenando una de las siguientes acciones inmediatas (RF11-RF14) o futuras (RF34-RF37).	A

RF52	Encendido calefactor	El sistema enciende la calefacción del hogar, si la temperatura está por debajo de 10 ºC.	A
RF53	Apagado calefactor	El sistema apaga la calefacción si la temperatura está por encima de 14 ºC.	A
RF54	Modo automático aire acondicionado	El sistema opera con el aire acondicionado en modo automático siempre que el usuario no actúe manualmente sobre este dispositivo ordenando una de las siguientes acciones inmediatas (RF15-RF18) o futuras (RF35-RF38).	A
RF55	Encendido aire acondicionado	El sistema enciende el aire acondicionado del hogar, si la temperatura está por encima de 25 ºC.	A
RF56	Apagado aire acondicionado	El sistema apaga el aire acondicionado si la temperatura está por debajo de 21 ºC.	A
RF57	Activación alarma	El sistema activa la alarma del hogar si detecta movimientos en el entorno.	A
RF58	Notificar eventos	El sistema notifica al usuario sobre algunos eventos en el hogar.	A
RF59	Notificación de ejecución satisfactoria	El sistema notifica al usuario que la acción ordenada se ha ejecutado satisfactoriamente.	M

Tabla 4.1.- Requisitos funcionales del sistema.

4.2.- Requisitos no Funcionales

En la Tabla 4.2 se especifican los requisitos no funcionales, donde se acuerdan las prestaciones del sistema, así como sus limitaciones:

REQUISITOS NO FUNCIONALES			
ID	**Nombre**	**Descripción**	**Prioridad**
Requisitos de Usuario			
RNF01	Acceso a Internet	El usuario debe contar con una conexión a Internet para acceder a todas las funcionalidades del sistema.	A
RNF02	Correo electrónico	El usuario necesita una cuenta de correo electrónico para hacer uso de los servicios de Google *Cloud*, específicamente de la API STT.	A
RNF03	Cuenta de Telegram	El usuario debe tener una cuenta de Telegram para que el sistema le envíe las notificaciones.	A
RNF04	Idioma	El sistema opera en lenguaje español.	A

Requisitos Tecnológicos			
RNF05	Terminal VoIP	El usuario debe tener un terminal VoIP hardware o software.	A
RNF06	Centralita telefónica	Se requiere de un ordenador con funciones de centralita telefónica (Asterisk).	A
RNF07	Servidor Apache	Se requiere de un servidor Apache para actuar de intermediario en las comunicaciones entre el hogar y el usuario.	A
RNF08	API STT de Google *Cloud*	Se requiere que el usuario tenga contratado o se registre gratuitamente en el servicio STT de Google *Cloud* para las traducciones de voz a texto.	A
RNF09	Protocolo HTTP	Se requiere que las comunicaciones entre los diferentes bloques funcionales del sistema deben hacerse a través del protocolo HTTP para que sea accesible desde cualquier lugar.	A
RNF10	Servidor web	Se requiere de un servidor web para que procese las peticiones HTTP desde la centralita y actúe sobre los dispositivos del hogar.	A
RNF11	Lenguaje de programación	El sistema utiliza *scripts* en Bash, Perl, PHP y el propio lenguaje de Asterisk.	A
Prestaciones			
RNF12	Tiempo de respuesta	Los dispositivos deben responder a la interacción del usuario en un tiempo menor o igual a 3 segundos.	A
RNF13	Latencia	La latencia entre un evento ocurrido en el hogar y la correspondiente notificación al usuario debe ser menor o igual a tres segundos.	A
RNF14	Disponibilidad	El ordenador usado como centralita y con el servidor de Apache debe estar siempre activo, garantizando una disponibilidad mínima del 97%. El periodo de indisponibilidad corresponde con los mantenimientos del servicio.	A

Tabla 4.2.- Requisitos no funcionales del sistema.

5.- DISEÑO DEL SISTEMA

En este capítulo se detalla el diseño del sistema propuesto. Primeramente, se presenta el esquema general de la arquitectura y se hace una descripción de las funciones que realizan cada uno de los elementos que componen el sistema. Posteriormente, se describen los diferentes casos de uso y algunos diagramas de secuencias.

5.1.- Arquitectura

La arquitectura de este prototipo está compuesta por cuatro bloques: el usuario, un servidor de Asterisk (además de un servidor Apache que más adelante se explica su función específica), la API STT de Google *Cloud* y una plataforma IoT basada en Arduino que integra múltiples sensores y actuadores. En la Figura 5.1 se muestra la arquitectura general del sistema.

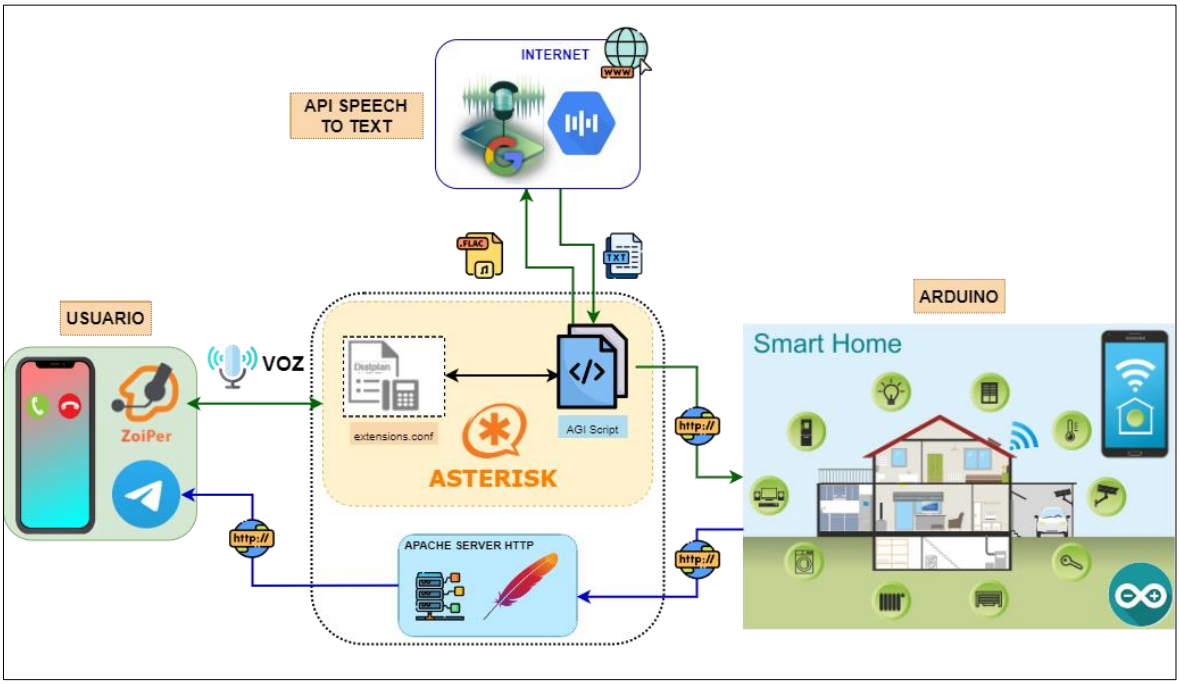


Figura 5.1.- Arquitectura general del sistema.

A continuación, una breve descripción de las funciones de cada uno de los bloques:

- **Usuario:** Es aquel que se beneficia de las diferentes funciones que ofrece el sistema. Para interactuar con el prototipo es necesario que el usuario disponga de un terminal de VoIP para que inicie la llamada a la centralita, en este caso concreto, se ha usado la aplicación Zoiper5. Además, debe tener una cuenta de Telegram para recibir las notificaciones generadas por el sistema.

- **Asterisk:** Es el elemento central de este proyecto ya que gestiona todo el flujo de las comunicaciones entre el resto de bloques del sistema. Tiene un IVR configurado que permite guiar al usuario durante la llamada, especialmente pensado para personas mayores, con alguna discapacidad o con nulos conocimientos informáticos. Como se ha mencionado, se implementa un servidor Apache para que actúe como intermediario de la comunicación cuando se desea enviar una notificación desde Arduino hacia el usuario.
- **API *Speech to Text*:** Este bloque es un elemento externo al sistema, se trata de una API de Google *Cloud* a la que se puede acceder a través de una conexión a Internet. Esta proporciona servicios de transcripción de voz a texto, lo cual puede ser usado para programar funciones en base al texto recibido.
- **Arduino:** Es el encargado de actuar directamente sobre los elementos que simulan electrodomésticos o funcionalidades de un hogar tradicional. También, recolecta información del entorno para su posterior notificación al usuario final.

5.1.1.- Descripción del Funcionamiento

Para una mejor comprensión del funcionamiento general del diseño mostrado en la Figura 5.1, se enumeran la lista secuencial de acciones que se producen.

1. El usuario, inicia la llamada a una extensión predefinida de la centralita de Asterisk (ext. 40), a través de un teléfono VoIP (hardware o software). Es imprescindible que dicho usuario esté previamente registrado en la centralita.
2. La llamada es atendida por el servidor Asterisk mediante un IVR que contiene los audios pregrabados con las indicaciones para el usuario. Luego se ejecuta la aplicación AGI dentro del *dialplan* que invoca a un *script* para efectuar el reconocimiento vocal del usuario. En el punto del IVR donde se le solicita al usuario la acción específica que desea ejecutar sobre el hogar domotizado, es preciso que este conozca los comandos de voz que están disponibles, de lo contrario el IVR no es capaz de reconocer la orden solicitada por el usuario.
3. Este *script* realiza la grabación de voz, la codifica en formato "*flac*" y la envía al servicio de reconocimiento de Google *Cloud* (API STT).
4. Seguidamente Google devuelve una posible transcripción de voz a formato de texto, así como la probabilidad de ser correcta dicha transcripción.
5. A partir del texto devuelto por la API, en el *dialplan* se implementa una lógica que, en dependencia de la elección del usuario durante la llamada lanza un *script* con una petición HTTP al servidor de Arduino con la opción de la orden especificada.

6. Arduino recibe esa petición, la procesa y envía señales de control a través de sus pines de salida a los dispositivos del hogar domotizado.
7. Envía mensaje de confirmación de acción realizada al chat de Telegram del usuario.

En la Figura 5.2 se muestra un esquema del IVR configurado en el *dialplan* de Asterisk para tener una visión global del ciclo que realiza una llamada a la centralita.

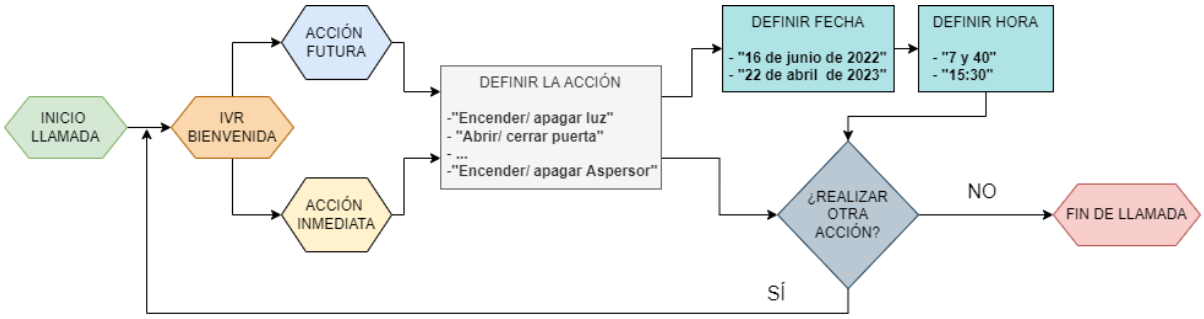


Figura 5.2.- Esquema general del IVR configurado.

Por otro lado, existe otro posible escenario donde no se ejecutan las mismas acciones anteriores, es decir, que no es el usuario quien inicia la comunicación sino algunos eventos ocurridos en el hogar. A continuación, se describe esta secuencia de pasos:

1. Se captura continuamente la lectura de dos sensores: el infrarrojo para detectar movimiento y el sensor DHT11 que mide la temperatura del entorno.
2. El primer sensor envía una señal de a los pines de entrada de Arduino y este activa la alarma. El segundo sensor envía lecturas analógicas del valor de la temperatura que Arduino utiliza para tomar decisiones automáticas sobre los dispositivos del hogar (el calefactor y el aire acondicionado) en función de unos umbrales establecidos.
3. Para enviar una notificación de lo ocurrido al usuario, Arduino envía una petición HTTP al servidor Apache. Esta petición activa un *script* escrito en PHP que invoca a otro *script* en Bash el cual envía finalmente un mensaje al chat de Telegram del usuario. En este sentido de la comunicación la API de Google queda totalmente excluida.

5.1.2.- Acciones configuradas en el Hogar

En este subapartado, se relacionan los sensores y actuadores seleccionados que pretenden simular una determinada funcionalidad dentro de un hogar convencional. En la Tabla 5.1 se aprecia dicha relación. Además, en la última columna se incluyen los comandos de voz permitidos, si no se encuentra coincidencia el sistema solicita que se repita el comando.

Nº	Dispositivos	Descripción de la funcionalidad	Comando de voz
1	Módulo LED luz blanca 1	Encendido/ apagado de las luces del hogar.	Encender luz Apagar luz
2	Módulo RGB	Encendido/ apagado de las luces de Navidad. Este módulo genera un patrón de colores aleatorios más allá de los 3 colores primarios.	Encender luces navideñas Apagar luces navideñas
3	Módulo *Fan* Motor Módulo LED luz roja	Encendido/ apagado del calefactor, permite especificar tres niveles de temperaturas simbolizados por tres velocidades diferentes del motor (baja, media, alta). Además, la luz roja se visualiza con tres intensidades en función de estos niveles. Puede ser activado manualmente o automático si la temperatura está por debajo de un umbral (10 ºC).	Encender calefactor en uno, dos o tres Apagar calefactor
4	Módulo *Fan* Motor Módulo LED luz blanca 2	Encendido/ apagado del aire acondicionado, permite especificar tres niveles de intensidad (baja, media, alta) emulado con tres velocidades del motor. Además, la luz blanca se visualiza con tres intensidades en función de la velocidad del motor. Puede ser activado manualmente o automático si la temperatura está por encima de un umbral (25 ºC).	Encender aire en uno, dos o tres Apagar aire
5	Micro Servomotor 1	Encendido/ apagado del aspersor, los movimientos giratorios en ambos sentidos del servomotor simulan el aspersor del jardín.	Encender aspersor Apagar aspersor
6	Micro Servomotor 2 Módulo LED luz verde Módulo LED luz amarilla 1 Módulo *Buzzer* Activo	Apertura/ cierre de la puerta del garaje emulado por el servomotor. El módulo *buzzer* y el LED amarillo simbolizan una señal sonora y luminosa respectivamente, que indica que la puerta está en un estado transitorio. Por el contrario el LED verde, indica que la puerta ha llegado a un estado final seguro, bien sea totalmente abierta o cerrada.	Abrir puerta Cerrar puerta
7	LCD 1602	Enviar mensaje/ apagado del *display*. Este dispositivo simula un panel electrónico que sirve para mostrar mensajes recordatorios.	Mensaje de longitud ≤ 32 caracteres Apagar panel
8	Módulo PIR Módulo *Buzzer* Pasivo Módulo LED luz amarilla 2	Encendido/ apagado de la alarma. El módulo PIR detecta movimientos, y se genera una señal sonora y luminosa que emulan una sirena. Este equipo puede ser activado manualmente por el usuario, o automáticamente según la activación del PIR.	Activar alarma Desactivar alarma
9	Módulo DTH11	Mide continuamente la temperatura del entorno. No permite realizar acciones manuales sobre él, sino que proporciona lecturas del entorno y en función de eso se programan acciones.	--

Tabla 5.1.- Relación de los dispositivos seleccionados y sus funciones.

Además, se ha programado una opción genérica de "Apagar todo" para una mayor comodidad del usuario, por si tuviera varias funcionalidades en ejecución no tenga que apagarlas una a una.

Igualmente, existe otra funcionalidad implementada que consiste en activar el modo automático en el hogar. Para ello se ha definido la extensión 41, a la cual el usuario debe llamar para activar este modo de operación. El modo automático activado implica Arduino toma decisiones sobre el hogar en función de los valores recogidos por los sensores. Sin embargo, si el usuario ordena una ejecución sobre un dispositivo que está actuando en modo automático esto provoca que se desactive dicho modo. Para habilitar este modo nuevamente es necesario que el usuario marque la extensión previamente dicha.

Esta funcionalidad tiene como objetivo que las órdenes del usuario tengan prioridad frente a las decisiones automáticas del hogar, de esta forma el usuario tiene un mayor control sobre sus dispositivos.

Del mismo modo, se ha habilitado la extensión 30 en la centralita para brindarle al usuario una herramienta que permita testear si el reconocimiento de su voz es correcto a través de la API STT. Cuando se marca esta extensión, se emite un tono después del cual se puede decir a viva voz cualquier frase. El sistema envía al chat de Telegram del usuario el resultado de la traducción de la API y la probabilidad de que la traducción sea correcta. Una probabilidad mayor o igual a 0.9 se considera confiable.

5.2.- Casos de Uso

En este apartado se detallan todos los posibles casos de uso que ofrece el sistema basado en los requisitos funcionales establecidos en el apartado 4.1. Estos casos de uso se han clasificado en cuatro subsistemas para lograr una mejor estructuración y distribución de las funcionalidades brindadas por el sistema:

- Subsistema de Acciones Inmediatas
- Subsistema de Acciones Futuras
- Subsistema de Acciones Automáticas
- Subsistema de Acciones Extras

Debido a la naturaleza similar de muchos de estos casos de uso, el lector puede obviar algunos de ellos.

5.2.1.- Subsistema de Acciones Inmediatas

A continuación, se muestran las tablas que describen cada uno de los casos de uso del Subsistema de Acciones Inmediatas que se presentan en el esquema de la Figura 5.3.

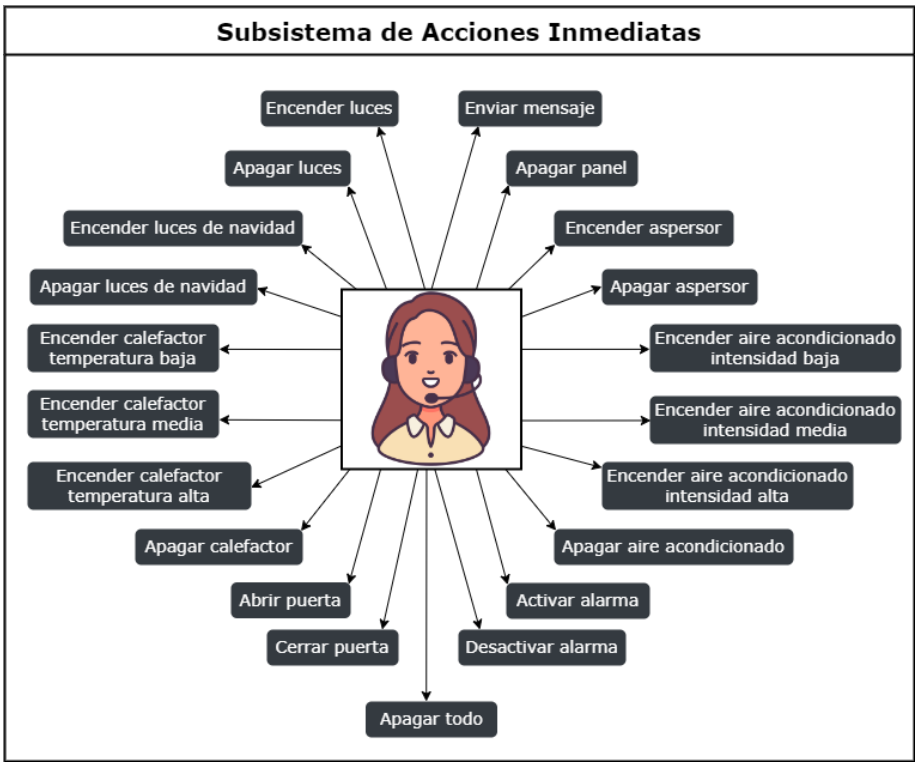


Figura 5.3.- Casos de uso del Subsistema de Acciones Inmediatas.

Nombre	**Encender luces**	**ID**	**CU-01**
Actores	Usuarios, API STT		
Objetivo	Encender las luces del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y luces apagadas		
Post-Condiciones	Las luces se encienden		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “ahora” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “encender luz” *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce		

	audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF07, RF59

Tabla 5.2.- CU-01: Encender luces.

Nombre	**Apagar luces**	**ID**	**CU-02**
Actores	Usuarios, API STT		
Objetivo	Apagar las luces del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y luces encendidas		
Post-Condiciones	Las luces se apagan		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar luz" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF08, RF59		

Tabla 5.3.- CU-02: Apagar luces.

Nombre	**Encender luces de Navidad**	**ID**	**CU-03**
Actores	Usuarios, API STT		
Objetivo	Encender las luces de Navidad del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y luces navideñas apagadas		
Post-Condiciones	Las luces navideñas se encienden		
Descripción	1. La centralita reproduce audio explicativo.		

	2. El usuario dice “ahora” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “encender luces navideñas” *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF09, RF59

Tabla 5.4.- CU-03: Encender las luces de Navidad.

Nombre	**Apagar luces de Navidad**	**ID**	**CU-04**
Actores	Usuarios, API STT		
Objetivo	Apagar las luces de Navidad del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y luces navideñas encendidas		
Post-Condiciones	Las luces navideñas se apagan		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “ahora” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “apagar luces navideñas” *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF10, RF59		

Tabla 5.5.- CU-04: Apagar las luces de Navidad.

Nombre	**Encender calefactor temperatura baja**	**ID**	**CU-05**
Actores	Usuarios, API STT		
Objetivo	Encender la calefacción del hogar a temperatura baja		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y calefactor apagado o encendido en temperatura media o alta		
Post-Condiciones	El calefactor se enciende en temperatura baja		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender calefactor en uno" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF11, RF59		

Tabla 5.6.- CU-05: Encender el calefactor (temperatura baja).

Nombre	**Encender calefactor temperatura media**	**ID**	**CU-06**
Actores	Usuarios, API STT		
Objetivo	Encender la calefacción del hogar a temperatura media		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y calefactor apagado o encendido en temperatura baja o alta		
Post-Condiciones	El calefactor se enciende en temperatura media		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender calefactor en dos" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un		

	mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige hacia el usuario al menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF12, RF59

Tabla 5.7.- CU-06: Encender el calefactor (temperatura media).

Nombre	**Encender calefactor temperatura alta**	**ID**	**CU-07**
Actores	Usuarios, API STT		
Objetivo	Encender la calefacción del hogar a temperatura alta		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y calefactor apagado o encendido en temperatura baja o media		
Post-Condiciones	El calefactor se enciende en temperatura alta		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender calefactor en tres" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige hacia el usuario al menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF13, RF59		

Tabla 5.8.- CU-07: Encender el calefactor (temperatura alta).

Nombre	**Apagar calefactor**	**ID**	**CU-08**
Actores	Usuarios, API STT		
Objetivo	Apagar la calefacción del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y calefactor encendido en temperatura baja, media o alta		
Post-Condiciones	El calefactor se apaga		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar calefactor" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF14, RF59		

Tabla 5.9.- CU-08: Apagar el calefactor.

Nombre	**Encender aire acondicionado intensidad baja**	**ID**	**CU-09**
Actores	Usuarios, API STT		
Objetivo	Encender la climatización del hogar con intensidad baja		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado apagado o encendido con intensidad media o alta		
Post-Condiciones	El aire acondicionado se enciende con intensidad baja		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aire en uno" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un		

	mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF15, RF59

Tabla 5.10.- CU-09: Encender el aire acondicionado (intensidad baja).

Nombre	**Encender aire acondicionado intensidad media**	**ID**	**CU-10**
Actores	Usuarios, API STT		
Objetivo	Encender la climatización del hogar con intensidad media		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y el aire acondicionado apagado o encendido con intensidad baja o alta		
Post-Condiciones	El aire acondicionado se enciende con intensidad media		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aire en dos" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF16, RF59		

Tabla 5.11.- CU-10: Encender el aire acondicionado (intensidad media).

Nombre	Encender aire acondicionado intensidad alta	ID	CU-11
Actores	Usuarios, API STT		
Objetivo	Encender la climatización del hogar con intensidad alta		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y el aire acondicionado apagado o encendido con intensidad baja o media		
Post-Condiciones	El aire acondicionado se enciende con intensidad alta		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aire en tres" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF17, RF59		

Tabla 5.12.- CU-11: Encender el aire acondicionado (intensidad alta).

Nombre	Apagar aire acondicionado	ID	CU-12
Actores	Usuarios, API STT		
Objetivo	Apagar la climatización del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado encendido con intensidad baja, media o alta		
Post-Condiciones	El aire acondicionado se apaga		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar aire" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía un		

	mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF18, RF59

Tabla 5.13.- CU-12: Apagar el aire acondicionado.

Nombre	**Encender aspersor**	**ID**	**CU-13**
Actores	Usuarios, API STT		
Objetivo	Encender el aspersor del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y aspersor apagado		
Post-Condiciones	El aspersor se enciende		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aspersor" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF19, RF59		

Tabla 5.14.- CU-13: Encender el aspersor.

Nombre	**Apagar aspersor**	**ID**	**CU-14**
Actores	Usuarios, API STT		
Objetivo	Apagar el aspersor del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y aspersor encendido		
Post-Condiciones	El aspersor se apaga		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar aspersor" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF20, RF59		

Tabla 5.15.- CU-14: Apagar el aspersor.

Nombre	**Abrir puerta**	**ID**	**CU-15**
Actores	Usuarios, API STT		
Objetivo	Abrir la puerta del garaje del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y puerta cerrada		
Post-Condiciones	La puerta se abre		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "abrir puerta" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción y procesa la apertura, además se procesa una señal sonora y luminosa para alertar que la puerta está en estado		

	transitorio. Cuando se abre completamente se observa una señal luminosa verde. Además, envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige hacia el usuario al menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF21, RF59

Tabla 5.16.- CU-15: Abrir la puerta del garaje.

Nombre	**Cerrar puerta**	**ID**	**CU-16**
Actores	Usuarios, API STT		
Objetivo	Cerrar la puerta del garaje del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y puerta abierta		
Post-Condiciones	La puerta se cierra		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "cerrar puerta" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción y procesa el cierre, además se procesa una señal sonora y luminosa para alertar que la puerta está en estado transitorio. Cuando se cierra completamente se observa una señal luminosa verde durante tres segundos. Además, envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF22, RF59		

Tabla 5.17.- CU-16: Cerrar la puerta del garaje.

Nombre	**Enviar mensaje**	**ID**	**CU-17**
Actores	Usuarios, API STT		
Objetivo	Enviar un mensaje de no más de 32 caracteres al panel electrónico del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y panel electrónico activo		
Post-Condiciones	Se muestra el mensaje por la pantalla del panel electrónico		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "enviar mensaje" *. 5. La centralita reproduce audio explicativo del mensaje. 6. La centralita graba el mensaje dicho por el usuario. 7. La centralita ejecuta el *script* con la opción correspondiente. 8. El servidor Arduino recibe el mensaje, lo procesa y lo muestra en el panel electrónico. Además, envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 9. La centralita reproduce audio explicativo para continuar o finalizar. 10. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 11. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF23, RF59		

Tabla 5.18.- CU-17: Enviar un mensaje al panel electrónico.

Nombre	**Apagar panel electrónico**	**ID**	**CU-18**
Actores	Usuarios, API STT		
Objetivo	Apagar el panel electrónico del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y el panel electrónico encendido		
Post-Condiciones	El panel electrónico se apaga		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones.		

	4. El usuario responde “apagar panel” *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa el apagado y envía mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF24, RF59

Tabla 5.19.- CU-18: Apagar el panel electrónico.

Nombre	**Activar alarma**	**ID**	**CU-19**
Actores	Usuarios, API STT		
Objetivo	Activar la alarma del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y alarma desactivada		
Post-Condiciones	Se activa la alarma		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “ahora” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “activar alarma” *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa la activación. y envía mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige hacia el usuario al menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF25, RF59		

Tabla 5.20.- CU-19: Activar la alarma.

Nombre	Desactivar alarma	ID	CU-20
Actores	Usuarios, API STT		
Objetivo	Desactivar la alarma del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y alarma activada		
Post-Condiciones	Se desactiva la alarma		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "desactivar alarma" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción, procesa la desactivación de la alarma y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF26, RF59		

Tabla 5.21.- CU-20: Desactivar la alarma.

Nombre	Apagar todo	ID	CU-21
Actores	Usuarios, API STT		
Objetivo	Apagar todos los dispositivos encendidos del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y cualquier dispositivo encendido		
Post-Condiciones	Se apagan todos los dispositivos		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "ahora" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar todo" *. 5. La centralita ejecuta el *script* con la opción correspondiente. 6. El servidor Arduino recibe la opción y procesa el apagado de todos los		

	dispositivos. 7. La centralita reproduce audio explicativo para continuar o finalizar. 8. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 9. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF27, RF59

Tabla 5.22.- CU-21: Apagar todos los dispositivos.

5.2.2.- Subsistema de Acciones Futuras

A continuación, se muestran las tablas que describen cada uno de los casos de uso del Subsistema de Acciones Futuras que se presentan en el esquema de la Figura 5.4.

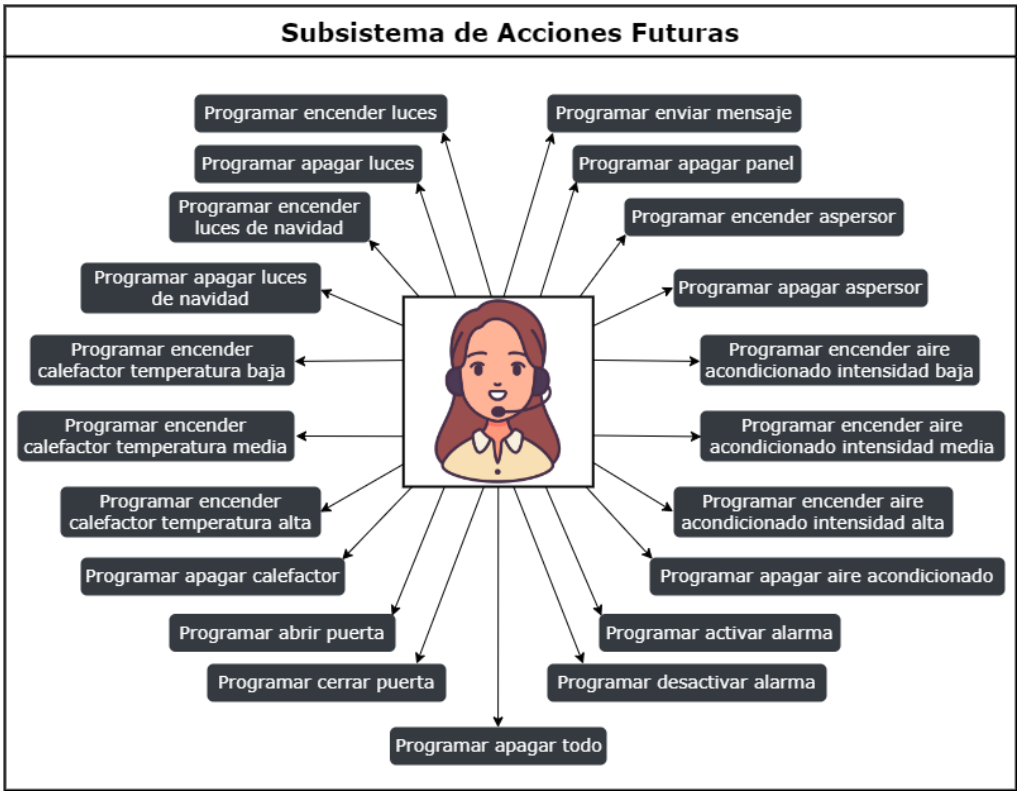


Figura 5.4.- Casos de uso del Subsistema de Acciones Futuras.

Nombre	**Programar encender luces**	**ID**	**CU-22**
Actores	Usuarios, API STT		
Objetivo	Programar encender las luces del hogar en un instante futuro		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita , luces apagadas		
Post-Condiciones	Las luces se encienden en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender luz" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. .* Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF30, RF59		

Tabla 5.23.- CU-22: Programar encender luces.

Nombre	**Programar apagar luces**	**ID**	**CU-23**
Actores	Usuarios, API STT		
Objetivo	Programar apagar las luces del hogar en un instante futuro		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita , luces encendidas		
Post-Condiciones	Las luces se encienden en el instante futuro especificado		

Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar luz" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF31, RF59

Tabla 5.24.- CU-23: Programar apagar luces.

Nombre	**Programar encender luces de Navidad**	**ID**	**CU-24**
Actores	Usuarios, API STT		
Objetivo	Programar encender las luces de Navidad del hogar en un instante futuro		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita , luces navideñas apagadas		
Post-Condiciones	Las luces navideñas se encienden en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender luces navideñas" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora.		

	8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF32, RF59

Tabla 5.25.- CU-24: Programar encender luces de Navidad.

Nombre	**Programar apagar luces de Navidad**	**ID**	**CU-25**
Actores	Usuarios, API STT		
Objetivo	Programar apagar las luces de Navidad del hogar en un instante futuro		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita , luces navideñas encendidas		
Post-Condiciones	Las luces navideñas se apagan en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar luces navideñas" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la		

	opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF33, RF59

Tabla 5.26.- CU-25: Programar apagar las luces de Navidad.

Nombre	**Programar encender calefactor temperatura baja**	**ID**	**CU-26**
Actores	Usuarios, API STT		
Objetivo	Programar encender la calefacción del hogar a temperatura baja		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, calefactor apagado o encendido en temperatura media o alta		
Post-Condiciones	El calefactor se enciende en temperatura baja en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “después” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “encender calefactor en uno” *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF34, RF59		

Tabla 5.27.- CU-26: Programar encender el calefactor (temperatura baja).

Nombre	**Programar encender calefactor temperatura media**	**ID**	**CU-27**
Actores	Usuarios, API STT		
Objetivo	Programar encender la calefacción del hogar a temperatura media		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, calefactor apagado o encendido en temperatura baja o alta		
Post-Condiciones	El calefactor se enciende en temperatura media en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender calefactor en dos" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF35, RF59		

Tabla 5.28.- CU-27: Programar encender el calefactor (temperatura media).

Nombre	**Programar encender calefactor temperatura alta**	**ID**	**CU-28**
Actores	Usuarios, API STT		
Objetivo	Programar encender la calefacción del hogar a temperatura alta		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, calefactor apagado o encendido en temperatura baja o media		

Post-Condiciones	El calefactor se enciende en temperatura alta en el instante futuro especificado
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender calefactor en tres" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF36, RF59

Tabla 5.29.- CU-28: Programar encender el calefactor (temperatura alta).

Nombre	**Programar apagar calefactor**	**ID**	**CU-29**
Actores	Usuarios, API STT		
Objetivo	Programar apagar la calefacción del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, calefactor encendido en temperatura baja, media o alta		
Post-Condiciones	El calefactor se apaga en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar calefactor" *. 5. La centralita reproduce audio de solicitud de fecha.		

	6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF37, RF59

Tabla 5.30.- CU-29: Programar apagar el calefactor.

Nombre	**Programar encender aire acondicionado intensidad baja**	**ID**	**CU-30**
Actores	Usuarios, API STT		
Objetivo	Programar encender la climatización del hogar en intensidad baja		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado apagado o encendido en intensidad media o alta		
Post-Condiciones	El aire acondicionado se enciende en intensidad baja en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aire en uno" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar.		

	11. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF38, RF59

Tabla 5.31.- CU-30: Programar encender el aire acondicionado (intensidad baja).

Nombre	**Programar encender aire acondicionado intensidad media**	**ID**	**CU-31**
Actores	Usuarios, API STT		
Objetivo	Programar encender la climatización del hogar en intensidad media		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado apagado o encendido en intensidad baja o alta		
Post-Condiciones	El aire acondicionado se enciende en intensidad media en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “después” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “encender aire en dos” *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada.		

	* Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF39, RF59

Tabla 5.32.- CU-31: Programar encender el aire acondicionado (intensidad media).

Nombre	**Programar encender aire acondicionado intensidad alta**	**ID**	**CU-32**
Actores	Usuarios, API STT		
Objetivo	Programar encender la climatización del hogar en intensidad alta		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado apagado o encendido en intensidad baja o media		
Post-Condiciones	El aire acondicionado se enciende en intensidad alta en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “después” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “encender aire en tres” *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF40, RF59		

Tabla 5.33.- CU-32: Programar encender el aire acondicionado (intensidad alta).

Nombre	**Programar apagar aire acondicionado**	**ID**	**CU-33**
Actores	Usuarios, API STT		
Objetivo	Programar apagar la climatización del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aire acondicionado encendido en intensidad baja, media o alta		
Post-Condiciones	El aire acondicionado se apaga en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar aire" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF41, RF59		

Tabla 5.34.- CU-33: Programar apagar el aire acondicionado.

Nombre	**Programar encender aspersor**	**ID**	**CU-34**
Actores	Usuarios, API STT		
Objetivo	Programar encender el aspersor del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aspersor encendido		
Post-Condiciones	El aspersor se enciende en el instante futuro especificado		

Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "encender aspersor" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el encendido y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF42, RF59

Tabla 5.35.- CU-34: Programar encender el aspersor.

Nombre	**Programar apagar aspersor**	**ID**	**CU-35**
Actores	Usuarios, API STT		
Objetivo	Programar apagar el aspersor del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, aspersor encendido		
Post-Condiciones	El aspersor se apaga en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar aspersor" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora.		

	8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF43, RF59

Tabla 5.36.- CU-35: Programar apagar el aspersor.

Nombre	**Programar abrir puerta**	**ID**	**CU-36**
Actores	Usuarios, API STT		
Objetivo	Programar abrir la puerta del garaje del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, puerta cerrada		
Post-Condiciones	La puerta se abre en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "abrir puerta" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la		

	opción correspondiente. 14. El servidor Arduino recibe la opción y procesa la apertura. Además, procesa una señal sonora y luminosa para alertar que la puerta está en estado transitorio. Cuando se abre completamente se observa una señal luminosa verde. 14. También envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF44, RF59

Tabla 5.37.- CU-36: Programar abrir la puerta del garaje.

Nombre	**Programar cerrar puerta**	**ID**	**CU-37**
Actores	Usuarios, API STT		
Objetivo	Programar cerrar la puerta del garaje del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, puerta abierta		
Post-Condiciones	La puerta se cierra en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “después” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “cerrar puerta” *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción y procesa el cierre. Además, procesa una señal sonora y luminosa para alertar que la puerta está en estado transitorio. Cuando se cierra completamente se observa una señal luminosa verde durante tres segundos. También envía un mensaje al chat de Telegram del usuario que confirma la acción realizada.		

	* Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF45, RF59

Tabla 5.38.- CU-37: Programar cerrar la puerta del garaje.

Nombre	**Programar enviar mensaje**	**ID**	**CU-38**
Actores	Usuarios, API STT		
Objetivo	Programar enviar un mensaje de no más de 32 caracteres al panel electrónico del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, panel electrónico activo		
Post-Condiciones	Se muestra el mensaje por la pantalla del panel electrónico en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice “después” *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde “enviar mensaje” *. 5. La centralita reproduce audio de solicitud del mensaje. 6. La centralita graba el mensaje dicho por el usuario. 7. La centralita reproduce audio de solicitud de fecha. 8. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 9. La centralita reproduce audio de solicitud de hora. 10. El usuario define la hora verbalmente en formato (hh y mm) *. 11. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 12. La centralita reproduce audio explicativo para continuar o finalizar. 13. El usuario responde “uno” * o “dos” * para continuar o finalizar la llamada respectivamente. 14. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 15. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 16. El servidor Arduino recibe el mensaje, lo procesa y lo muestra en el panel electrónico. Además, envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF46, RF59		

Tabla 5.39.- CU-38: Programar enviar mensaje al panel electrónico.

Nombre	**Programar apagar panel electrónico**	**ID**	**CU-39**
Actores	Usuarios, API STT		
Objetivo	Programar apagar panel electrónico del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, panel electrónico encendido		
Post-Condiciones	Se apaga el panel electrónico en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio de solicitud de acciones. 4. El usuario responde "apagar panel" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa el apagado y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.		
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF47, RF59		

Tabla 5.40.- CU-39: Programar apagar el panel electrónico.

Nombre	**Programar activar alarma**	**ID**	**CU-40**
Actores	Usuarios, API STT		
Objetivo	Programar activar la alarma del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, alarma desactivada		
Post-Condiciones	Se activa la alarma en el instante futuro especificado		

Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio explicativo de acciones. 4. El usuario responde "activar alarma" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa la activación de la alarma y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF48, RF59

Tabla 5.41.- CU-40: Programar activar la alarma.

Nombre	**Programar desactivar alarma**	**ID**	**CU-41**
Actores	Usuarios, API STT		
Objetivo	Programar desactivar la alarma del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, alarma activada		
Post-Condiciones	Se desactiva la alarma en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio explicativo de acciones. 4. El usuario responde "desactivar alarma" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora.		

	8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada. 13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción, procesa la desactivación de la alarma y envía un mensaje al chat de Telegram del usuario que confirma la acción realizada. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF49, RF59

Tabla 5.42.- CU-41: Programar desactivar la alarma.

Nombre	**Programar apagar todo**	**ID**	**CU-42**
Actores	Usuarios, API STT		
Objetivo	Programar apagar todos los dispositivos encendidos del hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, cualquier dispositivo encendido		
Post-Condiciones	Se apagan todos los dispositivos en el instante futuro especificado		
Descripción	1. La centralita reproduce audio explicativo. 2. El usuario dice "después" *. 3. La centralita reproduce audio explicativo de acciones. 4. El usuario responde "apagar todo" *. 5. La centralita reproduce audio de solicitud de fecha. 6. El usuario define la fecha verbalmente en formato (dd de mm de aaaa) *. 7. La centralita reproduce audio de solicitud de hora. 8. El usuario define la hora verbalmente en formato (hh y mm) *. 9. La centralita ejecuta el *script* que crea un *call file* con la información proporcionada por el usuario. 10. La centralita reproduce audio explicativo para continuar o finalizar. 11. El usuario responde "uno" * o "dos" * para continuar o finalizar la llamada respectivamente. 12. Si desea continuar se redirige al usuario hacia el menú de inicio, si desea finalizar se cuelga la llamada.		

	13. Llegado el instante futuro definido, la centralita ejecuta el *script* con la opción correspondiente. 14. El servidor Arduino recibe la opción y procesa el apagado de todos los dispositivos. * Si el usuario dice el comando incorrecto o no dice nada, se reproduce audio de error y se repite el audio explicativo correspondiente.
Correspondencia	RF01, RF02, RF03, RF04, RF28, RF29, RF50, RF59

Tabla 5.43.- CU-42: Programar apagar todos los dispositivos.

5.2.3.- Subsistema de Acciones Automáticas

A continuación, se muestran las tablas que describen cada uno de los casos de uso del Subsistema de Acciones Automáticas que se presentan en el esquema de la Figura 5.5.

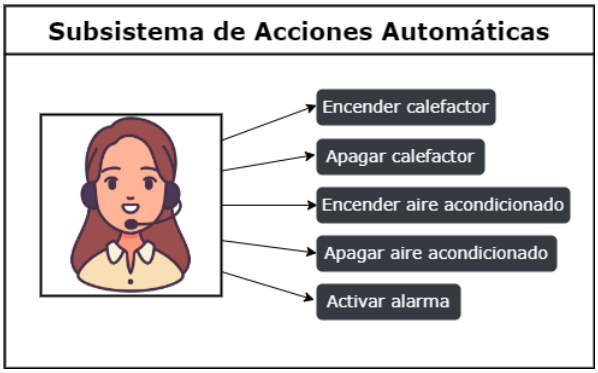


Figura 5.5.- Casos de uso del Subsistema de Acciones Automáticas.

Nombre	**Encender calefactor**	**ID**	**CU-43**
Actores	Usuarios, API STT		
Objetivo	Encender la calefacción del hogar si la temperatura está por debajo de los 10 ºC		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, medición del sensor de temperatura por debajo de 10 ºC, calefactor apagado		
Post-Condiciones	El calefactor se enciende en temperatura media		
Descripción	1. Medición del sensor de temperatura por debajo de 10 ºC. 2. El servidor Arduino procesa el encendido del calefactor en temperatura media. 3. El servidor Arduino (ahora actuando de cliente) hace una petición HTTP al servidor Apache donde se aloja un *script* PHP que invoca a otro *script* en Bash que envía el mensaje al usuario. 4. El usuario es notificado de la acción a través de su chat de Telegram.		
Correspondencia	RF51, RF52, RF58		

Tabla 5.44.- CU-43: Encendido automático del calefactor.

Nombre	Apagar calefactor	ID	CU-44
Actores	Usuarios, API STT		
Objetivo	Apagar la calefacción del hogar si la temperatura está por encima de 14 ºC		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, medición del sensor de temperatura por encima de 14 ºC, calefactor encendido		
Post-Condiciones	El calefactor se apaga		
Descripción	1. Medición del sensor de temperatura por encima de 14 ºC. 2. El servidor Arduino procesa el apagado del calefactor. 3. El servidor Arduino (ahora actuando de cliente) hace una petición HTTP al servidor Apache donde se aloja un *script* PHP que invoca a otro *script* en Bash que envía el mensaje al usuario. 4. El usuario es notificado de la acción a través de su chat de Telegram.		
Correspondencia	RF51, RF53, RF58		

Tabla 5.45.- CU-44: Apagado automático del calefactor.

Nombre	Encender aire acondicionado	ID	CU-45
Actores	Usuarios, API STT		
Objetivo	Encender la climatización del hogar si la temperatura está por encima de 25 ºC		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, medición del sensor de temperatura por encima de 25 ºC, aire acondicionado apagado		
Post-Condiciones	El aire acondicionado se enciende en intensidad media		
Descripción	1. Medición del sensor de temperatura por encima de 25 ºC. 2. El servidor Arduino procesa el encendido del aire acondicionado en temperatura media. 3. El servidor Arduino (ahora actuando de cliente) hace una petición HTTP al servidor Apache donde se aloja un *script* PHP que invoca a otro *script* en Bash que envía el mensaje al usuario. 4. El usuario es notificado de la acción a través de su chat de Telegram.		
Correspondencia	RF54, RF55, RF58		

Tabla 5.46.- CU-45: Encendido automático del aire acondicionado.

Nombre	Apagar aire acondicionado	ID	CU-46
Actores	Usuarios, API STT		
Objetivo	Apagar la climatización del hogar si la temperatura está por debajo de 21 ºC		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, medición del sensor de temperatura por debajo de 21 ºC, aire acondicionado encendido		
Post-Condiciones	El aire acondicionado se apaga		
Descripción	1. Medición del sensor de temperatura por debajo de 21 ºC. 2. El servidor Arduino procesa el apagado del aire acondicionado. 3. El servidor Arduino (ahora actuando de cliente) hace una petición HTTP al servidor Apache donde se aloja un *script* PHP que invoca a otro *script* en Bash que envía el mensaje al usuario. 4. El usuario es notificado de la acción a través de su chat de Telegram.		
Correspondencia	RF54, RF56, RF58		

Tabla 5.47.- CU-46: Apagado automático del aire acondicionado.

Nombre	Activación alarma	ID	CU-47
Actores	Usuarios, API STT		
Objetivo	Activar la alarma si se detectan movimientos en el hogar		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita, alarma desactivada y sensor infrarrojo operativo		
Post-Condiciones	Se activa la alarma		
Descripción	1. Detección de movimientos a través del sensor infrarrojo. 2. El servidor Arduino procesa la activación de la alarma. 3. El servidor Arduino (ahora actuando de cliente) hace una petición HTTP al servidor Apache donde se aloja un *script* PHP que invoca a otro *script* en Bash que envía el mensaje al usuario. 4. El usuario es notificado de la activación de la alarma en el hogar.		
Correspondencia	RF57, RF58		

Tabla 5.48.- CU-47: Activación automática de la alarma.

5.2.4.- Subsistema de Acciones Extras

A continuación, se muestra la tabla que describe el caso de uso del Subsistema de Acciones Extras que se presenta en el esquema de la Figura 5.6.

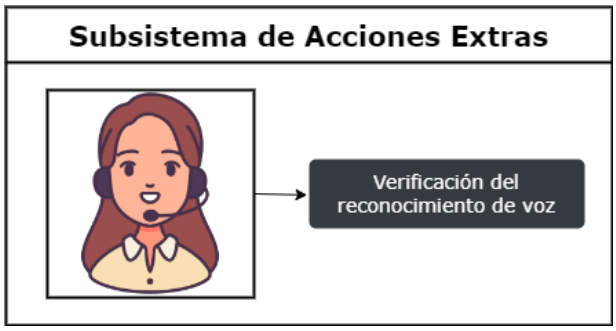


Figura 5.6.- Caso de uso del Subsistema de Acciones Extras.

Nombre	**Verificación del reconocimiento de voz**	**ID**	**CU-48**
Actores	Usuarios, API STT		
Objetivo	Realizar el reconocimiento de la voz del usuario a través de la API STT para verificar el correcto funcionamiento de la transcripción		
Pre-Condiciones	Tener una conexión a Internet, terminal del usuario registrado en la centralita y un mensaje de voz del usuario		
Post-Condiciones	La API devuelve la transcripción del audio a texto y la probabilidad de ser correcta la transcripción		
Descripción	1. El usuario marca la extensión 30 de la centralita. 2. Se emite un tono después del cual el usuario dice verbalmente una frase que desee transcribir. 3. El sistema envía el resultado de la transcripción y la probabilidad de ser correcto al chat de Telegram del usuario. 4. El usuario comprueba si el reconocimiento fue correcto o no.		
Correspondencia	RF02, RF05, RF06		

Tabla 5.49.- CU-48: Verificación del reconocimiento de voz.

5.3.- Diagramas de Secuencias

En este apartado se muestran los diagramas de secuencia correspondientes a algunos de los ejemplos de los casos de uso del sistema. Se propone solo una muestra de ellos para no saturar este documento con información que en muchos casos resulta redundante. El objetivo es mostrar un diagrama que represente a todos los casos de uso de funcionamiento similar.

5.3.1.- Acciones Inmediatas

En la Figura 5.7 se aprecia el diagrama de secuencia que sirve para ejemplificar una acción inmediata. Se puede observar que el elemento central de la comunicación es Asterisk, el cual actúa de intermediario entre el usuario y la API de Google. Este escenario es muy sencillo, una vez que el usuario indica que desea realizar una acción inmediata, se le solicita la acción y la centralita procesa la petición. Por último, se inicia un diálogo para continuar o finalizar la llamada, siendo la segunda opción la que se representa en la figura. La parte relativa a la confirmación de la acción realizada se ha obviado para una mayor claridad en la representación del diagrama de secuencia.

Además, se ha sombreado en color naranja los diferentes IVR involucrados. Este sombreado representa el hecho de que el sistema permanece en bucle en esa fase hasta que el usuario introduzca correctamente el respectivo comando de voz. Mientras sea incorrecto el sistema no continúa a la ejecución de la siguiente tarea.

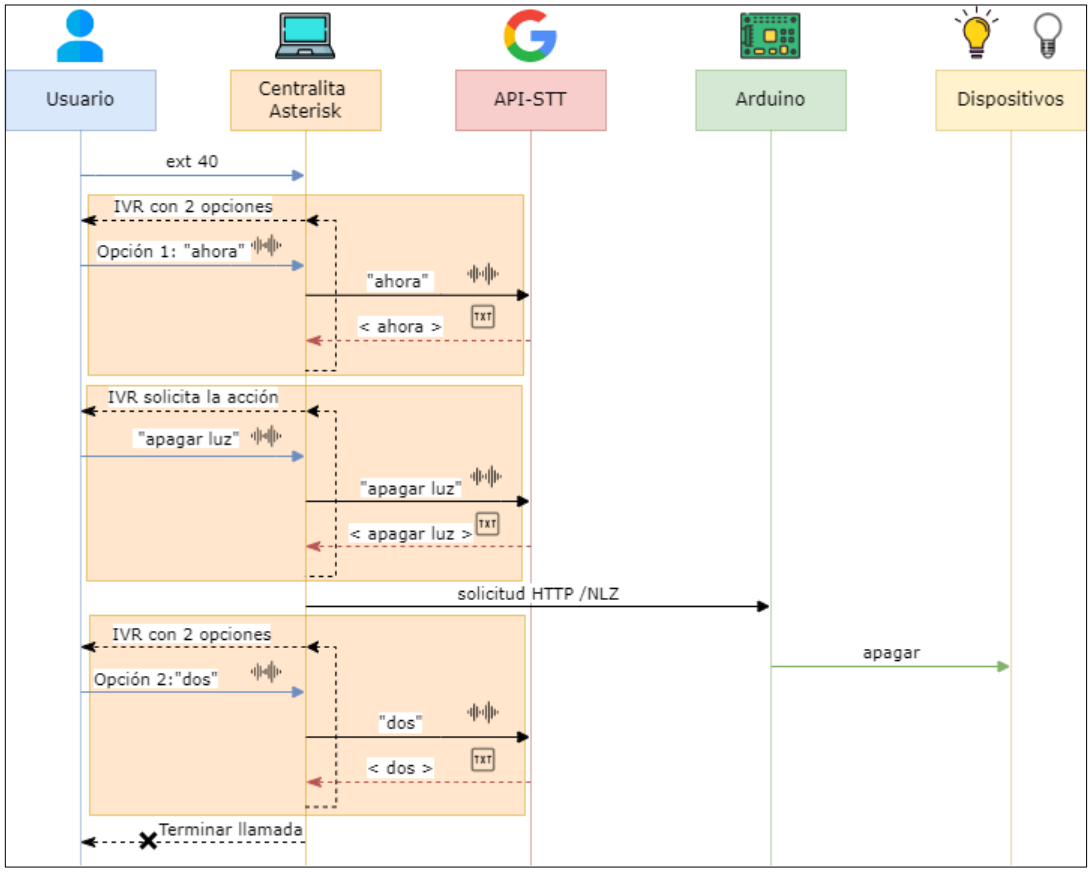


Figura 5.7.- Diagrama de secuencia para un ejemplo de una acción inmediata.

5.3.2.- Acciones Futuras

En la Figura 5.8 se aprecia el diagrama de secuencia que sirve para ejemplificar una acción futura. De igual manera que en el caso anterior se puede observar que el elemento central de la comunicación sigue siendo Asterisk. Este escenario es ligeramente más

complejo que el anterior ya que una vez que el usuario indica la acción futura que desea planificar, se le solicita información correspondiente a la fecha y hora de la ejecución de la acción. Durante el proceso la centralita realiza tareas para comprobar que los formatos de fecha y hora aportados son correctos. En ese caso crea un *call file* para su posterior ejecución. Por último, se inicia un diálogo para continuar o finalizar la llamada. Transcurrido el tiempo señalado por el usuario la centralita ejecuta el *call file* y se procesa la petición al servidor de Arduino para que ejecute la apertura de la puerta.

De la misma forma que el caso anterior el sombreado naranja representa las fases donde el sistema permanece en bucle hasta que no se introduzcan los comandos de voz correctamente para avanzar a la siguiente tarea.

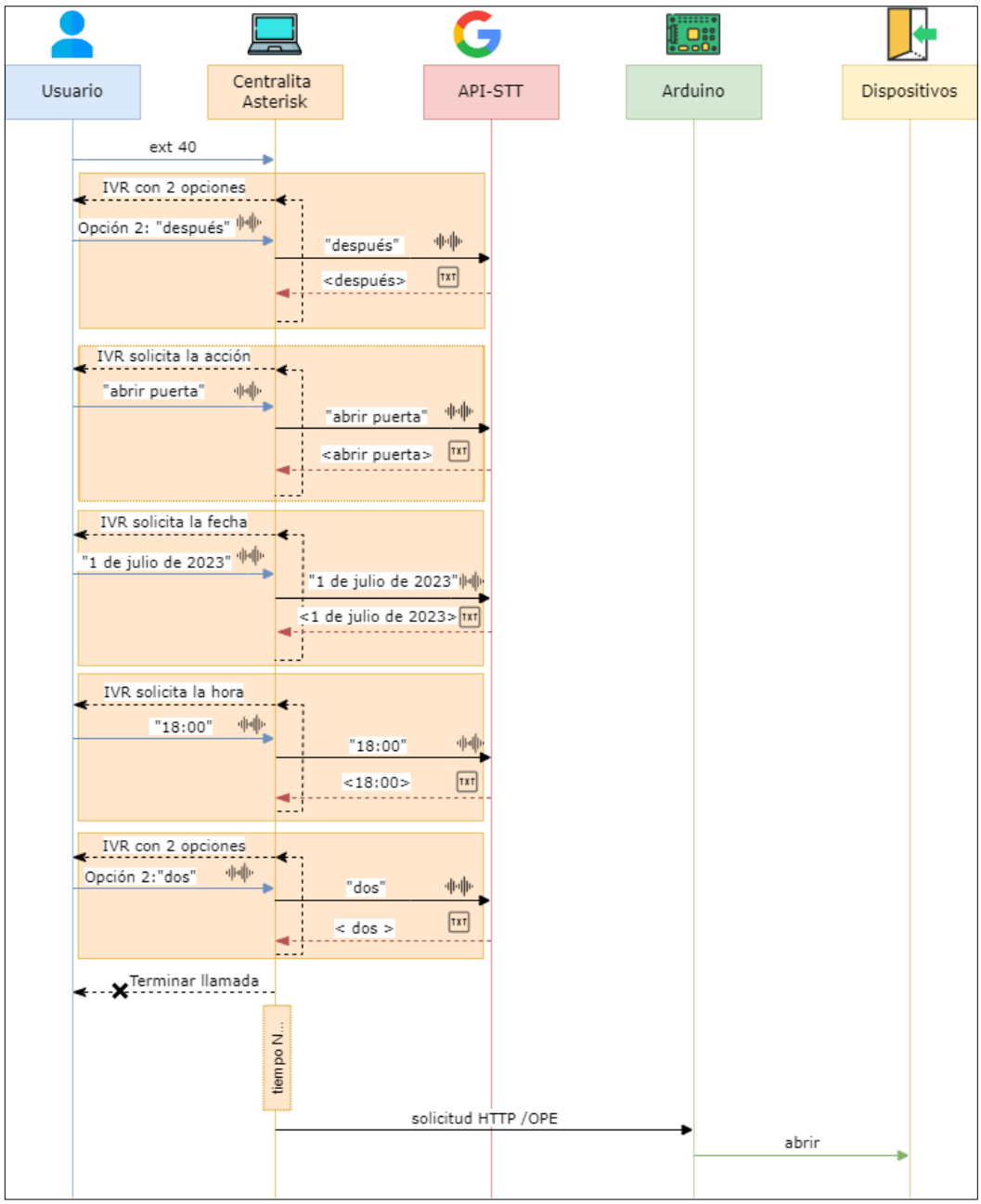


Figura 5.8.- Diagrama de secuencia para un ejemplo de una acción futura.

5.3.3.- Acciones Automáticas

En la Figura 5.9 se aprecia el diagrama de secuencia correspondiente a la activación de la alarma de forma automática. Este caso difiere en gran medida de los vistos anteriormente debido a que no es el usuario quien inicia la comunicación. Como se puede observar, ahora es el servidor de Arduino el que inicia la secuencia ya que recibe una señal por sus pines de entrada que provocan la activación de la alarma. Tras esto, envía una petición HTTP al servidor Apache que activa el *script* "notif.php" para invocar el *script* "telegram.sh" que finalmente notifica al usuario del evento ocurrido.

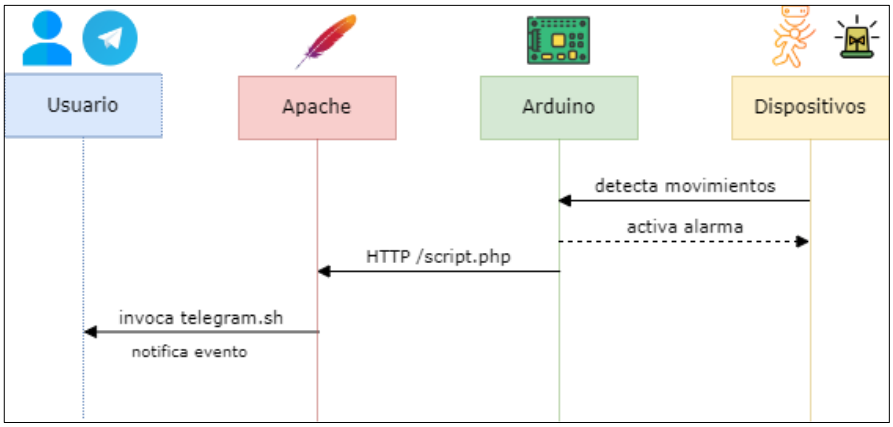


Figura 5.9.- Diagrama de secuencia para la activación de la alarma.

En la Figura 5.10 se aprecia el diagrama de secuencia para el encendido automático del aire acondicionado. Arduino recibe por sus pines de entrada las lecturas del sensor de temperatura y cuando se supera un umbral enciende automáticamente la climatización en el hogar. Posteriormente envía una petición HTTP al servidor Apache, el cual ejecuta directamente el *script* que envía la notificación al usuario.

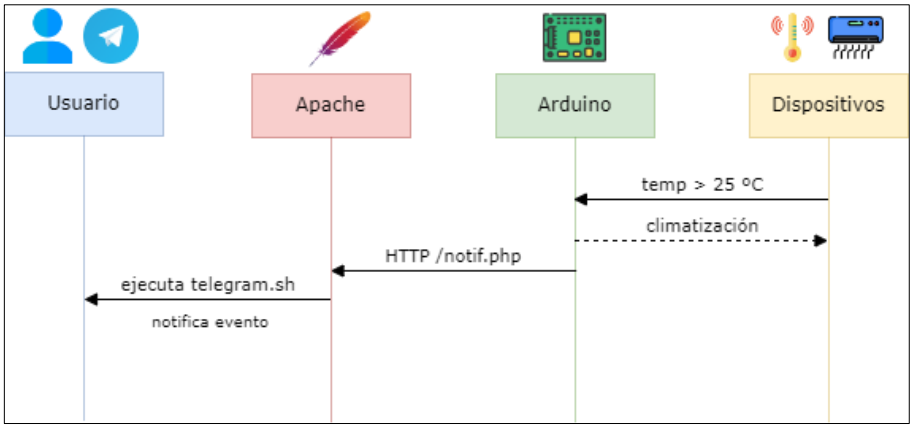


Figura 5.10.- Diagrama de secuencia para el encendido del aire acondicionado.

6.- DESCRIPCIÓN TÉCNICA DEL SISTEMA

En este capítulo se describen los detalles de la implementación sobre la centralita como la configuración de los ficheros principales de Asterisk y el conjunto de *scripts* programados para lograr el funcionamiento del sistema integrado. Además, se mencionan diversos aspectos relacionados con la plataforma de Arduino como son: conexiones, librerías y la programación de los ficheros que gestionan los dispositivos.

6.1.- Asterisk

En esta sección se detallan los aspectos más relevantes correspondientes a la implementación del prototipo en la centralita de Asterisk, ya que se trata del elemento central de este proyecto.

6.1.1.- Dialplan

El *dialplan* es el elemento clave de Asterisk, en este se define el orden del plan de acciones que sigue una llamada realizada por un usuario. Dicho usuario debe estar previamente configurado en el fichero *pjsip.conf* para tener acceso al sistema. Cabe recordar que el *dialplan* se configura en el fichero *extensions.conf*.

Para conseguir una mayor organización dentro del fichero la configuración del *dialplan* (ver Anexo 10.3.1) se ha estructurado definiendo los siguientes contextos. Cada uno se encarga de una función específica.

- home
- inmediata-ivr
- acciones
- futura-ivr
- continuar_fin-ivr
- google_stt
- call_interna
- google_stt_test

Antes de describir brevemente la función desempeñada por cada uno de los contextos que se han definido, es importante mencionar que los audios que se utilizan para dar instrucciones al usuario a través de los diferentes IVR deben estar en el directorio /var/lib/asterisk/sounds/. En este caso se ha creado una nueva carpeta dentro de este directorio denominado /custom para alojar los audios citados y así se puedan diferenciar de los que trae Asterisk por defecto.

Los ficheros de audio tienen extensión ".wav" y han sido grabados con la herramienta Audacity. Antes de comenzar la grabación de los mismos, se han configurado los parámetros estándar de audio para la telefonía (*Signed* 16 bits PCM, Mono, 8000 Hz).

6.1.1.1.- Home

Es el contexto inicial donde entra la llamada cuando el usuario marca la extensión 40 para interactuar con el hogar domotizado. Según la solicitud de acción del usuario el sistema lo redirige a los contextos [inmediata-ivr] o [futura-ivr] para continuar el diálogo. En este sentido, se ha implementado el acceso a estos contextos a través del uso de un IVR basado en el reconocimiento de voz.

Si el sistema no encuentra alguna coincidencia en el comando de voz dicho por el usuario, o este permanece en silencio, se reproduce un audio que indica que el comando de voz es inválido o que la respuesta es vacía en sus respectivos casos. Seguidamente redirige al usuario al inicio de este contexto para comenzar nuevamente el proceso.

Este contexto también permite activar el modo automático en el hogar tras la marcación de la extensión 41 por parte del usuario. Además, contiene la extensión 30 que le permite al usuario realizar una comprobación del reconocimiento de su voz a través de la API STT de Google *Cloud*.

6.1.1.2.- Inmediata-ivr

A este contexto se accede mediante la pronunciación del comando "ahora". Tras esto, se reproduce un audio pregrabado que solicita al usuario decir la acción que desea realizar sobre el hogar domotizado. En este contexto se definen el conjunto de comandos de voz que se corresponden con las acciones válidas que el sistema es capaz de procesar. En función del texto devuelto por la API STT se redirige la llamada al contexto [acciones]. Si el sistema no encuentra coincidencia o el usuario permanece en silencio se reproduce el audio correspondiente para cada error y se redirige al usuario al inicio de este contexto.

6.1.1.3.- Acciones

En este contexto se define una extensión para cada una de las acciones válidas del sistema y se mapean con una combinación de tres caracteres alfanuméricos, exceptuando la acción de "enviar mensaje" que lleva un procesamiento diferenciado. Luego estos caracteres (ya sea la combinación de los tres caracteres alfanuméricos correspondientes o la cadena de caracteres del mensaje) se pasan como parámetro a un *script* para realizar la petición HTTP al servidor de Arduino a través de la aplicación AGI. Por último, se redirige al usuario al contexto [continuar_fin-ivr] para continuar o finalizar la llamada.

En el caso de la acción de "enviar mensaje" de forma inmediata, se reproduce un audio para solicitar al usuario dicho mensaje. Tras esto, se realiza una comparación para detectar si la calidad del texto devuelto por Google es insuficiente y así garantizar que el mensaje enviado sea coherente. Cuando la probabilidad de acierto de la transcripción de Google es elevada, entonces se guarda dicha transcripción en una variable que inmediatamente se pasa como parámetro a la petición HTTP realizada al servidor de Arduino.

Para el caso de la acción de "enviar mensaje" en un instante futuro, la grabación de dicho mensaje se realiza en el contexto [futura-ivr] donde se guarda en un fichero. Una vez transcurrido el tiempo de la ejecución de la acción, en la extensión correspondiente se lee del fichero que contiene la cadena de caracteres del mensaje para enviarlo como parámetro del *script* que lanza la petición HTTP al servidor de Arduino.

Se puede examinar el Anexo 10.3.1, la Figura 10.28 para una mejor comprensión de lo descrito anteriormente.

6.1.1.4.- Futura-ivr

A este contexto se accede mediante la pronunciación del comando "después". Inmediatamente se reproduce un audio pregrabado que solicita al usuario decir la acción que desea realizar sobre el hogar domotizado. Del mismo modo que en el contexto [inmediata-ivr], se definen el conjunto de comandos de voz que se corresponden con las acciones válidas del sistema y se realiza una asignación de la extensión que se corresponde a las acciones que se definen en el contexto [acciones] (para ver los detalles consultar el Anexo 10.3.1, la Figura 10.29).

Si el sistema no encuentra coincidencia o el usuario permanece en silencio se reproduce el audio correspondiente para cada error y se redirige al usuario al inicio de este contexto donde se le solicita que diga nuevamente la acción que desea realizar.

Como se ha comentado anteriormente, para la acción de "enviar mensaje" es necesario almacenar el mensaje del usuario en un fichero para que pueda ser leído posteriormente cuando llegue el momento de la ejecución de la acción.

Una vez se ha realizado la asignación de la extensión correspondiente a cada acción, se reproducen dos audios para solicitar la fecha y la hora en la que se desea ejecutar la acción. En este sentido, se invocan los *scripts* que chequean si los formatos de fecha y hora introducidos por el usuario son correctos.

A continuación, se invoca un *script* a través de AGI donde se le pasa como parámetro la fecha, la hora y la extensión de la acción. Con esta información el *script* crea un *call file* que se ejecuta en el tiempo futuro indicado. Seguidamente el sistema reproduce un audio para indicar que la acción se ha planificado correctamente y redirige al usuario al contexto [continuar_fin-ivr].

6.1.1.5.- Continuar_fin-ivr

En este contexto se reproduce un audio explicativo que le permite al usuario continuar la interacción con el sistema mediante la pronunciación del comando "uno". En este caso se redirige al usuario al inicio del contexto [home] para comenzar nuevamente todo el proceso. También se puede finalizar la llamada mediante la pronunciación del comando "dos".

6.1.1.6.- Google_stt

Este contexto es el que establece la comunicación con los servicios de Google *Cloud* cada vez que se precisa que el usuario introduzca al sistema un comando verbalmente. A través de la aplicación AGI se invoca un *script* creado por Lefteris Zafiris [33] en lenguaje Perl que se conecta a la API STT de Google para efectuar la transcripción.

6.1.1.7.- Call_interna

Este contexto se ha definido únicamente para uso del *script* que crea los *call files*. En un *call file* el campo "*Channel*" es obligatorio, habitualmente en este campo se sitúa el canal PJSIP del usuario que se desea llamar. En este caso no se efectúa una llamada directamente al usuario, sino que se llama a un canal local de Asterisk (transparente para el usuario) para poder ejecutar el resto del *call file*.

6.1.1.8.- Google_stt_test

A este contexto se accede mediante la marcación de la extensión 30. Este contexto se ha creado para proveerle al usuario de una herramienta que le permita comprobar el correcto reconocimiento de su voz a través de la API STT. Desde el *dialplan* de Asterisk se invoca un *script* que se conecta a la API para transcribir los mensajes de voz del usuario. Se envía la transcripción y la probabilidad de ser correcta al chat del Telegram del usuario para que pueda visualizar el resultado.

6.1.2.- Descripción de los *Scripts*

En este apartado se describen los *scripts* utilizados en la implementación de este proyecto. Los que se invocan desde el *dialplan* de Asterisk se han programado en el lenguaje de programación Bash, exceptuando el *script* que permite realizar la transcripción de voz a texto que está escrito en Perl. Además, se programa un *script* en PHP para invocar otros *scripts* en Bash cuando el hogar domotizado envía eventos a la centralita de Asterisk.

A continuación, una breve descripción de cada uno, para consultar detalles del código fuente ver el Anexo correspondiente:

6.1.2.1.- *Script* de reconocimiento de voz

Nombre del archivo: speech-recog.agi

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca múltiples veces en el *dialplan*, se utiliza para grabar los comandos de voz del usuario.

Uso: agi(speech-recog.agi, [lang], [timeout], [intkey], [NOBEEP])

Entrada: Se le pasa como parámetros de entrada el lenguaje en el que se detecta el audio para la transcripción [lang], el tiempo de silencio antes de dejar de grabar [timeout] (establecerlo en -1 desactiva la detección del silencio), la tecla de interrupción, por defecto es # [intkey] y el parámetro [NOBEEP] si se desea no activar un *beep* previo para indicar el inicio de la grabación, de lo contrario se deja este campo vacío.

Salida: devuelve dos variables:

- *utterance*: la cadena de texto transcrita.
- *confidence*: Un valor entre 0 y 1 que indica la probabilidad de un reconocimiento correcto. Los valores superiores a 0,90 suelen significar que el texto resultante es correcto.

6.1.2.2.- *Script* de petición HTTP

Nombre del archivo: request.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca para lanzar una petición HTTP al servidor Arduino con la opción que corresponde a la acción que desea ejecutar el usuario sobre el hogar domotizado.

Uso: AGI(request.sh, ${State})

Entrada: El argumento ${State} es una variable de canal que se establece con un valor de tres caracteres alfanuméricos en función de la orden solicitada por el usuario, exceptuando la orden de "enviar mensaje" donde el argumento ${State} toma otra sintaxis ya que esta opción requiere de un tratamiento diferenciado. En la Figura 6.1 se aprecia un fragmento del *dialplan* donde se puede observar lo comentado anteriormente. En el caso de la acción "enviar mensaje" se utiliza el conjunto de "sms=" para identificar el inicio del mensaje y el carácter "%" para identificar el final del mensaje en el servidor de Arduino.

```
...
exten => 11, 1, NoOp (Abrir puerta)
same => n, Set(State= OPE)
same => n,Goto(25,1)

exten => 12, 1, NoOp (Cerrar puerta)
same => n, Set(State= CLO)
same => n,Goto(25,1)

exten => 13, 1, NoOp (Enviar mensaje)
...
same => n,Set(State= sms=${utterance}%)
same => n,Goto(25,1)

...
exten => 25,1, NoOp (Común a todas las acciones)
same => n, AGI(request.sh, ${State})
...
```

Figura 6.1.- Fragmento del *dialplan*: uso del *script* "request.sh".

Procesa: Dentro del *script* se parsea el argumento de entrada sustituyendo los espacios en blanco por el carácter "@", esto solo tiene sentido para la acción "enviar mensaje" ya que el servidor Arduino espera recibir esta petición con el siguiente formato: "sms=palabra1@palabra2@palabra3@...palabraN%". Para el resto de acciones no tiene repercusión ya que se envía una cadena de tres caracteres alfanuméricos sin espacios.

Salida: lanza la petición HTTP a la IP del servidor de Arduino con el argumento correspondiente.

Para una mejor comprensión del *script* descrito, se recomienda consultar el código fuente en el Anexo 10.3.2, la Figura 10.32.

6.1.2.3.- *Script* para verificar formato de fecha

Nombre del archivo: checkdate.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca cuando el usuario desea planificar una acción en un tiempo futuro que el sistema le solicita la fecha. Su función es chequear que la fecha introducida tenga un formato válido y coherente.

Uso: AGI(checkdate.sh, ${fecha})

Entrada: El parámetro de entrada es la fecha en el formato (dd de mm de aaaa).

Procesa: Se convierte la fecha de entrada al formato deseado (compatible con los *call files*) y se chequea que sea válida con el comando "date" de *bash*.

Salida: Se crea el fichero /tmp/checkdate.txt. Si la fecha es válida se guarda dentro del fichero la fecha en el formato (aaaammdd), sino se guarda el fichero vacío.

Para ver más detalles de este *script*, se puede consultar el código fuente en el Anexo 10.3.2, la Figura 10.33.

6.1.2.4.- *Script* para verificar formato de hora

Nombre del archivo: checktime.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca cuando el usuario desea planificar una acción en un tiempo futuro que el sistema le solicita la hora. Su función es chequear que la hora introducida tenga un formato válido.

Uso: AGI(checkdate.sh, ${horario})

Entrada: El parámetro de entrada es la hora en el formato (hh:mm / hh y mm).

Procesa: Se realiza un tratamiento de la hora para llevarla al formato deseado (compatible con los *call files*) para cualquiera de los dos formatos de entrada citados anteriormente. Además, se garantiza que tanto la hora como los minutos tengan dos dígitos. Por último, se chequea que sea válida con el comando "date" de *bash*.

Salida: Se crea el fichero /tmp/checktime.txt. Si la hora es válida se guarda dentro del fichero la hora en el formato (hhmm), sino se guarda el fichero vacío.

Para más información al respecto se puede ver el código fuente en el Anexo 10.3.2, la Figura 10.34.

6.1.2.5.- *Script* para guardar grabar mensaje

Nombre del archivo: anotamensaje.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca cuando el usuario solicita enviar un mensaje en un instante futuro. Su función es almacenar el mensaje para ser leído posteriormente cuando transcurra el tiempo señalado por el usuario.

Uso: AGI(anotamensaje.sh, ${mensj})

Entrada: La entrada tiene la sintaxis sms=cadenadelmensaje%.

Salida: Crea el fichero /tmp/anotamensaje.txt donde se guarda el mensaje que se desea enviar al panel electrónico del hogar.

Se puede consultar el código fuente en el Anexo 10.3.2, la Figura 10.35.

6.1.2.6.- *Script* para crear *call file* futuros

Nombre del archivo: creating_callfile.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca cuando el usuario solicita planificar una acción en un tiempo futuro. El *script* genera un *call file* con la información proporcionada por el usuario, el cual se ejecuta en el tiempo futuro indicado.

Uso: AGI(creating_callfile.sh, ${fecha}, ${horario}, ${ext})

Entrada: Los parámetros de entrada son la fecha, el horario y la extensión que se corresponde con la opción especificada por el usuario.

Salida: se genera el *call file* con el formato (aaaammdd-hhmm.call) necesario para ejecutar la orden planificada por el usuario en el instante futuro indicado.

Para una mayor claridad en lo que se ha descrito, se recomienda consultar el código fuente en el Anexo 10.3.2, la Figura 10.36.

6.1.2.7.- *Script* para enviar mensaje al chat de Telegram

Nombre del archivo: telegram.sh

Ubicación: /var/lib/asterisk/agi-bin/

Descripción: Este *script* se invoca desde el servidor Apache cada vez que se desea notificar al usuario de algún evento o sencillamente enviarle algún mensaje de confirmación de que la acción ha sido realizada.

Entrada: Recibe como parámetro de entrada el mensaje que el sistema envía al chat de Telegram del usuario.

Procesa: Dentro del *script* se inserta el *token* de acceso que proporciona el *bot* de Telegram y el identificador del chat (ver Anexo 10.1.5).

Salida: Se lanza una petición HTTP a la URL de la API de Telegram con el mensaje pasado como argumento.

Para ver los detalles de este *script*, se puede consultar el código fuente en el Anexo 10.3.2, la Figura 10.37. Cabe comentar que en el código de este *script* se ha anonimizado parte de la información relativa al *token* y al id del chat.

6.1.2.8.- *Script* de notificación desde Arduino

Nombre del archivo: notif.php

Ubicación: /var/www/smarthome/

Descripción: Este *script* es ejecutado desde el servidor Arduino cuando suceden varios eventos en el hogar. La petición HTTP de Arduino al servidor Apache lleva como argumento un número que se corresponde a varios eventos definidos en el hogar. La función de este *script* es notificar al usuario a su chat de Telegram.

Entrada: Recibe una variable que puede tomar valores de [0-9] que viene en la petición GET del servidor Arduino. Para el caso de las notificaciones relacionadas con el sensor de temperatura, también recibe como parámetro de entrada la medición del sensor.

Procesa: Se definen los mensajes que se envían al chat del usuario en función del evento que haya ocurrido en el hogar.

Salida: Se ejecuta el *script* "telegram.sh" pasando como parámetro el mensaje correspondiente al evento que se desee notificar.

Para más información al respecto, se puede ver el código fuente en el Anexo 10.3.2, la Figura 10.38.

6.2.- Arduino

En esta sección se describe brevemente los aspectos principales correspondientes a la implementación en la plataforma de Arduino.

6.2.1.- Conexiones

Primeramente, se realiza una selección de los distintos sensores y actuadores que son empleados en el proyecto, definiendo la funcionalidad que representan dentro del hogar domotizado. Una vez escogidos los dispositivos se conectan físicamente a los puertos de la placa de Arduino con cables PnP de tres o cuatro pines según el módulo en cuestión. A continuación, en la Tabla 6.1, aparece la relación de las conexiones.

Nº	Dispositivo	Puerto	Funcionalidad
1	Módulo LED luz blanca 1	D2, digital de 3 pines	Luces
2	Módulo LED luz roja	D3, digital de 3 pines	Calefactor
3	Módulo LED luz blanca 2	D4, digital de 3 pines	Aire Acondicionado
4	Módulo *Fan* Motor	D5, digital de 3 pines, el pin INA conectado a D5, pin INB conectado a pin 6 de Arduino	Calefactor y Aire acondicionado

5	Módulo *Buzzer* Activo	D7, digital de 3 pines	Puerta Garaje
6	Módulo *Buzzer* Pasivo	D8, digital de 3 pines	Alarma
7	Micro Servomotor 2	D9, digital de 3 pines	Aspersor
8	Micro Servomotor 1	D10, digital de 3 pines	Puerta Garaje
9	Módulo PIR	D11, digital de 3 pines	Alarma
10	Módulo LED luz verde	D12, digital de 3 pines	Puerta Garaje
11	Módulo LED luz amarilla 1	D13, digital de 3 pines	Puerta Garaje
12	Módulo LED luz amarilla 2	A0, analógico de 3 pines	Alarma
13	Módulo DTH11	A1, analógico de 3 pines	Temperatura
14	Módulo RGB	RGB, digital de 4 pines	Luces navideñas
15	LCD 1602	I2C-1, digital de 4 pines	Panel electrónico

Tabla 6.1.- Conexiones de los dispositivos a los puertos de la placa de Arduino.

6.2.2.- Ficheros

Los programas en Arduino son un fichero con extensión "*ino*" contenido dentro de una carpeta de igual nombre que el fichero, aunque también es posible estructurar el programa en varios ficheros. En tal caso, el programa principal es el que tiene que tener el mismo nombre de la carpeta del proyecto.

El código desarrollado está estructurado en un programa principal y ocho subprogramas, para ver el código fuente se puede consultar el fichero comprimido subido como Anexo al portal web. El programa principal se encarga de establecer la conexión a la WiFi y procesar las peticiones HTTP recibidas desde Asterisk. Estas peticiones incluyen en su cabecera una combinación de tres caracteres alfanuméricos que mapean cada una de las funcionalidades implementadas en el hogar. En la Figura 6.2 se muestra dos capturas de dos peticiones HTTP recibidas en Arduino donde se señala dicha combinación. En función de la opción recibida en la petición se realiza una u otra acción.

```
GET /ALL HTTP/1.1
Host: 192.168.1.131
User-Agent: curl/7.68.0
Accept: */*
```

```
GET /RGB HTTP/1.1
Host: 192.168.1.131
User-Agent: curl/7.68.0
Accept: */*
```

Figura 6.2.- Ejemplo de peticiones HTTP recibidas en Arduino.

Por otra parte, los ocho subprogramas se encargan de procesar de manera independiente cada uno de los dispositivos que representa una funcionalidad dentro del hogar. En la Tabla 6.2 se resume la función de cada fichero junto con las dependencias de las librerías que necesitan.

Fichero	Función	Librerías usadas
smarthomeMainprogram	Conexión con el cliente Asterisk a través de la WiFi. Procesamiento de peticiones HTTP procedentes de Asterisk. Conexión con el servidor Apache para notificar al usuario.	"WiFiEsp.h" <RingBuf.h>
lightProcessing	Procesa el encendido y apagado de las luces del hogar. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	--
christmasProcessing	Genera patrón de colores aleatorios. Procesa el encendido y apagado de las luces navideñas. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	"NoDelay.h"
coldhotProcessing	Procesa mediante señales PWM las velocidades del motor y las intensidades de los LEDs que simulan el comportamiento de un calefactor y un aire acondicionado. Procesa el encendido y apagado de estos dispositivos. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	--
sprinklerProcessing	Procesa el control del ángulo de rotación del servomotor para simular el efecto de un aspersor. Procesa el encendido y apagado del aspersor. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	<Servo.h> "NoDelay.h"
doorProcessing	Procesa la apertura y cierre de una puerta mediante el control del ángulo de rotación de otro servomotor que emula este efecto. Procesa la emisión de una señal visual y sonora durante el tiempo de apertura y cierre de la puerta. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	<Servo.h> "NoDelay.h"
displayProcessing	Procesa el mensaje recibido desde Asterisk compuesto por una cadena máxima de 32 caracteres y lo muestra en el *display* LCD1602 que simula un panel electrónico de recordatorios.	<LiquidCrystal_I2C.h> <Wire.h> "NoDelay.h"

	Procesa el encendido y apagado del panel. Llama a la función que se comunica con Apache para notificar que se ha realizado la acción.	
alarmProcessing	Procesa la activación de la alarma generando una señal sonora y visual que emulan una sirena. Llama a la función que se comunica con Apache para notificar al usuario de la activación de la alarma.	"NoDelay.h"
temperatProcessing	Procesa el encendido y apagado automático del calefactor y aire acondicionado en función de las mediciones del sensor de temperatura. Llama a la función que se comunica con Apache para notificar al usuario de los cambios automáticos.	"dht.h"

Tabla 6.2.- Ficheros de Arduino.

7.- VALIDACIÓN Y RESULTADOS

En este capítulo se refleja el conjunto de pruebas practicadas sobre el sistema y los resultados obtenidos. El entorno de desarrollo de las pruebas es un escenario como el que se muestra en la Figura 7.1.

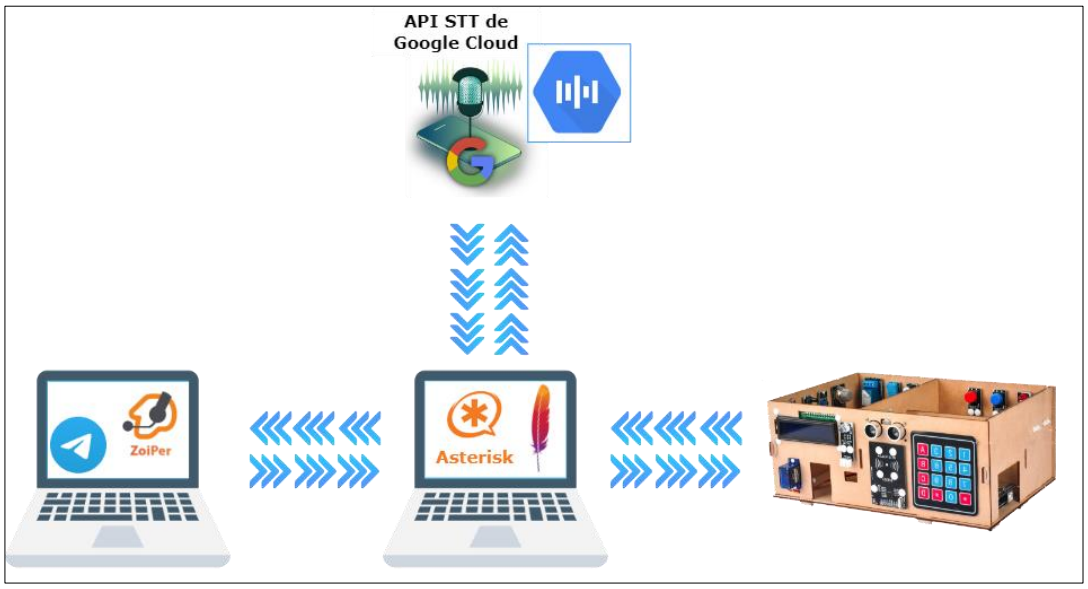


Figura 7.1.- Entorno de desarrollo de las pruebas.

El objetivo es validar que la solución desarrollada funciona adecuadamente en base a los requisitos establecidos en los apartados (4.1 y 4.2) y a los casos de uso declarados en el apartado 5.2. En cada prueba se especifican los siguientes campos:

1. **ID:** Identificador único de la prueba. Puede ser:
   - CPV-XX para los casos de prueba válidos
   - CPI-XX para los casos de prueba inválidos

   Siendo XX el número secuencial de la prueba.
2. **Prueba:** Nombre de la prueba.
3. **Descripción:** Breve descripción de la prueba.
4. **Resultado esperado:** Descripción del resultado que se espera obtener.
5. **Resultado obtenido:** Registro del resultado obtenido después de la realización de la prueba.
6. **Validación:** Revela si el resultado de la prueba ha sido favorable o no.
7. **Correspondencia**: Indica la correspondencia con el caso de uso.

7.1.- Casos de Pruebas Válidos

En este apartado se describen los casos de prueba válidos realizados sobre los diferentes subsistemas definidos:

- Subsistema de Acciones Inmediatas
- Subsistema de Acciones Futuras
- Subsistema de Acciones Automáticas
- Subsistema de Acciones Extras

7.1.1.- Subsistema de Acciones Inmediatas

En este subapartado se recogen las distintas pruebas realizadas sobre el Subsistema de Acciones Inmediatas:

Prueba	**Encender luces**	**ID**	**CPV-01**
Descripción	En esta prueba se verifica que las luces del hogar se encienden cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender las luces del hogar.		
Resultado obtenido	El sistema enciende las luces del hogar.		
Validación	Correcto		
Dependencias	CU-01: Encender luces		

Tabla 7.1.- CPV-01: Encender luces.

Prueba	**Apagar luces**	**ID**	**CPV-02**
Descripción	En esta prueba se verifica que las luces del hogar se apagan cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe apagar las luces del hogar.		
Resultado obtenido	El sistema apaga las luces del hogar.		
Validación	Correcto		
Dependencias	CU-02: Apagar luces		

Tabla 7.2.- CPV-02: Apagar luces.

Prueba	Encender luces de Navidad	ID	CPV-03
Descripción	En esta prueba se verifica que las luces de Navidad del hogar se encienden cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender las luces de Navidad del hogar.		
Resultado obtenido	El sistema enciende las luces de Navidad del hogar.		
Validación	Correcto		
Dependencias	CU-03: Encender luces de Navidad		

Tabla 7.3.- CPV-03: Encender luces de Navidad.

Prueba	Apagar luces de Navidad	ID	CPV-04
Descripción	En esta prueba se verifica que las luces de Navidad del hogar se apagan cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe apagar las luces de Navidad del hogar.		
Resultado obtenido	El sistema apaga las luces de Navidad del hogar.		
Validación	Correcto		
Dependencias	CU-04: Apagar luces de Navidad		

Tabla 7.4.- CPV-04: Apagar luces de Navidad.

Prueba	Encender calefactor temperatura baja	ID	CPV-05
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura baja cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el calefactor en temperatura baja.		
Resultado obtenido	El sistema enciende el calefactor en temperatura baja.		
Validación	Correcto		
Dependencias	CU-05: Encender calefactor temperatura baja		

Tabla 7.5.- CPV-05: Encender calefactor temperatura baja.

Prueba	Encender calefactor temperatura media	ID	CPV-06
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura media cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el calefactor en temperatura media.		
Resultado obtenido	El sistema enciende el calefactor en temperatura media.		
Validación	Correcto		
Dependencias	CU-06: Encender calefactor temperatura media		

Tabla 7.6.- CPV-06: Encender calefactor temperatura media.

Prueba	Encender calefactor temperatura alta	ID	CPV-07
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura alta cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el calefactor en temperatura alta.		
Resultado obtenido	El sistema enciende el calefactor en temperatura alta.		
Validación	Correcto		
Dependencias	CU-07: Encender calefactor temperatura alta		

Tabla 7.7.- CPV-07: Encender calefactor temperatura alta.

Prueba	Apagar calefactor	ID	CPV-08
Descripción	En esta prueba se verifica que el calefactor se apaga cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe apagar el calefactor.		
Resultado obtenido	El sistema apaga el calefactor.		
Validación	Correcto		
Dependencias	CU-08: Apagar calefactor		

Tabla 7.8.- CPV-08: Apagar calefactor.

Prueba	Encender aire acondicionado intensidad baja	ID	CPV-09
Descripción	En esta prueba se verifica el aire acondicionado se enciende con intensidad baja cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el aire acondicionado con intensidad baja.		
Resultado obtenido	El sistema enciende el aire acondicionado con intensidad baja.		
Validación	Correcto		
Dependencias	CU-09: Encender aire acondicionado intensidad baja		

Tabla 7.9.- CPV-09: Encender aire acondicionado intensidad baja.

Prueba	Encender aire acondicionado intensidad media	ID	CPV-10
Descripción	En esta prueba se verifica el aire acondicionado se enciende con intensidad media cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el aire acondicionado con intensidad media.		
Resultado obtenido	El sistema enciende el aire acondicionado con intensidad media.		
Validación	Correcto		
Dependencias	CU-10: Encender aire acondicionado intensidad media		

Tabla 7.10.- CPV-10: Encender aire acondicionado intensidad media.

Prueba	Encender aire acondicionado intensidad alta	ID	CPV-11
Descripción	En esta prueba se verifica el aire acondicionado se enciende con intensidad alta cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el aire acondicionado con intensidad alta.		
Resultado obtenido	El sistema enciende el aire acondicionado con intensidad alta.		
Validación	Correcto		
Dependencias	CU-11: Encender aire acondicionado intensidad alta		

Tabla 7.11.- CPV-11: Encender aire acondicionado intensidad alta.

Prueba	Apagar aire acondicionado	ID	CPV-12
Descripción	En esta prueba se verifica el aire acondicionado se apaga cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe apagar el aire acondicionado.		
Resultado obtenido	El sistema apaga el aire acondicionado.		
Validación	Correcto		
Dependencias	CU-12: Apagar aire acondicionado		

Tabla 7.12.- CPV-12: Apagar aire acondicionado.

Prueba	Encender aspersor	ID	CPV-13
Descripción	En esta prueba se verifica que el aspersor se enciende cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe encender el aspersor.		
Resultado obtenido	El sistema enciende el aspersor.		
Validación	Correcto		
Dependencias	CU-01: Encender aspersor		

Tabla 7.13.- CPV-13: Encender aspersor.

Prueba	Apagar aspersor	ID	CPV-14
Descripción	En esta prueba se verifica que el aspersor se apaga cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe apagar el aspersor.		
Resultado obtenido	El sistema enciende el aspersor.		
Validación	Correcto		
Dependencias	CU-01: Encender aspersor		

Tabla 7.14.- CPV-14: Encender aspersor.

Prueba	Abrir puerta	ID	CPV-15
Descripción	En esta prueba se verifica que la puerta se abre cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe abrir la puerta.		
Resultado obtenido	El sistema abre la puerta.		
Validación	Correcto		
Dependencias	CU-15: Abrir puerta		

Tabla 7.15.- CPV-15: Abrir puerta.

Prueba	Cerrar puerta	ID	CPV-16
Descripción	En esta prueba se verifica que la puerta se cierra cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe cerrar la puerta.		
Resultado obtenido	El sistema cierra la puerta.		
Validación	Correcto		
Dependencias	CU-16: Cerrar puerta		

Tabla 7.16.- CPV-16: Abrir puerta.

Prueba	Enviar mensaje	ID	CPV-17
Descripción	En esta prueba se verifica que el mensaje se ha mostrado correctamente en el panel electrónico.		
Resultado esperado	El sistema debe mostrar el mensaje en el panel electrónico.		
Resultado obtenido	El sistema muestra el mensaje en el panel electrónico.		
Validación	Correcto		
Dependencias	CU-17: Enviar mensaje		

Tabla 7.17.- CPV-17: Enviar mensaje.

Prueba	Apagar panel electrónico	ID	CPV-18
Descripción	En esta prueba se verifica que el panel electrónico borra el mensaje y se apaga.		
Resultado esperado	El sistema debe borrar el mensaje y apagar el panel electrónico.		
Resultado obtenido	El sistema borra el mensaje y apaga el panel electrónico.		
Validación	Correcto		
Dependencias	CU-18: Apagar panel electrónico		

Tabla 7.18.- CPV-18: Apagar panel electrónico.

Prueba	Activar alarma	ID	CPV-19
Descripción	En esta prueba se verifica que la alarma se activa cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe activar la alarma.		
Resultado obtenido	El sistema activa la alarma.		
Validación	Correcto		
Dependencias	CU-19: Activar alarma		

Tabla 7.19.- CPV-19: Activar alarma.

Prueba	Desactivar alarma	ID	CPV-20
Descripción	En esta prueba se verifica que la alarma se desactiva cuando el usuario ordena esta acción.		
Resultado esperado	El sistema debe desactivar la alarma.		
Resultado obtenido	El sistema desactiva la alarma.		
Validación	Correcto		
Dependencias	CU-20: Desactivar alarma		

Tabla 7.20.- CPV-20: Desactivar alarma.

Prueba	Apagar todo	ID	CPV-21
Descripción	En esta prueba se verifica que se apagan todos los dispositivos del hogar.		
Resultado esperado	El sistema debe apagar todos los dispositivos del hogar.		
Resultado obtenido	El sistema apaga todos los dispositivos del hogar.		
Validación	Correcto		
Dependencias	CU-21: Apagar todo		

Tabla 7.21.- CPV-21: Apagar todo.

7.1.2.- Subsistema de Acciones Futuras

En este subapartado se recogen las distintas pruebas realizadas sobre el Subsistema de Acciones Futuras:

Prueba	Programar encender luces	ID	CPV-22
Descripción	En esta prueba se verifica que las luces se encienden en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender las luces en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende las luces en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-22: Programar encender luces		

Tabla 7.22.- CPV-22: Programar encender luces.

Prueba	Programar apagar luces	ID	CPV-23
Descripción	En esta prueba se verifica que las luces se apagan en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar las luces en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga las luces en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-23: Programar apagar luces		

Tabla 7.23.- CPV-23: Programar apagar luces.

Prueba	Programar encender luces de Navidad	ID	CPV-24
Descripción	En esta prueba se verifica que las luces de Navidad se enciendan en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender las luces de Navidad en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende las luces de Navidad en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-24: Programar encender luces de Navidad		

Tabla 7.24.- CPV-24: Programar encender luces de Navidad.

Prueba	Programar apagar luces de Navidad	ID	CPV-25
Descripción	En esta prueba se verifica que las luces de Navidad se apagan en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar las luces de Navidad en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga las luces de Navidad en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-25: Programar apagar luces de Navidad		

Tabla 7.25.- CPV-25: Programar apagar luces de Navidad.

Prueba	Programar encender calefactor temperatura baja	ID	CPV-26
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura baja en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el calefactor en temperatura baja en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el calefactor en temperatura baja en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-26: Programar encender calefactor temperatura baja.		

Tabla 7.26.- CPV-26: Programar encender calefactor temperatura baja.

Prueba	Programar encender calefactor temperatura media	ID	CPV-27
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura media en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el calefactor en temperatura media en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el calefactor en temperatura media en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-27: Programar encender calefactor temperatura media		

Tabla 7.27.- CPV-27: Programar encender calefactor temperatura media.

Prueba	Programar encender calefactor temperatura alta	ID	CPV-28
Descripción	En esta prueba se verifica que el calefactor se enciende en temperatura alta en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el calefactor en temperatura alta en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el calefactor en temperatura alta en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-28: Programar encender calefactor temperatura alta		

Tabla 7.28.- CPV-28: Programar encender calefactor temperatura alta.

Prueba	Programar apagar calefactor	ID	CPV-29
Descripción	En esta prueba se verifica que el calefactor se apaga en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar el calefactor en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga el calefactor en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-29: Programar apagar calefactor		

Tabla 7.29.- CPV-29: Programar apagar calefactor.

Prueba	Programar encender aire acondicionado intensidad baja	ID	CPV-30
Descripción	En esta prueba se verifica que el aire acondicionado se enciende en intensidad baja en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el aire acondicionado en intensidad baja en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el aire acondicionado en intensidad baja en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-30: Programar encender aire acondicionado intensidad baja		

Tabla 7.30.- CPV-30: Programar encender aire acondicionado intensidad baja.

Prueba	Programar encender aire acondicionado intensidad media	ID	CPV-31
Descripción	En esta prueba se verifica que el aire acondicionado se enciende en intensidad media en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el aire acondicionado en intensidad media en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el aire acondicionado en intensidad media en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-31: Programar encender aire acondicionado intensidad media		

Tabla 7.31.- CPV-31: Programar encender aire acondicionado intensidad media.

Prueba	Programar encender aire acondicionado intensidad alta	ID	CPV-32
Descripción	En esta prueba se verifica que el aire acondicionado se enciende en intensidad alta en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el aire acondicionado en intensidad alta en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el aire acondicionado en intensidad alta en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-32: Programar encender aire acondicionado intensidad alta		

Tabla 7.32.- CPV-32: Programar encender aire acondicionado intensidad alta.

Prueba	Programar apagar aire acondicionado	ID	CPV-33
Descripción	En esta prueba se verifica que el aire acondicionado se apaga en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar el aire acondicionado en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga el aire acondicionado en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-33: Programar apagar aire acondicionado		

Tabla 7.33.- CPV-33: Programar apagar aire acondicionado.

Prueba	Programar encender aspersor	ID	CPV-34
Descripción	En esta prueba se verifica que el aspersor se enciende en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe encender el aspersor en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema enciende el aspersor en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-34: Programar encender aspersor		

Tabla 7.34.- CPV-34: Programar encender aspersor.

Prueba	Programar apagar aspersor	ID	CPV-35
Descripción	En esta prueba se verifica que el aspersor se apaga en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar el aspersor en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga el aspersor en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-35: Programar apagar aspersor		

Tabla 7.35.- CPV-35: Programar apagar aspersor.

Prueba	Programar abrir puerta	ID	CPV-36
Descripción	En esta prueba se verifica que la puerta se abre en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe abrir la puerta en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema abre la puerta en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-36: Programar abrir puerta		

Tabla 7.36.- CPV-36: Programar abrir puerta.

Prueba	Programar cerrar puerta	ID	CPV-37
Descripción	En esta prueba se verifica que la puerta se cierra en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe cerrar la puerta en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema cierra la puerta en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-37: Programar cerrar puerta		

Tabla 7.37.- CPV-37: Programar cerrar puerta.

Prueba	Programar enviar mensaje	ID	CPV-38
Descripción	En esta prueba se verifica que el mensaje se ha mostrado correctamente en el panel electrónico en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe mostrar el mensaje en el panel electrónico en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema muestra el mensaje en el panel electrónico en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-38: Programar enviar mensaje		

Tabla 7.38.- CPV-38: Programar enviar mensaje.

Prueba	Programar apagar panel electrónico	ID	CPV-39
Descripción	En esta prueba se verifica que el panel electrónico se apaga en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar el panel electrónico en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga el panel electrónico en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-39: Programar apagar panel electrónico		

Tabla 7.39.- CPV-39: Programar apagar panel electrónico.

Prueba	Programar activar alarma	ID	CPV-40
Descripción	En esta prueba se verifica que la alarma se activa en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe activar la alarma en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema activa la alarma en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-40: Programar activar alarma		

Tabla 7.40.- CPV-40: Programar activar alarma.

Prueba	Programar desactivar alarma	ID	CPV-41
Descripción	En esta prueba se verifica que la alarma se desactiva en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe desactivar la alarma en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema desactiva la alarma en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-41: Programar desactivar alarma		

Tabla 7.41.- CPV-41: Programar desactivar alarma.

Prueba	Programar apagar todo	ID	CPV-42
Descripción	En esta prueba se verifica que se apagan todos los dispositivos del hogar en la fecha y hora programadas por el usuario.		
Resultado esperado	El sistema debe apagar todos los dispositivos en el instante futuro especificado por el usuario.		
Resultado obtenido	El sistema apaga todos los dispositivos en el instante futuro especificado por el usuario.		
Validación	Correcto		
Dependencias	CU-42: Programar apagar todo		

Tabla 7.42.- CPV-42: Programar apagar todo.

7.1.3.- Subsistema de Acciones Automáticas

En este subapartado se recogen las distintas pruebas realizadas sobre el Subsistema de Acciones Automáticas:

Prueba	Encender calefactor	ID	CPV-43
Descripción	En esta prueba se verifica que el calefactor se enciende de forma automática a temperatura media cuando el sensor DHT11 obtiene lecturas por debajo de 10 ºC.		
Resultado esperado	El sistema debe encender el calefactor a temperatura media de forma automática si el sensor DHT11 obtiene lecturas por debajo de 10 ºC.		
Resultado obtenido	El sistema enciende el calefactor a temperatura media de forma automática cuando el sensor DHT11 obtiene lecturas por debajo de 10 ºC.		
Validación	Correcto		
Dependencias	CU-43: Encender calefactor		

Tabla 7.43.- CPV-43: Encender calefactor automático.

Prueba	Apagar calefactor	ID	CPV-44
Descripción	En esta prueba se verifica que el calefactor se apaga de forma automática cuando el sensor DHT11 obtiene lecturas por encima de los 14 ºC.		
Resultado esperado	El sistema debe apagar el calefactor de forma automática si el sensor DHT11 obtiene lecturas por encima de los 14 ºC.		
Resultado obtenido	El sistema apaga el calefactor de forma automática cuando el sensor DHT11 obtiene lecturas por encima de los 14 ºC.		

Validación	Correcto
Dependencias	CU-43: Apagar calefactor

Tabla 7.44.- CPV-44: Apagar calefactor automático.

Prueba	**Encender aire acondicionado**	**ID**	**CPV-45**
Descripción	En esta prueba se verifica que el aire acondicionado se enciende de forma automática con intensidad media cuando el sensor DHT11 obtiene lecturas por encima de los 25 ºC.		
Resultado esperado	El sistema debe encender el aire acondicionado con intensidad media de forma automática si el sensor DHT11 obtiene lecturas por encima de los 25 ºC.		
Resultado obtenido	El sistema enciende el aire acondicionado con intensidad media de forma automática cuando el sensor DHT11 obtiene lecturas por encima de los 25 ºC.		
Validación	Correcto		
Dependencias	CU-45: Encender aire acondicionado		

Tabla 7.45.- CPV-45: Encender aire acondicionado automático.

Prueba	**Apagar aire acondicionado**	**ID**	**CPV-46**
Descripción	En esta prueba se verifica que el aire acondicionado se apaga de forma automática cuando el sensor DHT11 obtiene lecturas inferiores a 21 ºC.		
Resultado esperado	El sistema debe apagar el aire acondicionado de forma automática si el sensor DHT11 obtiene lecturas por debajo de los 21 ºC.		
Resultado obtenido	El sistema apaga el aire acondicionado de forma automática cuando el sensor DHT11 obtiene lecturas por debajo de los 21 ºC.		
Validación	Correcto		
Dependencias	CU-46: Apagar aire acondicionado		

Tabla 7.46.- CPV-46: Apagar aire acondicionado automático.

Prueba	**Activación alarma**	**ID**	**CPV-47**
Descripción	En esta prueba se verifica que la alarma se activa de forma automática cuando el sensor de infrarrojo detecta movimientos.		
Resultado esperado	El sistema debe activar la alarma de forma automática si el sensor infrarrojo detecta movimientos.		

Resultado obtenido	El sistema activa la alarma de forma automática cuando el sensor infrarrojo detecta movimientos.
Validación	Correcto
Dependencias	CU-47: Activación alarma

Tabla 7.47.- CPV-47: Activación alarma automático.

7.1.4.- Subsistema de Acciones Extras

En este subapartado se describe la prueba realizada sobre el Subsistema de Acciones Extras:

Prueba	**Verificación del reconocimiento de voz**	**ID**	**CPV-48**
Descripción	En esta prueba se verifica que los mensajes de voz del usuario se transcriben de forma correcta a través de la API STT.		
Resultado esperado	El sistema debe transcribir de forma correcta los mensajes de voz enviados por el usuario y mostrar el resultado en su chat de Telegram.		
Resultado obtenido	El sistema transcribe de forma correcta los mensajes de voz enviados por el usuario y lo muestra en su chat de Telegram.		
Validación	Correcto		
Dependencias	CU-48: Verificación del reconocimiento de voz		

Tabla 7.48.- CPV-48: Verificación del reconocimiento de voz.

7.2.- Casos de Prueba Inválidos

En este apartado se realizan varias pruebas para verificar que el sistema contempla acciones correctivas ante un uso inadecuado del mismo por parte del usuario.

Prueba	**Formato de fecha incorrecto (día)**	**ID**	**CPI-01**
Descripción	En esta prueba se verifica que el sistema detecta que el formato de fecha introducido por el usuario no es correcto. Por ejemplo, un día mayor a 31.		
Resultado esperado	El sistema debe detectar que el formato de la fecha es incorrecto, reproducir un audio de “formato inválido” y redirigir al usuario al respectivo IVR.		
Resultado obtenido	El sistema reproduce el audio “formato inválido” y redirige al usuario al IVR correspondiente para repetir el proceso.		
Validación	Correcto		

Tabla 7.49.- CPI-01: Formato de fecha incorrecto (día).

Prueba	Formato de fecha incorrecto (mes)	ID	CPI-02
Descripción	En esta prueba se verifica que el sistema detecta que el formato de fecha introducido por el usuario no es correcto. Por ejemplo, se introduce un mes mayor a 12.		
Resultado esperado	El sistema debe detectar que el formato de la fecha es incorrecto reproduciendo un audio de "formato inválido" y redirigir al usuario al IVR correspondiente.		
Resultado obtenido	El sistema reproduce el audio "formato inválido" y redirige al usuario al IVR correspondiente para repetir el proceso.		
Validación	Correcto		

Tabla 7.50.- CPI-02: Formato de fecha incorrecto (mes).

Prueba	Formato de hora incorrecto (hora)	ID	CPI-03
Descripción	En esta prueba se verifica que el sistema detecta que el formato de la hora introducido por el usuario no es correcto. Por ejemplo, se introduce una hora mayor a 23.		
Resultado esperado	El sistema debe detectar que el formato de la hora es incorrecto reproduciendo un audio de "formato inválido" y redirigir al usuario al IVR correspondiente.		
Resultado obtenido	El sistema reproduce el audio "formato inválido" y redirige al usuario al IVR correspondiente para repetir el proceso.		
Validación	Correcto		

Tabla 7.51.- CPI-03: Formato de hora incorrecto (hora).

Prueba	Formato de hora incorrecto (minutos)	ID	CPI-04
Descripción	En esta prueba se verifica que el sistema detecta que el formato de la hora introducido por el usuario no es correcto. Por ejemplo, se introduce unos minutos mayor a 59.		
Resultado esperado	El sistema debe detectar que el formato de la hora es incorrecto reproduciendo un audio de "formato inválido" y redirigir al usuario al IVR correspondiente.		
Resultado obtenido	El sistema reproduce el audio "formato inválido" y redirige al usuario al IVR correspondiente para repetir el proceso.		
Validación	Correcto		

Tabla 7.52.- CPI-04: Formato de hora incorrecto (minutos).

Prueba	Fecha anterior a la actual	ID	CPI-05
Descripción	En esta prueba se verifica la respuesta del sistema cuando el usuario introduce una fecha anterior a la actual.		
Resultado esperado	El sistema debe ejecutar la acción inmediatamente.		
Resultado obtenido	El sistema ejecuta la acción inmediatamente.		
Validación	Correcto		

Tabla 7.53.- CPI-05: Fecha anterior a la actual.

Prueba	Hora anterior a la actual	ID	CPI-06
Descripción	En esta prueba se verifica la respuesta del sistema cuando el usuario introduce una hora anterior a la actual.		
Resultado esperado	El sistema debe ejecutar la acción inmediatamente.		
Resultado obtenido	El sistema ejecuta la acción inmediatamente.		
Validación	Correcto		

Tabla 7.54.- CPI-06: Hora anterior a la actual.

Prueba	Indicar comando de voz desconocido	ID	CPI-07
Descripción	En esta prueba se verifica que el sistema responde con un audio de error cuando el usuario indica un comando de voz desconocido y lo redirige al IVR correspondiente.		
Resultado esperado	El sistema debe reproducir un audio de “comando inválido” y redirigir al usuario al IVR correspondiente para repetir el proceso.		
Resultado obtenido	El sistema reproduce el audio “comando inválido” y redirige al usuario al IVR correspondiente para repetir el proceso.		
Validación	Correcto		

Tabla 7.55.- CPI-07: Indicar comando de voz desconocido.

Prueba	No indicar ningún comando de voz	ID	CPI-08
Descripción	En esta prueba se verifica que se reproduce un audio de error cuando el usuario permanece en silencio y el sistema espera recibir un comando de voz, entonces redirige al usuario al IVR correspondiente.		
Resultado esperado	El sistema debe reproducir un audio de “respuesta vacía” y redirigir al usuario al IVR correspondiente para repetir el proceso.		

Resultado obtenido	El sistema reproduce el audio "respuesta vacía" y redirige al usuario al IVR correspondiente para repetir el proceso.
Validación	Correcto

Tabla 7.56.- CPI-08: No indicar ningún comando de voz.

Prueba	**Activar modo automático cuando ya está activado**	**ID**	**CPI-09**
Descripción	En esta prueba se verifica la respuesta del sistema cuando el usuario activa el modo automático y ya el sistema se encontraba en este modo de funcionamiento.		
Resultado esperado	El sistema debe mantenerse funcionando en modo automático si el usuario activa este modo y ya estaba activado.		
Resultado obtenido	El sistema se mantiene funcionando en modo automático cuando el usuario activa este modo y ya estaba activado.		
Validación	Correcto		

Tabla 7.57.- CPI-09: Activar modo automático cuando ya está activado.

7.3.- Conclusiones Parciales

Una vez finalizada la ejecución del conjunto de pruebas realizadas sobre el sistema se puede decir que este responde según lo esperado tanto en casos de prueba válidos como inválidos. Por tanto, se puede concluir que el sistema responde adecuadamente con las funcionalidades planteadas en el catálogo de requisitos.

8.- CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se recogen las principales conclusiones obtenidas una vez terminado el proyecto y se proponen las líneas futuras de trabajo, las cuales sirven como base para mejorar aspectos que no han sido abordados en el proyecto.

8.1.- Conclusiones

Tras la realización de este Trabajo Fin de Máster se ha llegado a las siguientes conclusiones:

Asterisk es un software libre que proporciona funcionalidades de una central telefónica y es ampliamente usado en entornos empresariales. Sin embargo, debido a su versatilidad y flexibilidad puede usarse en entornos domésticos, permitiendo crear nuevas funcionalidades que van más allá de lo que se puede esperar de una central telefónica convencional. Por tanto, el uso de Asterisk ha sido clave ya que se adapta perfectamente a las necesidades iniciales de este proyecto.

Se ha conseguido desarrollar una plataforma de comunicación integrando diversas tecnologías libres como Asterisk, Apache, Arduino, Telegram, etc. El resultado ha sido un prototipo que traduce las órdenes de voz dadas por el usuario en acciones ejecutadas directamente sobre el hogar domotizado. Además, ante sucesos en el hogar se notifica al usuario por mensajería instantánea o se realizan algunas acciones automáticamente.

A día de hoy existen varias plataformas en el mercado para el control domótico, pero tienen un coste elevado de adquisición. Por tanto, implementar un prototipo usando software libre permite que la solución sea más asequible y de bajo coste.

Se han empleado varios sensores y actuadores para simular un hogar inteligente, de esta forma poder demostrar la viabilidad y eficacia del prototipo propuesto. Además, que se trata de un prototipo extensible y configurable ya que permite integrar nuevas funcionalidades con muy poco esfuerzo.

También, se ha conseguido un sistema funcional y fácil de utilizar ya que se implementa un menú de voz interactiva para guiar al usuario durante la llamada, especialmente pensado para personas mayores o con nulos conocimientos informáticos.

Si bien es cierto que el prototipo que se propone no es del todo flexible debido a que su funcionamiento se basa en comandos de voz predefinidos, este trabajo puede servir como base para la búsqueda de soluciones tecnológicas mejor adaptadas a partir de plataformas de libre acceso. Contar con herramientas que permitan controlar y gestionar los dispositivos del hogar, ayuda a mejorar el estilo de vida de personas con algunas discapacidades o simplemente por comodidad, simplificando tareas cotidianas. Además, se pueden obtener importantes ahorros energéticos y económicos.

8.2.- Líneas Futuras

A continuación, se proponen una serie de líneas de trabajo futuro que pueden aportar a este proyecto varias mejoras:

En primer lugar, se propone configurar la centralita de Asterisk adecuadamente para que permita el acceso al control de los dispositivos del hogar domotizado desde el exterior del entorno doméstico. Esta ampliación es posible técnicamente, pero este proyecto solo se ha ceñido al uso de los dispositivos dentro de la red doméstica.

Además, resulta conveniente integrar en el hogar domotizado una librería SIP que permita realizar llamadas a la centralita como si de un usuario se tratase. Conviene destacar que para la versión de la placa de Arduino utilizada en este proyecto no se ha encontrado ninguna compatible. Este enfoque basado exclusivamente en el protocolo SIP permite prescindir de la entidad correspondiente con el servidor Apache dando lugar a un sistema basado exclusivamente en tecnologías relacionadas con la VoIP.

Igualmente, se plantea realizar un plan de pruebas intensivo de la solución que involucre a usuarios de diferentes edades, con diferentes acentos, etc. Esto permitiría comprobar que el sistema reconoce perfectamente la voz de los usuarios con independencia del tono de voz de los mismos. También, se puede involucrar a usuarios con escasas habilidades digitales con el fin de demostrar que el prototipo propuesto es una solución sencilla y de fácil manejo para cualquier usuario ya que en todo momento se le va guiando durante la llamada a la centralita.

Asimismo, haciendo uso de la API STT de Google *Cloud* que permite hacer traducciones de voz a texto en más de ciento veinte idiomas, se propone extender el servicio para que esté disponible en otros idiomas y no solamente en lenguaje español. Esto permitiría llegar a mayor cantidad de usuarios.

Por otra parte, la citada API STT de Google *Cloud* no es gratuita a partir de los 60 minutos de audios procesados en un mes. Por tanto, se hace necesario buscar una alternativa libre de pago y fácilmente integrable en el sistema, sin que se penalice la calidad de la traducción que proporciona esta API.

Finalmente, es preciso investigar en el uso de motores semánticos que proporcionen mayor flexibilidad durante la petición de órdenes del usuario a la centralita, ya que el sistema actualmente funciona en base a una serie de órdenes predefinidas que la centralita es capaz de procesar, ya que de lo contrario el sistema no funciona adecuadamente.

9.- PRESUPUESTO

En este capítulo se detalla los costes derivados de la implementación del proyecto. En este presupuesto se identifican tres capítulos presupuestarios:

- **Recursos hardware:** Se refiere al coste de todos los equipos, dispositivos y materiales empleados en el proyecto.
- **Recursos software:** Está asociado al coste de las licencias de los programas empleados en el proyecto.
- **Recursos humanos:** Hace referencia al coste derivado de la mano de obra necesaria para el desarrollo e implementación del proyecto.

9.1.- Recursos Hardware

En la Tabla 9.1 se refleja las características y funciones de los ordenadores portátiles, así como de las maquetas comerciales de *Smart Home* que se han empleado:

1.-	**Ordenador Portátil LENOVO**
Función	Funciona como teléfono VoIP (Zoiper). Adicionalmente, es donde se programan las funciones de los sensores y actuadores del hogar inteligente a través del IDE de Arduino.
Procesador	Intel Core i7-9750H CPU @ 2.60GHz
Memoria RAM	16,0 GB
Disco Duro	HDD de 1TB, SSD de 256 GB
Sistema Operativo	Windows 10.0.19043 de 64 bits
Aplicaciones	Zoiper 5 *Free*, IDE Arduino 1.8.13
2.-	**Ordenador Portátil ASUS**
Función	Actúa como centralita telefónica de Asterisk. Además, se instala un servidor Apache y PHP para atender peticiones desde Arduino hacia el usuario.
Procesador	Intel Pentium CPU 2020M @ 2.40GHz
Memoria RAM	4,0 GB
Disco Duro	SSD de 256 GB
Sistema Operativo	Linux distribución Ubuntu 20.04.3 LTS de 64 bits

Aplicaciones	Asterisk 18.2.1, Apache 2.4.41, PHP 7.4.3
3.-	**OSOYOO *Smart Home* IoT compatible con Arduino Mega2560**
Función	Es una maqueta que simula el comportamiento de un hogar domotizado.
Dispositivos	Placa Mega2560 de OSOYOO x1 *Shield* Mega-IoT de OSOYOO x1 Módulo RGB x1 Módulo LED luz blanca x1 Módulo LED luz roja x1 Módulo LED luz verde x1 Módulo LED luz amarilla x1 Micro Servomotor x1 Módulo *Buzzer* Activo x1 Módulo PIR x1 Módulo DHT11 x1 LCD 1602 x1
Materiales	Cable USB tipo B x1 Cable PnP de 4 pines x2 Cable PnP de 3 pines x11 Modelo de casa de madera x1
4.-	**KS0085 Keyestudio *Smart Home* para Arduino**
Función	Complementaria. Se añaden otros dispositivos no presentes en el *Smart Home* de OSOYOO.
Dispositivos	Módulo LED luz blanca x1 Módulo LED luz amarilla x1 Módulo *Buzzer* Pasivo x1 Micro Servomotor x1 Módulo *Fan* Motor x1

Tabla 9.1.- Características y funciones del hardware empleado.

En la Tabla 9.2 se figuran los costes de los recursos hardware:

ID	Descripción	Medición	Cantidad	Precio Unitario (€)	Precio Total (€)
HW1	Ordenador Lenovo	Uds.	1	940,00	940,00
HW2	Ordenador Asus	Uds.	1	400,00	400,00
HW3	*Smart Home* OSOYOO	Uds.	1	82,00	82,00
HW4	*Smart Home* Keyestudio	Uds.	1	73,99	73,99
SUBTOTAL RECURSOS HARDWARE					**1495,99 €**

Tabla 9.2.- Presupuesto recursos hardware.

En la tabla anterior se muestran los costes absolutos de los recursos hardware, sin embargo, en la Tabla 9.3 se aprecia un estimado de la amortización de dichos recursos en este proyecto:

ID	Descripción	Precio Unitario (€)	Tiempo Total (años)	Cuota Mensual (€)	Tiempo de uso (meses)	Amortización (€)
HW1	Ordenador Lenovo	940	5	15,66	7	109,67
HW2	Ordenador Asus	400	5	6,66	7	46,67
HW3	*Smart Home* OSOYOO	82	3	2,28	7	15,94
HW4	*Smart Home* Keyestudio	73.99	3	2,06	7	14,39
SUBTOTAL AMORTIZADO DE RECURSOS HARDWARE						**186,66 €**

Tabla 9.3.- Amortización de los recursos hardware.

9.2.- Recursos Software

En la Tabla 9.4 figuran los costes de los recursos software empleados, la mayoría son de código abierto (por lo que son gratuitos), otros como el sistema operativo Windows 10 y la aplicación Zoiper 5 ofrecen una versión gratuita, dichas versiones son las que se han utilizado en el proyecto.

En el caso de la API STT de Google *Cloud*, esta sí es de pago, pero brinda un periodo gratuito de noventa días si se realiza el registro con una cuenta de correo nueva. En el marco de este proyecto, no se agotó este periodo por lo que el uso de esta API ha resultado gratuito.

Para acceder al listado oficial de precios de la API STT de Google *Cloud* consultar [24].

ID	Descripción	Medición	Cantidad	Precio Unitario (€)	Precio Total (€)
SW1	Windows 10.0.19043	Uds.	1	0	0
SW2	Zoiper 5 *Free*	Uds.	1	0	0
SW3	IDE Arduino 1.8.13	Uds.	1	0	0
SW4	Ubuntu 20.04.3 LTS	Uds.	1	0	0
SW5	Asterisk 18.2.1	Uds.	1	0	0
SW6	Apache 2.4.41	Uds.	1	0	0
SW7	PHP 7.4.3	Uds.	1	0	0
SW8	API STT de Google *Cloud*	Uds.	1	0	0
SUBTOTAL DE RECURSOS SOFTWARE					**0 €**

Tabla 9.4.- Presupuesto recursos software.

9.3.- Recursos Humanos

En la Tabla 9.5 se refleja los costes originados por la mano de obra asociada al análisis, diseño, desarrollo, implementación y pruebas del sistema:

ID	Descripción	Medición	Cantidad	Precio Unitario (€)	Precio Total (€)
HM1	Análisis y diseño del sistema	Horas	240	12	2880,00
HM2	Desarrollo e implementación	Horas	320	12	3840,00
HM3	Pruebas de verificación	Horas	40	12	480,00
SUBTOTAL DE RECURSOS HUMANOS					**7200,00 €**

Tabla 9.5.- Presupuesto recursos humanos.

9.4.- Presupuesto Total

En la Tabla 9.6 se muestra el resumen del presupuesto total del proyecto:

PRESUPUESTO TOTAL DEL PROYECTO	
CONCEPTO	**IMPORTE (€)**
Recursos Hardware	186,66

Recursos Software	0,00
Recursos Humanos	7200,00
Subtotal	**7386.66**
Gastos Generales (12%)	886,40
Beneficio Industrial (7%)	517,07
Subtotal Libre de Impuestos	**8790,13**
IVA (21%)	1845,93
IMPORTE TOTAL	**10636,06**

Tabla 9.6.- Presupuesto total del proyecto.

El presupuesto total asciende a la cantidad de **DIEZ MIL SEISCIENTOS TREINTA Y SEIS EUROS CON SEIS CÉNTIMOS** #10636,06€#.

10.- ANEXOS

En este apartado se incluye todo el material complementario que se considera relevante en este proyecto, pero debido a su extensión no se han incluido en el desarrollo de la Memoria. Primeramente, se presenta el manual del instalador, seguido el manual de usuario y finalmente los códigos fuentes de los ficheros que se han programado para el funcionamiento del sistema.

10.1.- Manual del Instalador

Para desarrollar este proyecto es necesario realizar una serie de instalaciones de software para preparar la infraestructura sobre la cual corre el sistema. Con tal fin, se dispone de dos ordenadores portátiles con el propósito de que las funciones de usuario y servidor estén implementadas en equipos independientes, para que la comunicación entre ellos se realice a través de Internet.

En un ordenador se instala el *softphone* Zoiper para realizar las llamadas a la centralita de Asterisk, además del IDE de Arduino para programar las funciones de los sensores y actuadores del hogar inteligente. En el otro se instala el propio servidor de Asterisk y un servidor Apache que actúa como intermediario de la comunicación entre Arduino y el usuario para realizar las notificaciones de algunos eventos en el hogar.

En la Tabla 10.1 se muestra un resumen de las características de los ordenadores empleados y las aplicaciones instaladas en cada uno de ellos.

Características	**Lenovo**	**Asus**
Procesador	Intel Core i7-9750H CPU @ 2.60GHz 2.59 GHz, 64 bits	Intel Pentium CPU 2020M @ 2.40GHz, 64 bits
RAM	16,0 GB	4,0 GB
Sistema Operativo	Windows 10.0.19043	Ubuntu 20.04.3 LTS
Aplicaciones necesarias	IDE Arduino 1.8.13 Zoiper 5 Free	Asterisk 18.2.1 Apache 2.4.41 PHP 7.4.3

Tabla 10.1.- Especificaciones de los ordenadores empleados.

A continuación, se describe detalladamente los pasos seguidos para la instalación de los programas que han sido necesarios para preparar el entorno de trabajo de este proyecto.

10.1.1.- Instalación de Arduino

Existen dos formas diferentes de utilizar el entorno de desarrollo de Arduino. Una es usarlo en un navegador web y la otra es instalarlo en el ordenador. En el caso de este proyecto, se ha optado por la segunda variante.

El primer paso es ir al sitio web oficial de descargas [34], donde aparece el enlace para descargar la versión 1.8.13 del IDE (última disponible en el momento de la realización del trabajo de fin de máster) para el sistema operativo que se esté usando, en este caso Windows 10 (ver Figura 10.1).

Figura 10.1.- Descarga del IDE Arduino 1.8.13 para Windows.

Se descarga un fichero comprimido en formato *zip* y se descomprime apareciendo una estructura de archivos y subcarpetas que no deben ser modificados. Se clica sobre el archivo ejecutable "*arduino.exe*" para comenzar la instalación.

Se aceptan los términos de la licencia pulsando el botón "*I Agree*", se selecciona los componentes a instalar (marcarlos todos) y se pulsa "*Next*". Seguidamente se selecciona el directorio donde se va a instalar el IDE, se recomienda mantener la ruta por defecto que es "*C:\Program Files (x86)\Arduino*", y luego se pulsa el botón "*Install*". Cuando acaba el proceso, se cierra el instalador y se ejecuta la aplicación desde el acceso directo que se ha creado.

Una vez instalado el software de Arduino, hay que conectar la placa al ordenador con el cable USB, a través del cual esta tarjeta se alimenta automáticamente. En ese momento se debe encender un led verde etiquetado como "ON".

10.1.1.1.- Configuración de la placa y el puerto

Una vez realizada la conexión, el siguiente paso es indicarle a Arduino el tipo de placa que tiene conectada al ordenador y el puerto serie que se ha de utilizar para la transferencia de datos vía USB. Esto se configura en el menú "Herramientas", en la opción "Placa" seleccionar el modelo de la placa que se está utilizando, en esta ocasión Arduino

Mega 2560. En la opción “Puerto” elegir el puerto serie correspondiente. En la Figura 10.2 se observa el resultado de dicha configuración. Esta solo es necesario especificarlo una sola vez mientras no se use otro tipo de placa.

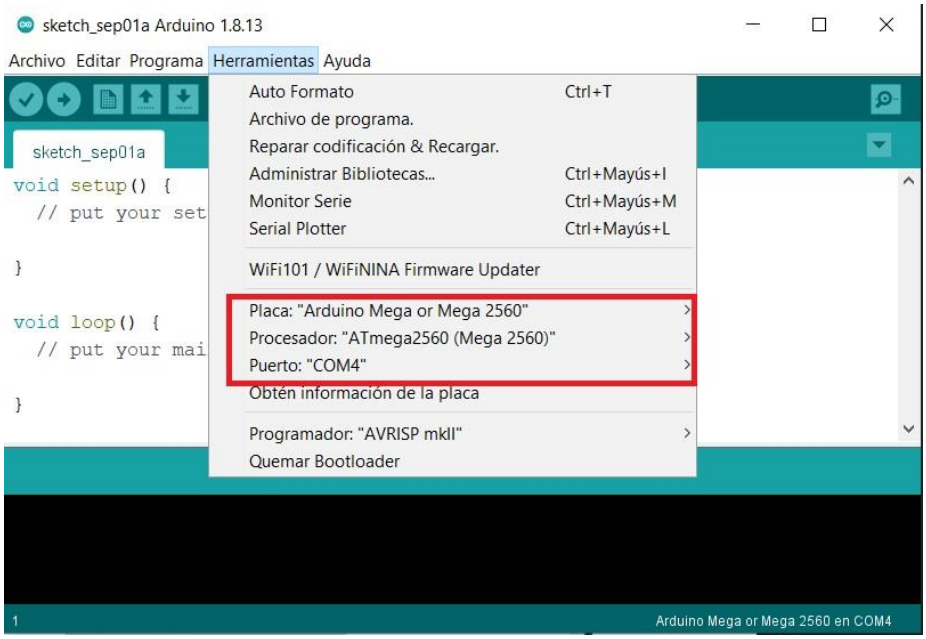


Figura 10.2.- Configuración de la placa y el puerto en el IDE Arduino.

Si en el momento de configurar el puerto esta opción aparece deshabilitada, eso quiere decir que los drivers de la tarjeta utilizada no están disponibles en el computador, y por tanto es necesario instalarlos. Para ello, se accede al “Administrador de dispositivos” de Windows 10. El operativo muestra una lista de todos los dispositivos conectados donde debe aparecer algún dispositivo desconocido. Entonces se hace clic con el botón derecho sobre él y se elige la opción “Actualizar software de controlador”.

Posteriormente se elige la opción “Buscar software de controlador en el equipo” y se navega hasta la carpeta que contiene los drivers, o Se introduce directamente la ruta de los drivers de Arduino. Se presiona “Siguiente” y seguido “Instalar”. Para comprobar que la instalación fue exitosa se debe observar en la lista mostrada por el “Administrador de dispositivos” un nuevo dispositivo dentro de la categoría “Puertos (COM y LPT)” con un nombre que hace referencia a la tarjeta de Arduino y el puerto asignado.

Para finalizar, es conveniente comprobar que todo funciona correctamente ejecutando uno de los programas de prueba que trae Arduino por defecto. Para eso ir al menú “Archivo > Ejemplos > 01.Basics > Blink”. Se compila el código y se carga en el microcontrolador de la placa. Se debe observar que el led parpadea.

10.1.1.2.- Instalación de librerías

Otro aspecto importante es la instalación de librerías. Conviene destacar que existen dos tipos: librerías estándar (desarrolladas por el equipo de Arduino) y librerías no estándar (desarrolladas por terceros).

Si se trata de una librería oficial o estándar, tan solo hay que ir al menú "Programa > Incluir Librería" y seleccionar la librería deseada del menú desplegable que aparece (ver Figura 10.3). Como resultado, se inserta una línea de código en la parte superior del programa con la sintaxis: *#include <nombreLibreria.h>*, esto se debe a que dichas librerías son instaladas junto al IDE de Arduino.

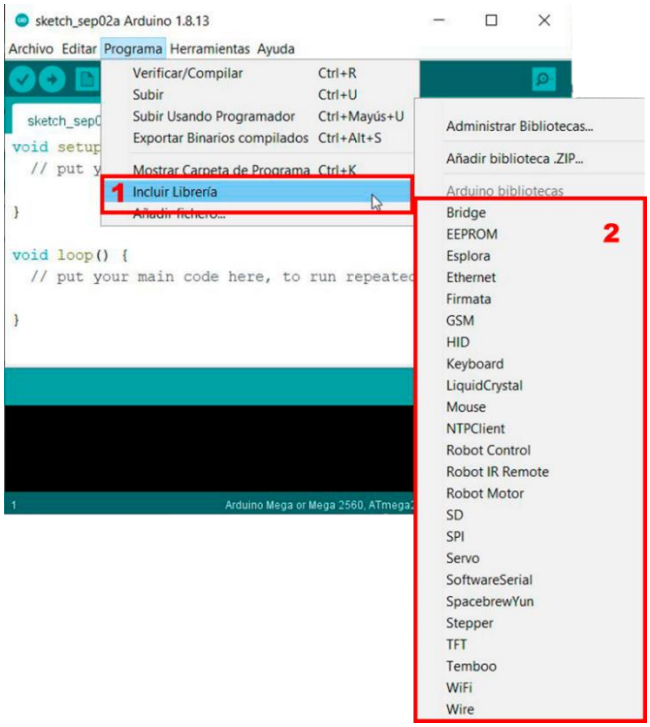


Figura 10.3.- Instalación de librerías estándar en el IDE Arduino.

Por el contrario, si se trata de una librería de terceros éstas no vienen preinstaladas en el IDE de Arduino, luego hay que instalarlas para poder hacer uso de ellas. Existen varias formas de hacerlo, la más sencilla es a través del Gestor de Librerías, yendo al menú "Programa>Incluir Librería>Administrar Bibliotecas" como se muestra en la Figura 10.4. Esto abre una nueva ventana que permite buscar e instalar las librerías que han sido aprobadas y supervisadas por Arduino, pero que no son estándar.

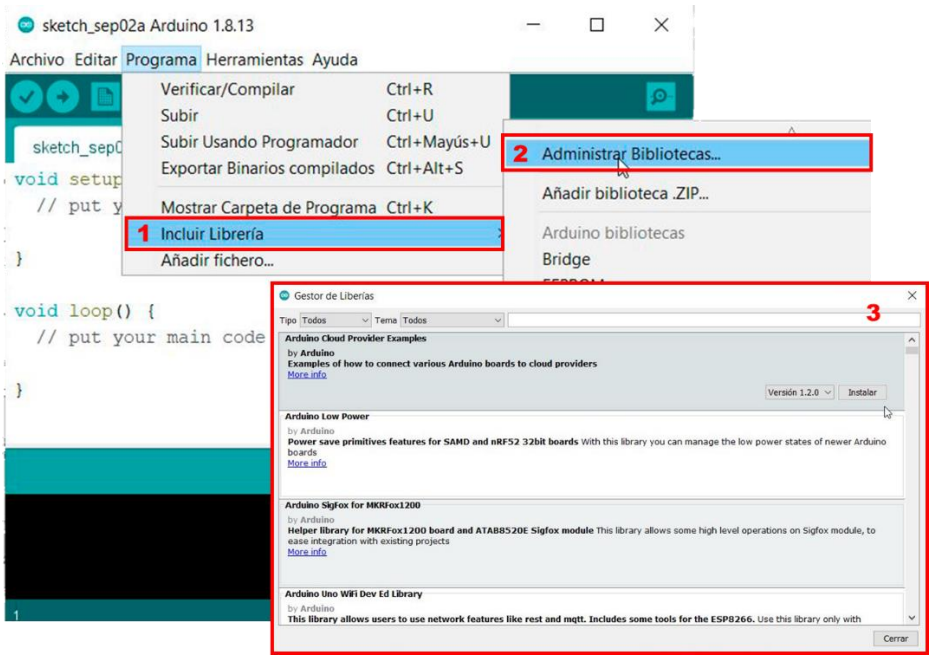


Figura 10.4.- Instalación de librerías no estándar a través del Gestor de Librerías.

En varias ocasiones en el repositorio oficial de Arduino no están todas las librerías. En ese caso lo más habitual es buscar por Internet, por ejemplo, en el repositorio GitHub, y se descarga en formato "*.zip*". Después se accede a la opción del menú "Programa>Incluir Librería>Añadir biblioteca .ZIP" (ver Figura 10.5), se abre una nueva ventana donde se navega por el explorador de Windows hasta llegar a la ruta donde se descargó el fichero y se añade la librería al IDE. El directorio por defecto donde se instalan estas librerías es "C:\Documents\Arduino\libraries".

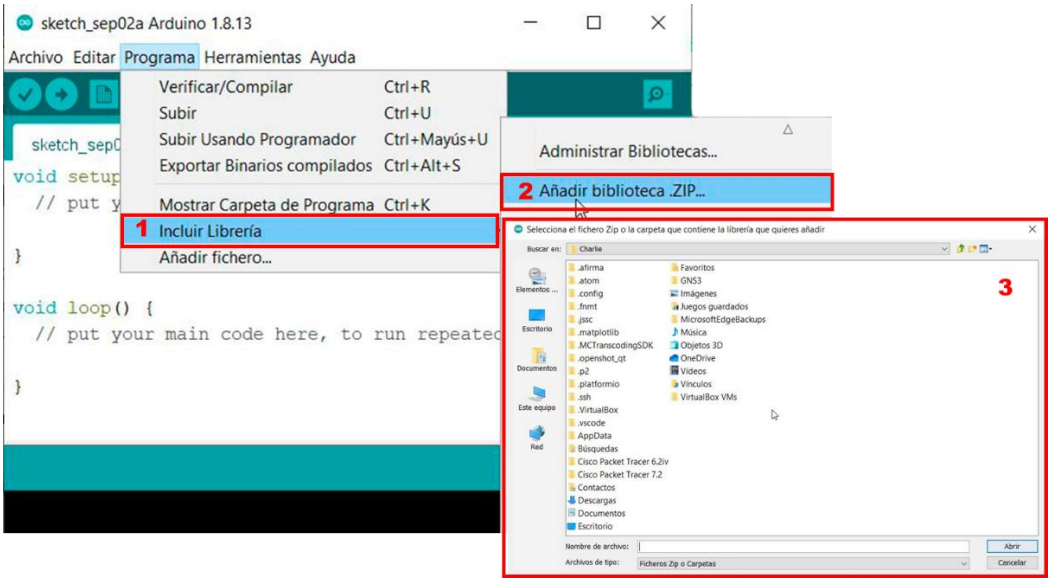


Figura 10.5.- Instalación de librerías no estándar cargando un archivo .zip al IDE.

Siguiendo los pasos explicados anteriormente se instalan las siguientes librerías, las cuales han sido empleadas en la programación del código que controla los dispositivos en el hogar inteligente:

WiFiEsp-master [35]	Servo [38]
RingBuf [36]	LiquidCristal I2C [39]
NoDelay [37]	DHT [40]

10.1.2.- Instalación de Asterisk

Para instalar Asterisk se toma como punto de partida la instalación previa del sistema operativo Linux en su distribución Ubuntu 20.04.3 LTS disponible en [41]. Posteriormente, se usan como guía los enlaces [42] [43] para comenzar el proceso de instalación de Asterisk 18.2.1. Cabe mencionar que el ordenador debe estar provisto de una conexión a Internet.

Primeramente, es recomendable iniciar sesión como usuario '*sudo'* con privilegios o iniciar sesión con el súper-usuario '*root'* para evitar problemas de permisos en los directorios y archivos. Asimismo, es conveniente iniciar la instalación con la actualización del sistema Ubuntu, si la actualización resulta satisfactoria se reinicia el sistema:

```
$ sudo su
# apt update && apt –y upgrade
# systemctl reboot
```

Después se instalan todas las dependencias que necesita Asterisk en Ubuntu 20.04:

```
# apt update
# add-apt-repository universe
# apt -y install git curl wget libnewt-dev libssl-dev libncurses5-dev subversion libsqlite3-dev build-essential libjansson-dev libxml2-dev uuid-dev
```

Dado que Asterisk tiene una estructura modular, agregar funciones adicionales después de la instalación no supone ningún problema. Por eso, en estos pasos se omite la instalación de bibliotecas como Dahdi y Libpri. Si luego se necesitan estas bibliotecas se pueden instalar con posterioridad.

Se descarga la fuente de Asterisk en el directorio /usr /src, que es la ubicación común para colocar los archivos de código fuente y se descomprime:

```
# cd /usr/src/
# wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-18-current.tar.gz
# tar xvf asterisk-18-current.tar.gz
```

Luego se accede al directorio de Asterisk recién creado para descargar las fuentes MP3 que se requieren para construir el módulo MP3 y así poder usar archivos con dicha extensión en Asterisk. Seguido usar el *script install_prereq* para resolver todas las dependencias en el sistema Ubuntu:

```
# cd asterisk-18.*/
# contrib/scripts/get_mp3_source.sh
# contrib/scripts/install_prereq install
```

Al finalizar, si la instalación tuvo éxito, se muestra un mensaje por pantalla como el mostrado en la Figura 10.6.

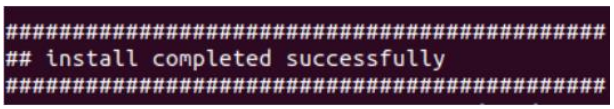


Figura 10.6.- Mensaje exitoso de la instalación de dependencias de Asterisk.

Todo estaría listo para comenzar a instalar Asterisk desde la fuente que se descargó anteriormente. Para ello, se ejecuta el *script* "*configure*", que realiza una serie de comprobaciones para asegurarse de que todas las dependencias del sistema están presentes:

```
# ./configure
```

Si todo estuvo correcto, se observa un mensaje de salida como el mostrado en la Figura 10.7:

Figura 10.7.- Mensaje de salida después de ejecutar el *script* "*configure*".

El siguiente paso es seleccionar los módulos que se quieren compilar e instalar, escribiendo:

```
# make menuselect
```

Como ya se ha descargado los archivos fuentes MP3, ahora se crea el módulo seleccionando en el menú la opción "*format_mp3*". Adicionalmente se pueden elegir otras opciones de interés. Para terminar y guardar los cambios se hace clic en "*Save & Exit*".

Finalmente, se instala Asterisk, sus módulos y sus archivos de configuración introduciendo lo siguientes comandos:

```
# make
# make install
# make samples
# make config
# ldconfig
```

Asterisk por defecto se ejecuta como usuario *root*. Por motivos de seguridad se crea un nuevo usuario llamado "asterisk" con el siguiente comando:

```
# adduser --system --group --home /var/lib/asterisk --no-create-home --gecos "Asterisk PBX" asterisk
```

Para que Asterisk se ejecute como usuario "asterisk" se abre el archivo "/etc/default/asterisk" con el siguiente comando y se descomentan las dos líneas AST_USER="asterisk" y AST_GROUP="asterisk":

```
# nano /etc/default/asterisk
```

Después se agrega el usuario "asterisk" a los grupos *dialout* y de audio. También se cambia la propiedad y los permisos de todos los archivos y directorios de Asterisk para que el usuario creado pueda acceder a esos archivos:

```
# usermod -a -G dialout,audio asterisk
# chown -R asterisk: /var/{lib,log,run,spool}/asterisk /usr/lib/asterisk /etc/asterisk
# chmod -R 750 /var/{lib,log,run,spool}/asterisk /usr/lib/asterisk /etc/asterisk
```

Se inicializa y habilita el servicio de Asterisk para que se inicie siempre que este arranque:

```
# systemctl start asterisk
# systemctl enable asterisk
```

Si todo está correcto, será posible conectarse a la consola CLI (*Command-Line Interface*) de Asterisk. Esta herramienta es muy útil para los administradores, ya que permite obtener información de componentes, configuraciones, aplicaciones, funciones, lo que ayuda a depurar errores de ejecución.

```
$ sudo asterisk –vvvr
```

En la Figura 10.8 se puede ver una captura donde se comprueba que el servicio de Asterisk está corriendo.

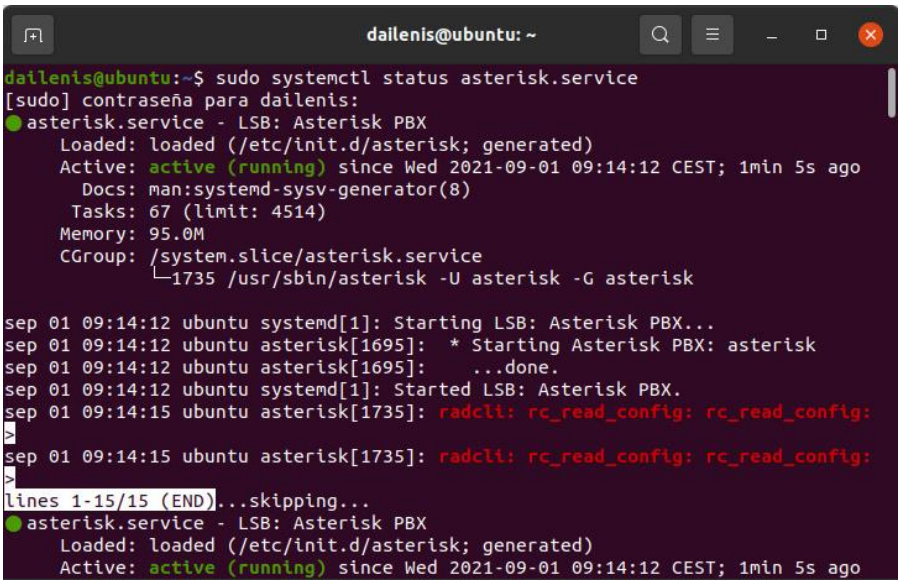


Figura 10.8.- Captura del estado activo de Asterisk.

Para una mayor seguridad, siempre es interesante activar el *firewall* para permitir solo el tráfico deseado. Para ello, se pueden abrir los puertos por defecto tanto de HTTP (80) como de SIP (5060):

```
$ sudo ufw allow http
$ sudo ufw allow 5060/udp
```

10.1.3.- Instalación de Zoiper

Lo primero es descargar el instalador de Zoiper 5 desde la web oficial [44], seleccionando la versión *Free* para Windows (ver Figura 10.9). Se descarga un ejecutable desde el que se lanza el proceso de instalación el cual es muy intuitivo.

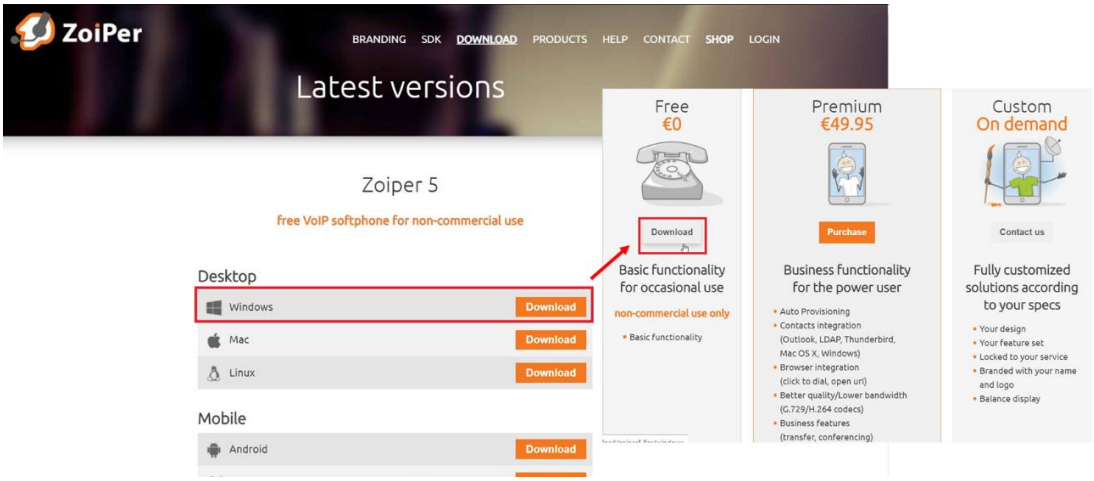


Figura 10.9.- Descarga del instalador Zoiper 5 *Free* para Windows.

Inicialmente, se aceptan los términos de la licencia, se habilita la opción de añadir un acceso directo al escritorio, se indica la ruta de instalación "C:\Program Files (x86)\Zoiper5" y se especifica el nombre de la carpeta en el menú de inicio. Hasta este punto de la instalación siempre se clica en "*Next*" manteniendo los valores por defecto del instalador. A continuación, hay que seleccionar la versión de la arquitectura (en este caso de 64 bits), después elegir si la aplicación estará disponible para el usuario actual o todos los usuarios (se elige "*all users*") y entonces se instalan todos los paquetes de la aplicación. Cuando haya finalizado, se clica en el botón "*Finish*" para cerrar el instalador.

10.1.4.- Instalación de Apache y PHP

A continuación, se menciona los pasos seguidos para instalar un servidor Apache y el procesador de código PHP. Cabe mencionar que estas aplicaciones solamente se utilizan con la finalidad de establecer la comunicación de Arduino hacia el usuario, usando como intermediario el servidor Apache. Por esta razón, no es explotado todo el potencial que brinda este software, sino que se utiliza su configuración básica para realizar únicamente la función de atender la petición de Arduino invocando un *script* dentro del servidor para dar aviso al usuario. Para eso, no es necesario ninguna interfaz web para el usuario.

Tomando como guía los enlaces [45] [46], lo primero[es iniciar sesión como usuario con privilegios, luego se instalan las actualizaciones de versiones y parches del sistema y seguidamente se instala el servidor Apache con los siguientes comandos:

```
$ sudo apt update; sudo apt upgrade
$ sudo apt install –y apache2 apache2-utils
```

Antes de probar Apache, es necesario ajustar la configuración del *firewall* para permitir tráfico HTTP si aún no se ha configurado, para ello:

```
$ sudo ufw status
$ sudo ufw allow 'Apache'
```

Una vez que la instalación se complete es conveniente verificar la versión que se ha instalado y comprobar que el servidor está en ejecución (ver Figura 10.10).

```
$ sudo apache2 –v
$ sudo systemctl status apache2
```

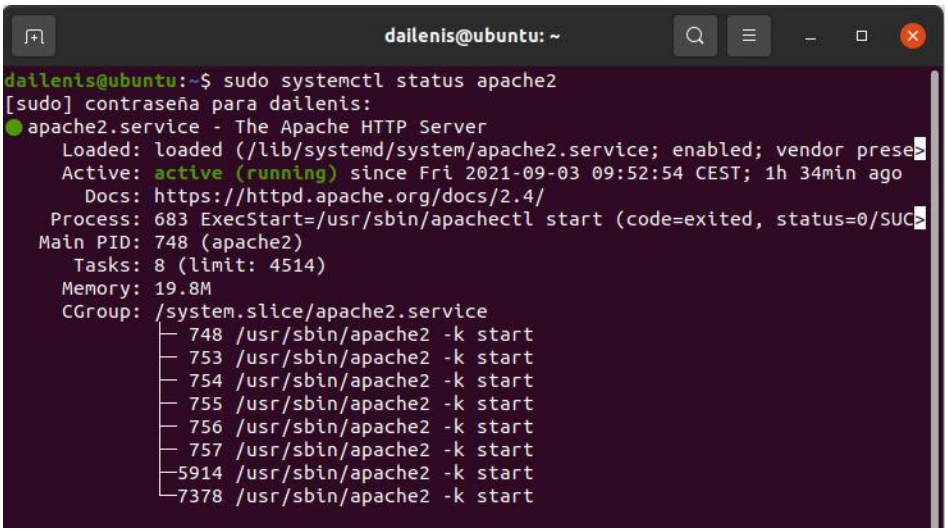


Figura 10.10.- Captura del servidor Apache en ejecución.

No obstante, la mejor forma de comprobar que el servidor funciona correctamente es solicitarle un recurso web. Para acceder a la página predeterminada de Apache es tan sencillo como realizar una petición HTTP desde un navegador a la dirección IP donde se encuentra el servidor (http: //192.168.1.94) y se observa lo que aparece en la Figura 10.11.

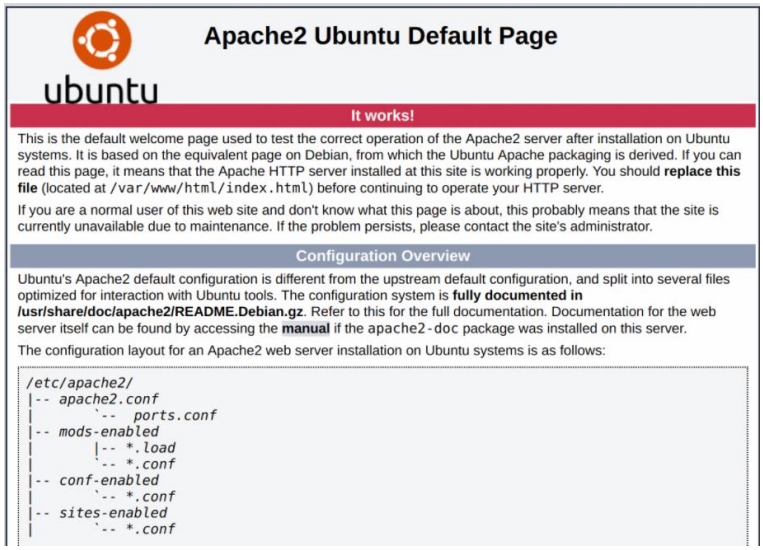


Figura 10.11.- Captura de la página web predeterminada de Apache.

Para establecer el usuario apache como propietario de la raíz web se utiliza el siguiente comando:

```
$ sudo chown www-data: www-data /var/www/html/ -R
```

Por otra parte, se instala PHP que se encarga de procesar el código en dicho lenguaje. Los paquetes PHP básicos se instalan automáticamente como dependencias cuando se ejecuta el siguiente comando.

```
$ sudo apt install php libapache2-mod-php
```

Para probar el funcionamiento, se crea un archivo *info.php* en el directorio web de Apache con las siguientes líneas:

```
<?php
phpinfo();
?>
```

Luego se abre el navegador web y se realiza una petición HTTP a la dirección IP del servidor seguido del nombre del fichero recién creado (http://192.168.1.94/info.php). Debe aparecer una página como la mostrada en la Figura 10.12.

PHP Version 7.4.3

System	Linux ubuntu 5.11.0-27-generic #29~20.04.1-Ubuntu SMP Wed Aug 11 15:58:17 UTC 2021 x86_64
Build Date	Jul 5 2021 15:13:35
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-ffi.ini, /etc/php/7.4/apache2/conf.d/20-fileinfo.ini, /etc/php/7.4/apache2/conf.d/20-ftp.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-json.ini, /etc/php/7.4/apache2/conf.d/20-mysqli.ini, /etc/php/7.4/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.4/apache2/conf.d/20-phar.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvsem.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,NTS
PHP Extension Build	API20190902,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar

Figura 10.12.- Captura de la página de información de PHP.

Tras comprobar la correcta instalación de PHP es recomendable borrar este archivo ya que proporciona información confidencial sobre el entorno PHP y el servidor en Ubuntu.

10.1.5.- Integración de Telegram

El primer paso es registrarse en la aplicación si aún no se dispone de una cuenta. Luego instalar en un dispositivo electrónico la aplicación desde la *Play Store* o vía web desde el siguiente enlace [47].

Una vez hechos esos pasos preliminares, para integrar el servicio de mensajería instantánea de Telegram con Asterisk se hace lo siguiente:

1. Abrir @BotFatther: se introduce en el buscador de Telegram el nombre @botfatther que es el robot de Telegram que permite crear nuevos bots. Cuando esté abierto escribir el comando */start*. Esto muestra las diferentes acciones que se pueden hacer con BotFather.
2. Crear nuevo *bot*: se pincha la opción */newbot* o se escribe directamente ese comando. Luego se proporciona la información solicitada (ver Figura 10.13). Se puede observar que da varios errores en el nombre de usuario si este ya existiese. Finalmente, la API asigna un *token* de acceso.

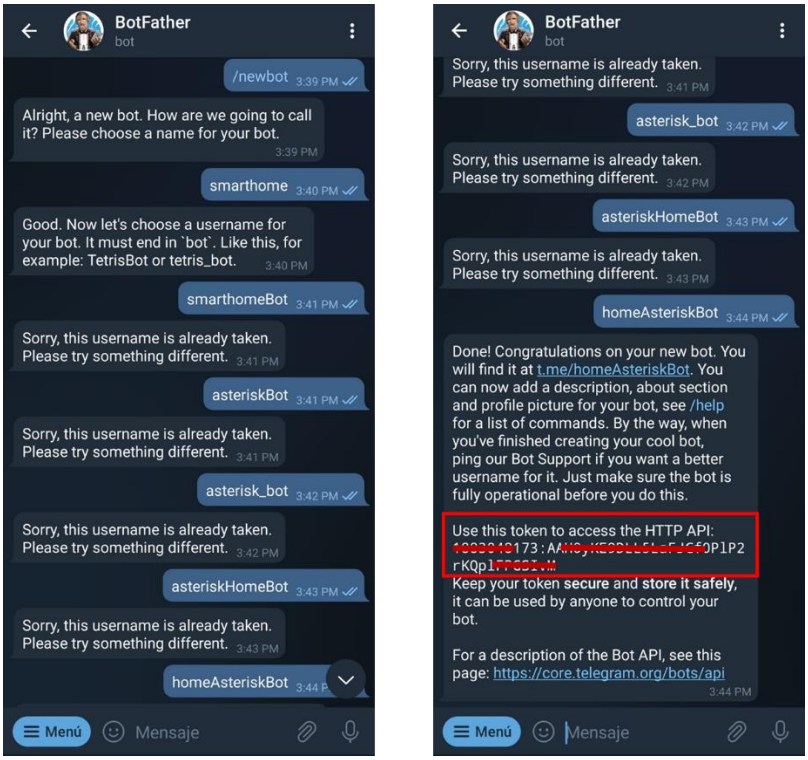


Figura 10.13.- Pasos para la creación del nuevo *bot*.

3. Añadir el *bot* creado a un grupo o canal: se abre la ventana del *bot*, se hace clic encima del nombre del *bot* y se pulsan los tres puntos de la parte superior derecha. Entonces, se selecciona la "Añadir a un grupo", se escoge el grupo deseado y se pulsa "Añadir". Ahora el *bot* forma parte del grupo y puede enviar mensajes como un usuario normal.

4. Obtener identificador del grupo: Se añade al mismo grupo a @GroupIdbot siguiendo la misma secuencia del paso 3. Se escribe el comando */start* o */Id*, el *bot* contesta con el ID (ver Figura 10.14).

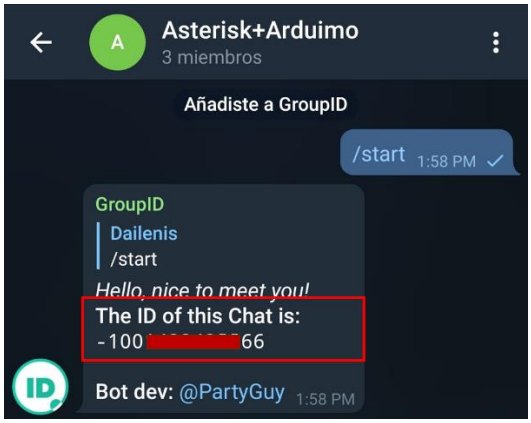


Figura 10.14.- Obtención del ID del grupo.

5. Enviar mensajes desde el *bot* creado: Siguiendo el formato mostrado en la Figura 10.15, se procede a enviar un mensaje de prueba para comprobar el correcto funcionamiento.

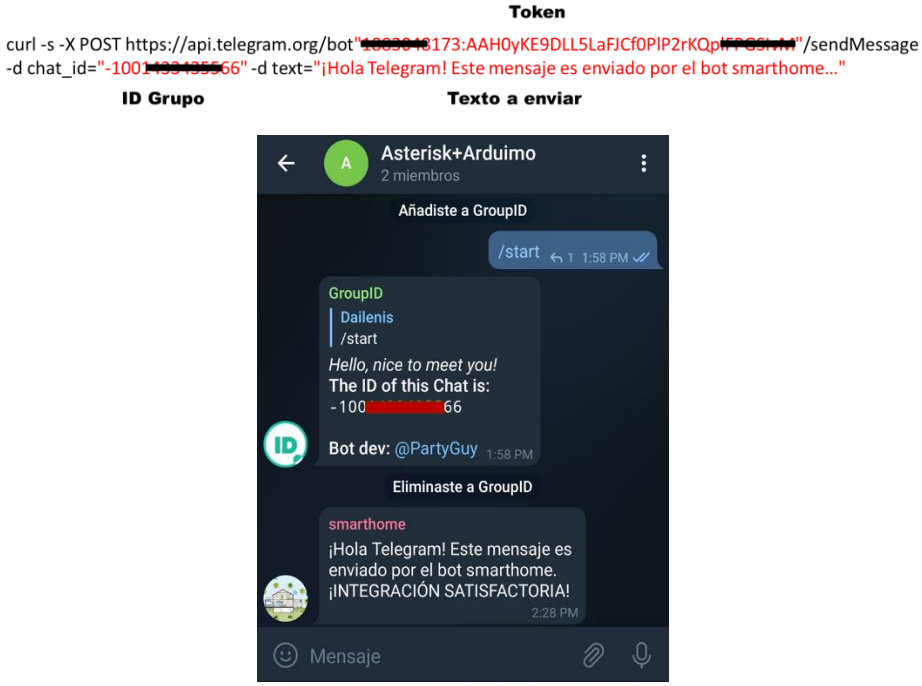


Figura 10.15.- Formato del mensaje y recepción en el chat de Telegram.

10.1.6.- Integración de la API STT

Para realizar el reconocimiento de voz automático, es importante para el desarrollo de este proyecto que en la API utilizada esté incluido el idioma español y que pueda

integrarse con Asterisk. Teniendo en cuenta estos dos factores, se ha decidido emplear la API STT de Google *Cloud*.

Para integrar correctamente este servicio se ha usado el *script* creado por Zafiris, el cual puede descargarse en [33]. Luego se mueve el fichero "speech-recog.agi" a la ruta donde se almacenan los AGI de Asterisk, por lo general /var/lib/asterisk/agi-bin. Para verificar la ubicación, se puede consultar en el fichero /etc/asterisk/asterisk.conf.

También se instalan las siguientes dependencias con el comando "*apt-get -y install paquete_a_instalar"*, ya que el *script* está escrito en lenguaje Perl.

- Perl: el lenguaje de programación.
- perl-libwww: la biblioteca de World-Wide Web para Perl.
- perl-libjson: módulo para manipular datos con formato JSON.
- IO-Socket-SSL: módulo que implementa una interfaz para sockets SSL.
- flac: códec de audio gratuito sin pérdidas.

Seguidamente, se modifica el fichero *extensions.conf*, donde se crea el contexto en el cual se invoca este *script*. A continuación, un fragmento del *dialplan*:

```
[google_stt]
extens => s,1,agi(speech-recog.agi,es,2,#)
same => n,Verbose(1,Lo que dijiste fue: ${utterance})
same => n,Verbose(1,La probabilidad de ser correcto es: ${confidence})
```

Como se puede observar en la segunda línea, se le pasan cuatro parámetros a la aplicación AGI: el nombre del *script* de Zafiris, el idioma español, 2 segundos de *timeout*, y la tecla de interrupción #. Luego este *script* devuelve dos variables de canal que pueden usarse en el *dialplan* de Asterisk. La primera es "*utterance*" que almacena la cadena de texto transcrito. La segunda es "*confidence*" que almacena un valor entre 0 y 1 que indica la probabilidad de un reconocimiento correcto.

Hecho todo lo anterior, aún el *script* no está listo para usarse, ya que es necesario generar una *key* de la API en Google *Cloud* y activar la facturación. Una vez obtenida la clave se inserta en el fichero "speech-recog.agi" en el parámetro (my $key = "__" ;).

A continuación, una breve explicación de los pasos seguidos para obtener la clave.

1. Disponer de una cuenta de correo electrónico.
2. Ir a la consola de desarrolladores de Google *Cloud* y crear un nuevo proyecto asignando un nombre. Una vez dentro del proyecto se clica en la pestaña "Habilitar API y Servicios", se busca dentro de la biblioteca "*Cloud Speech-to-Text* API" y se hace clic en la opción "Habilitar" tal como se muestra en la Figura 10.16.

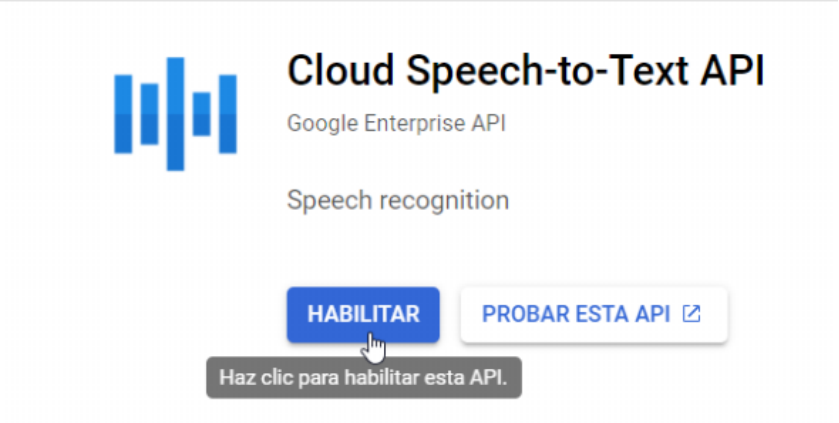


Figura 10.16.- Habilitación de la API STT de Google *Cloud*.

3. Después de habilitar la API, es necesario activar un perfil de facturación válido. En este punto es necesario introducir la información de pago, aunque se esté usando el periodo gratuito. Se omiten estos pasos por motivos de privacidad.
4. El siguiente paso es acceder a la pestaña "Credenciales" para crearlas según como se muestra en el Figura 10.17.

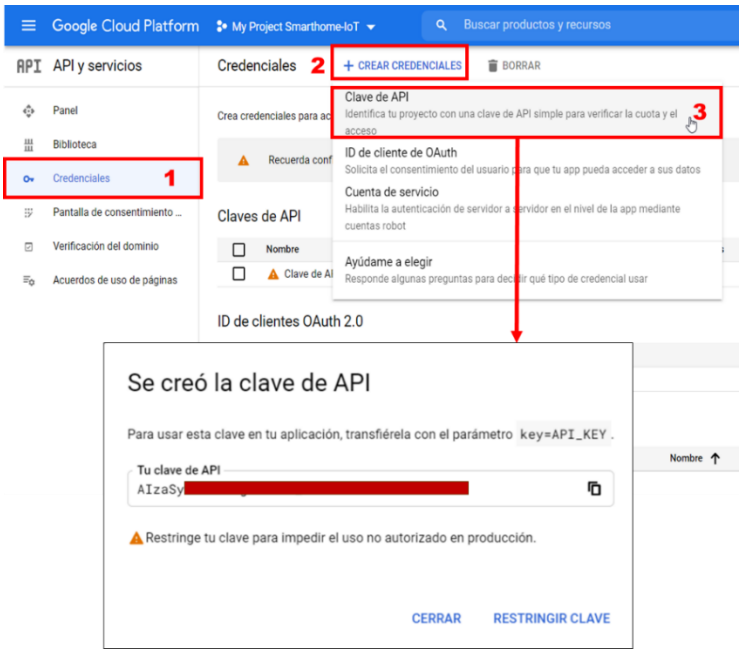


Figura 10.17.- Creación de las credenciales.

5. Por último, se edita el fichero "speech-recog.agi" y se añade la clave recién creada a la variable $key, tal como muestra la Figura 10.18.

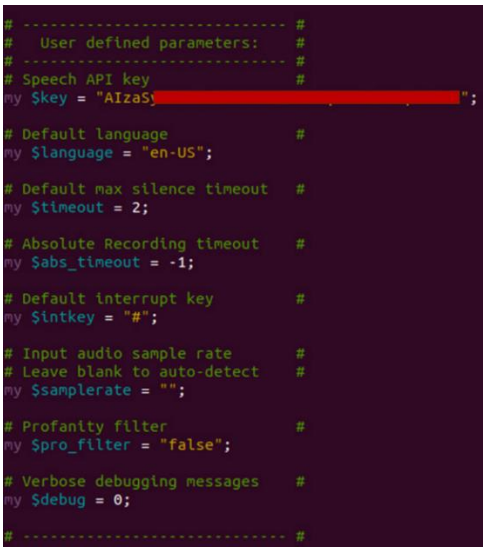


Figura 10.18.- Integración de la clave de Google *Cloud* en el *script* de Zafiris.

10.2.- Manual de Usuario

En este documento se proporciona toda la información necesaria para que el usuario pueda utilizar la herramienta propuesta. Primeramente, se explica los pasos para la configuración de su cuenta en un ordenador o teléfono móvil según las preferencias del usuario. Posteriormente, se mencionan las cuestiones que debe conocer el usuario para un correcto uso y funcionamiento de la herramienta.

10.2.1.- Configuración de la cuenta SIP

Para configurar la cuenta de usuario se ejecuta el acceso directo creado en el escritorio y aparece una interfaz como la que se muestra en la Figura 10.19, donde hace clic en "*Continue as a Free User*".

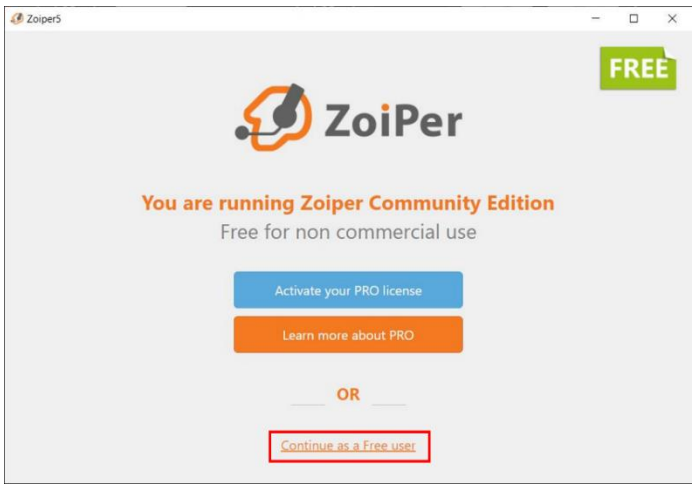


Figura 10.19.- Configuración de Zoiper 5. Paso 1.

El siguiente paso es proporcionar la información que se muestra en la Figura 10.20, relativa al nombre de usuario y la contraseña asociada a la cuenta SIP que debe estar configurada previamente en el archivo *pjsip.conf* de la centralita.

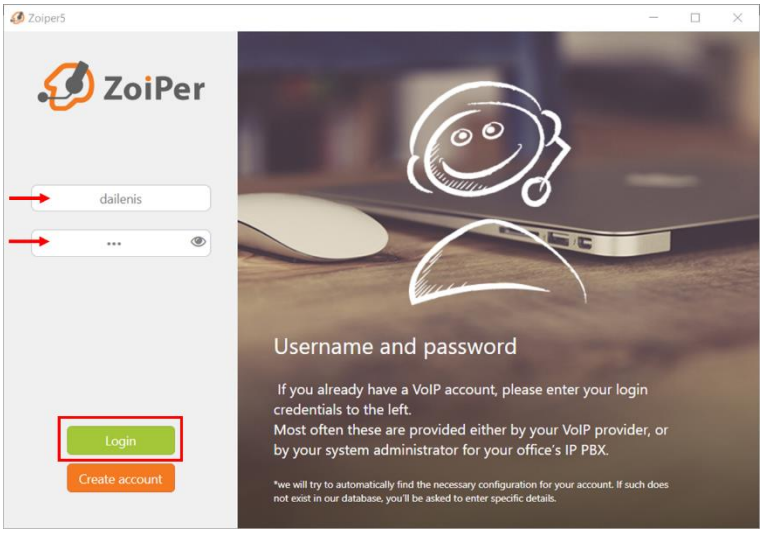


Figura 10.20.- Configuración de Zoiper 5. Paso 2.

Posteriormente se presiona "*Next*", confirmando la dirección IP de la centralita de Asterisk tal como se observa en la Figura 10.21.

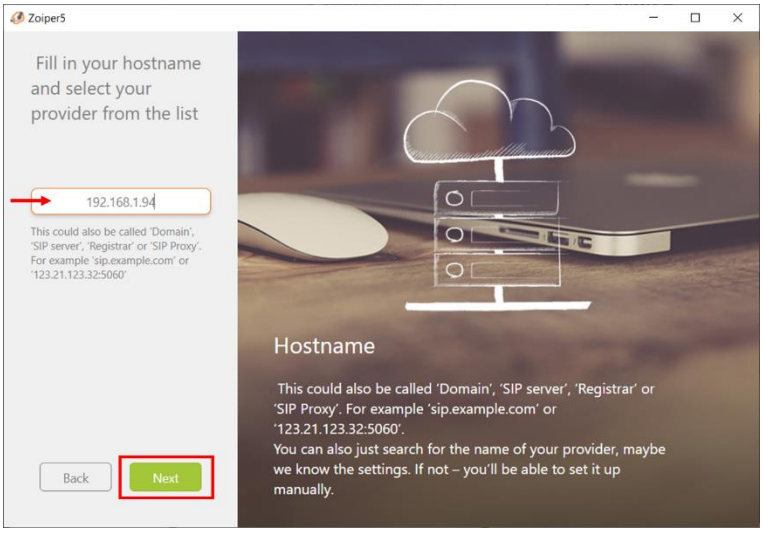


Figura 10.21.- Configuración de Zoiper 5. Paso 3.

En la siguiente pantalla no es necesario configurar nada más, solo presionar "*Skip*" como se ve en la Figura 10.22.

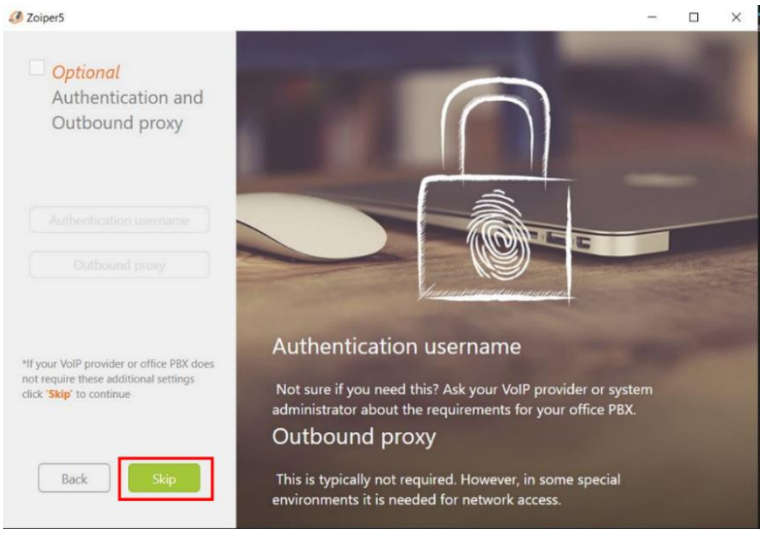


Figura 10.22.- Configuración de Zoiper 5. Paso 4.

El paso siguiente es que el *softphone* identifique a través de qué protocolo se va a conectar a la centralita. Una vez que reconozca el protocolo se presiona "*Next*" tal como se muestra en la Figura 10.23.

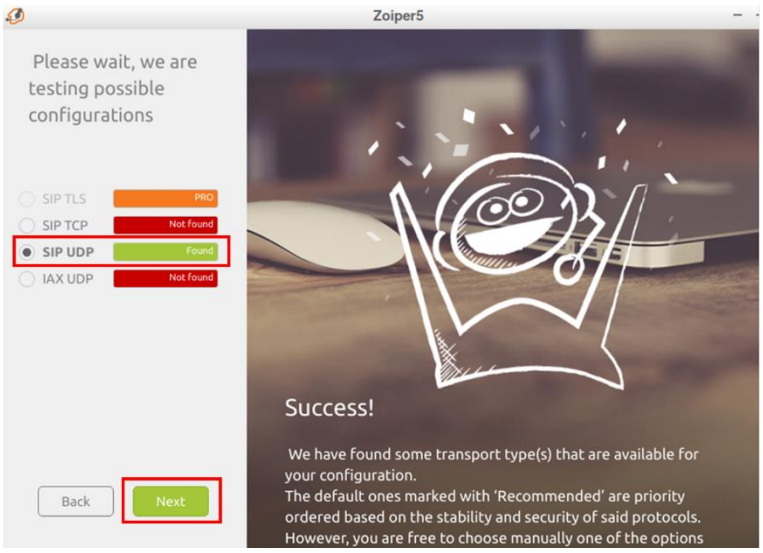


Figura 10.23.- Configuración de Zoiper 5. Paso 5.

Llegado a este punto, el *softphone* ya está listo para realizar y recibir llamadas, mostrando una interfaz de usuario como la que se aprecia en la Figura 10.24 La palomita verde indica que la extensión está registrada correctamente en la centralita. En la rueda dentada se configuran las preferencias, ajustes avanzados, entre otros aspectos, mientras que los puntos señalados permiten desplegar el teclado numérico del *smartphone*.

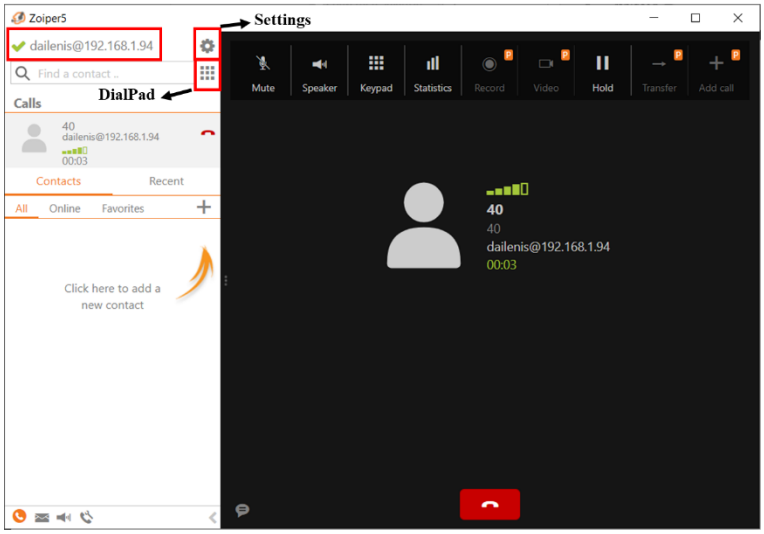


Figura 10.24.- Interfaz gráfica de Zoiper 5.

10.2.2.- Instrucciones para el Usuario

Una vez esté operativo el *softphone*, el usuario tiene que marcar la **extensión 40** para comenzar a interactuar con el hogar inteligente.

Se escucha un audio pregrabado de bienvenida que explica las dos posibles opciones. El usuario tiene que responder "**ahora**" si desea ejecutar una orden inmediata o "**después**" si desea planificar una acción futura. En caso de que la herramienta no entienda lo dicho por el usuario, vuelve a reproducir el audio de inicio.

El siguiente audio pregrabado le pide al usuario que diga la acción que desea realizar. Si la acción no coincide con ninguna de las predefinidas por el sistema, se reproduce nuevamente el audio pidiendo que diga la opción que desea realizar. El sistema espera escuchar una de las siguientes frases:

1	"**Encender luz**"	2	"**Apagar luz**"
3	"**Encender luces navideñas**"	4	"**Apagar luces navideñas**"
5	"**Encender calefactor en uno/ dos / tres**"[2]	6	"**Apagar calefactor**"
7	"**Encender aire en uno/ dos / tres**"[3]	8	"**Apagar aire**"
9	"**Encender aspersor**"	10	"**Apagar aspersor**"
11	"**Abrir puerta**"	12	"**Cerrar puerta**"
13	"**Enviar mensaje**"	14	"**Apagar panel**"

[2] Corresponde con las temperaturas baja, media y alta del calefactor.
[3] Corresponde con las intensidades baja, media y alta del aire acondicionado.

15 "**Activar alarma**" 16 "**Desactivar alarma**"

17 "**Apagar todo**"

Si la acción ordenada es inmediata, se escucha otro audio que solicita al usuario que diga "**uno**" para continuar navegando por el menú o "**dos**" para finalizar la llamada. Si, por el contrario, la acción ordenada es para planificarla a futuro, entonces el usuario tiene que indicar la fecha en formato largo (por ejemplo, "**16 de junio de 2022**") y la hora (por ejemplo, "**18:30**"). Una vez ya se haya definido la fecha y hora de la acción, se solicita confirmar "**uno**" o "**dos**" para continuar o finalizar la llamada respectivamente.

En la Figura 10.25 se representa un diagrama de la secuencia que sigue la llamada del usuario a la centralita.

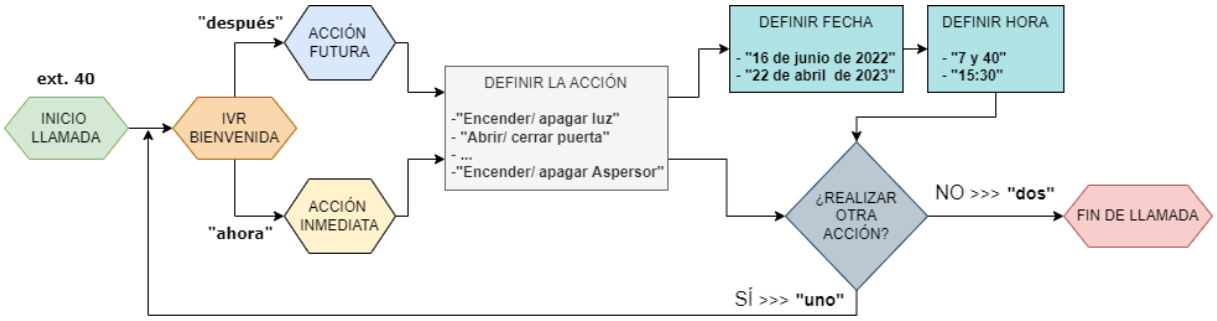


Figura 10.25.- Diagrama de secuencia del IVR configurado.

A continuación, se resume de manera concisa y para mayor claridad la secuencia correcta de comandos para cada escenario de uso.

Acciones Inmediatas:

Encender luz:	"ahora" → "encender luz" → "dos"
Apagar luz:	"ahora" → "apagar luz" → "dos"
Encender luces Navidad:	"ahora" → "encender luces navideñas" → "dos"
Apagar luces Navidad:	"ahora" → "apagar luces navideñas" → "dos"
Encender calefactor temperatura baja:	"ahora" → "encender calefactor en uno" → "dos"
Encender calefactor temperatura media:	"ahora" → "encender calefactor en dos" → "dos"
Encender calefactor temperatura alta:	"ahora" → "encender calefactor en tres" → "dos"
Apagar calefactor:	"ahora" → "apagar calefactor" → "dos"
Encender aire intensidad baja:	"ahora" → "encender aire en uno" → "dos"
Encender aire intensidad media:	"ahora" → "encender aire en dos" → "dos"
Encender aire intensidad alta:	"ahora" → "encender aire en tres" → "dos"
Apagar aire:	"ahora" → "apagar aire" → "dos"

Encender aspersor:	"ahora" → "encender aspersor" → "dos"
Apagar aspersor:	"ahora" → "apagar aspersor" → "dos"
Abrir puerta:	"ahora" → "abrir puerta" → "dos"
Cerrar puerta:	"ahora" → "cerrar puerta" → "dos"
Enviar mensaje:	"ahora" → "enviar mensaje" → "[Grabar]" → "dos"
Apagar panel electrónico:	"ahora" → "apagar panel" → "dos"
Activar alarma:	"ahora" → "activar alarma" → "dos"
Desactivar alarma:	"ahora" → "desactivar alarma" → "dos"
Apagar todo:	"ahora" → "apagar todo" → "dos"

Acciones Futuras:

Encender luz:	"después" → "encender luz" → "[fecha]" → "[hora]" → "dos"
Apagar luz:	"después" → "apagar luz" → "[fecha]" → "[hora]" → "dos"
Encender luces Navidad:	"después" → "encender luces navideñas" → "[fecha]" → "[hora]" → "dos"
Apagar luces Navidad:	"después" → "apagar luces navideñas" → "[fecha]" → "[hora]" → "dos"
Encender calefactor temperatura baja:	"después" → "encender calefactor en uno" → "[fecha]" → "[hora]" → "dos"
Encender calefactor temperatura media:	"después" → "encender calefactor en dos" → "[fecha]" → "[hora]" → "dos"
Encender calefactor temperatura alta:	"después" → "encender calefactor en tres" → "[fecha]" → "[hora]" → "dos"
Apagar calefactor:	"después" → "apagar calefactor" → "[fecha]" → "[hora]" → "dos"
Encender aire intensidad baja:	"después" → "encender aire en uno" → "[fecha]" → "[hora]" → "dos"
Encender aire intensidad media:	"después" → "encender aire en dos" → "[fecha]" → "[hora]" → "dos"
Encender aire intensidad alta:	"después" → "encender aire en tres" → "[fecha]" → "[hora]" → "dos"
Apagar aire:	"después" → "apagar aire" → "[fecha]" → "[hora]" → "dos"
Encender aspersor:	"después" → "encender aspersor" → "[fecha]" → "[hora]" → "dos"
Apagar aspersor:	"después" → "apagar aspersor" → "[fecha]" → "[hora]" → "dos"
Abrir puerta:	"después" → "abrir puerta" → "[fecha]" → "[hora]" → "dos"
Cerrar puerta:	"después" → "cerrar puerta" → "[fecha]" → "[hora]" → "dos"
Enviar mensaje:	"después" → "enviar mensaje" → "[Grabar]" → "[fecha]" → "[hora]" → "dos"
Apagar panel electrónico:	"después" → "apagar panel" → "[fecha]" → "[hora]" → "dos"
Activar alarma:	"después" → "activar alarma" → "[fecha]" → "[hora]" → "dos"

Desactivar alarma:	"ahora" → "desactivar alarma" → "[fecha]" → "[hora]" → "dos"
Apagar todo:	"ahora" → "apagar todo" → "[fecha]" → "[hora]" → "dos"

Modo Automático:

Por otra parte, el usuario tiene que conocer la extensión (**ext. 41**) para activar el modo automático en el hogar. El modo automático activado implica que el hogar toma decisiones en función de los eventos ocurridos en el entorno. Si está desactivado, el hogar no reacciona ante los eventos y es el propio usuario quien tiene que ordenar encender o apagar los dispositivos.

Esta funcionalidad está diseñada para que el usuario tome o ceda el control sobre sus dispositivos. Por ejemplo, si la casa enciende automáticamente el calefactor en respuesta a un evento y el usuario ordena apagarlo mediante una llamada a la centralita, esto provoca que se desactive el modo automático. Si el usuario desea ceder nuevamente el control de los dispositivos a la casa tiene que marcar la extensión 41 para hacerlo.

Verificación del reconocimiento de voz:

Del mismo modo, el usuario debe conocer la **extensión 30**. Esta extensión se ha habilitado para que el usuario disponga de una herramienta que le permita testear si la API STT de Google reconoce su voz correctamente. Cuando se marca esta extensión, se emite un tono después del cual se puede decir a viva voz cualquier frase. El resultado de la traducción de Google y la probabilidad de ser correcto se envía al chat de Telegram del usuario. Si la probabilidad es mayor o igual a 0.9 se considera que la traducción es correcta.

10.3.- Código fuente

En este apartado se muestra parte del código fuente del prototipo propuesto. Primeramente, se presenta la configuración del *dialplan* y posteriormente los diferentes *scripts*. Para mayor información del resto del código se puede consultar el fichero comprimido subido al portal web como un Anexo de este trabajo.

10.3.1.- Programación en Asterisk

- Fichero "*extensions.conf*":

Debido a la extensión de este fichero, se ha fragmentado en varias partes.

```
[home]
exten => 40,1,Answer()
same => n(bienvenida),Background(custom/welcome)
same => n(stt),Gosub(google_stt,s,1)
same => n,Set(textoGoogle=${utterance})
same => n,GotoIf($["${textoGoogle}" = "ahora"]?inmediata-ivr,s,1)
same => n,GotoIf($["${textoGoogle}" = "después"]?futura-ivr,s,1)
same => n,GotoIf($["${textoGoogle}" = ""]?null,1)

same => n,NoOp(Comando inválido)
same => n,Playback(custom/invalid)
same => n,Goto(stt)

exten =>null,1,NoOP(Respuesta vacía)
same => n,Playback(custom/silencio)
same => n,Goto(40,bienvenida)

exten => 41,1,Answer()
same => n,Set(State=OUT)
same => n,AGI(request.sh,${State})
same => n,Playback(custom/modoAutomatic)
same => n,Hangup()

include => google_stt_test

;*************************************************************
```

Figura 10.26.- Fichero "*extensions.conf*", contexto [home].

```
[inmediata-ivr]
exten => s,1,NoOP(IVR de acciones inmediatas)
same => n(inmediata),BackGround(custom/action)
same => n(stt),Gosub(google_stt,s,1)
same => n,Set(accionGoogle=${utterance})
same => n,GotoIf($["${accionGoogle}" = "encender luz"]?acciones,1,1)
same => n,GotoIf($["${accionGoogle}" = "apagar luz"]?acciones,2,1)
same => n,GotoIf($["${accionGoogle}" = "encender luces
navideñas"]?acciones,3,1)
same => n,GotoIf($["${accionGoogle}" = "apagar luces
navideñas"]?acciones,4,1)
same => n,GotoIf($["${accionGoogle}" = "encender calefactor en
uno"]?acciones,5,1)
same => n,GotoIf($["${accionGoogle}" = "apagar calefactor"]?acciones,6,1)
same => n,GotoIf($["${accionGoogle}" = "encender aire en
uno"]?acciones,7,1)
same => n,GotoIf($["${accionGoogle}" = "apagar aire"]?acciones,8,1)
same => n,GotoIf($["${accionGoogle}" = "encender aspersor"]?acciones,9,1)
```

```
same => n,GotoIf($["${accionGoogle}" = "apagar aspersor"]?acciones,10,1)
same => n,GotoIf($["${accionGoogle}" = "abrir puerta"]?acciones,11,1)
same => n,GotoIf($["${accionGoogle}" = "cerrar puerta"]?acciones,12,1)
same => n,GotoIf($["${accionGoogle}" = "enviar mensaje"]?acciones,13,1)
same => n,GotoIf($["${accionGoogle}" = "apagar panel"]?acciones,14,1)
same => n,GotoIf($["${accionGoogle}" = "encender calefactor en
dos"]?acciones,15,1)
same => n,GotoIf($["${accionGoogle}" = "encender calefactor en
tres"]?acciones,16,1)
same => n,GotoIf($["${accionGoogle}" = "encender aire en
dos"]?acciones,17,1)
same => n,GotoIf($["${accionGoogle}" = "encender aire en
tres"]?acciones,18,1)
same => n,GotoIf($["${accionGoogle}" = "activar alarma"]?acciones,19,1)
same => n,GotoIf($["${accionGoogle}" = "desactivar alarma"]?acciones,20,1)
same => n,GotoIf($["${accionGoogle}" = "apagar todo"]?acciones,21,1)
same => n,GotoIf($["${accionGoogle}" = ""]?null,1)

same => n,NoOp(Comando inválido)
same => n,Playback(custom/invalid)
same => n,Goto(inmediata)

exten =>null,1,NoOP(Respuesta vacía)
same => n,Playback(custom/silencio)
same => n,Goto(s,inmediata)

;*********************************************************************
```

Figura 10.27.- Fichero "*extensions.conf*", contexto [inmediata-ivr].

```
[acciones]
exten => 1,1,NoOp(Luz LED ON)
same => n,Set(State=LUZ)
same => n,Goto(25,1)

exten => 2,1,NoOp(Luz LED OFF)
same => n,Set(State=NLZ)
same => n,Goto(25,1)

exten => 3,1,NoOp(Módulo RGB ON)
same => n,Set(State=RGB)
same => n,Goto(25,1)

exten => 4,1,NoOp(Módulo RGB OFF)
same => n,Set(State=NRB)
same => n,Goto(25,1)
```

```
exten => 5,1,NoOp(Calefactor ON)
same => n,Set(State=HO1)
same => n,Goto(25,1)

exten => 6,1,NoOp(Calefactor OFF)
same => n,Set(State=OFF)
same => n,Goto(25,1)

exten => 7,1,NoOp(Acondicionado ON)
same => n,Set(State=CO1)
same => n,Goto(25,1)

exten => 8,1,NoOp(Acondicionado OFF)
same => n,Set(State=OFF)
same => n,Goto(25,1)
exten => 9,1,NoOp(Aspersor ON)
same => n,Set(State=ASP)
same => n,Goto(25,1)

exten => 10,1,NoOp(Aspersor OFF)
same => n,Set(State=NAP)
same => n,Goto(25,1)

exten => 11,1,NoOp(Puerta OPEN)
same => n,Set(State=OPE)
same => n,Goto(25,1)

exten => 12,1,NoOp(Puerta CLOSE)
same => n,Set(State=CLO)
same => n,Goto(25,1)

exten => 13,1,NoOp(Panel electrónico)
same => n,Playback(custom/mensaje)
same => n(stt),Gosub(google_stt,s,1)
same => n,GotoIf($[${confidence} < 0.9]?mal:bien)
same => n(mal),Playback(custom/badquality)
same => n,Goto(stt)
same => n(bien),Set(State=sms=${utterance}%)
same => n,Goto(25,1)

exten => 14,1,NoOp(Panel OFF)
same => n,Set(State=NKB)
same => n,Goto(25,1)

exten => 15,1,NoOp(Calefactor ON)
same => n,Set(State=HO2)
same => n,Goto(25,1)
```

```
exten => 16,1,NoOp(Calefactor ON)
same => n,Set(State=HO3)
same => n,Goto(25,1)

exten => 17,1,NoOp(Acondicionado ON)
same => n,Set(State=CO2)
same => n,Goto(25,1)

exten => 18,1,NoOp(Acondicionado ON)
same => n,Set(State=CO3)
same => n,Goto(25,1)

exten => 19,1,NoOp(Alarma ON)
same => n,Set(State=SIR)
same => n,Goto(25,1)

exten => 20,1,NoOp(Alarma OFF)
same => n,Set(State=NSR)
same => n,Goto(25,1)

exten => 21,1,NoOp(Todo OFF)
same => n,Set(State=ALL)
same => n,Goto(25,1)

exten => 22,1,NoOp(Enviar mensaje futuro)
same => n,Set(State=${FILE(/tmp/anotamensaje.txt,0,32)})
same => n,Goto(25,1)

exten => 25,1,NoOp(Líneas comunes a todas las acciones)
same => n,AGI(request.sh,${State})
same => n,Goto(continuar_fin-ivr,s,1)

;*********************************************************************
```

Figura 10.28.- Fichero "*extensions.conf*", contexto [acciones].

```
[futura-ivr]
exten => s,1,NoOP(IVR de acciones futuras)
same => n(futura),BackGround(custom/action)
same => n,Gosub(google_stt,s,1)
same => n,Set(accFutGoogle=${utterance})
same => n,GotoIf($["${accFutGoogle}" = "encender luz"]?asignac1:)
same => n,GotoIf($["${accFutGoogle}" = "apagar luz"]?asignac2:)
same => n,GotoIf($["${accFutGoogle}" = "encender luces
navideñas"]?asignac3:)
same => n,GotoIf($["${accFutGoogle}" = "apagar luces navideñas"]?asignac4:)
same => n,GotoIf($["${accFutGoogle}" = "encender calefactor en
```

```
uno"]?asignac5:)
same => n,GotoIf($["${accFutGoogle}" = "apagar calefactor"]?asignac6:)
same => n,GotoIf($["${accFutGoogle}" = "encender aire en uno"]?asignac7:)
same => n,GotoIf($["${accFutGoogle}" = "apagar aire"]?asignac8:)
same => n,GotoIf($["${accFutGoogle}" = "encender aspersor"]?asignac9:)
same => n,GotoIf($["${accFutGoogle}" = "apagar aspersor"]?asignac10:)
same => n,GotoIf($["${accFutGoogle}" = "abrir puerta"]?asignac11:)
same => n,GotoIf($["${accFutGoogle}" = "cerrar puerta"]?asignac12:)
same => n,GotoIf($["${accFutGoogle}" = "enviar mensaje"]?asignac13:)
same => n,GotoIf($["${accFutGoogle}" = "apagar panel"]?asignac14:)
same => n,GotoIf($["${accFutGoogle}" = "encender calefactor en
dos"]?asignac15:)
same => n,GotoIf($["${accFutGoogle}" = "encender calefactor en
tres"]?asignac16:)
same => n,GotoIf($["${accFutGoogle}" = "encender aire en dos"]?asignac17:)
same => n,GotoIf($["${accFutGoogle}" = "encender aire en tres"]?asignac18:)
same => n,GotoIf($["${accFutGoogle}" = "activar alarma"]?asignac19:)
same => n,GotoIf($["${accFutGoogle}" = "desactivar alarma"]?asignac20:)
same => n,GotoIf($["${accFutGoogle}" = "apagar todo"]?asignac21:)
same => n,GotoIf($["${accFutGoogle}" = ""]?null,1)

same => n,NoOp(Comando inválido)
same => n,Playback(custom/invalid)
same => n,Goto(futura)

same => n(asignac1),Set(ext=1)
same => n,Goto(sigue,1)
same => n(asignac2),Set(ext=2)
same => n,Goto(sigue,1)
same => n(asignac3),Set(ext=3)
same => n,Goto(sigue,1)
same => n(asignac4),Set(ext=4)
same => n,Goto(sigue,1)
same => n(asignac5),Set(ext=5)
same => n,Goto(sigue,1)
same => n(asignac6),Set(ext=6)
same => n,Goto(sigue,1)
same => n(asignac7),Set(ext=7)
same => n,Goto(sigue,1)
same => n(asignac8),Set(ext=8)
same => n,Goto(sigue,1)
same => n(asignac9),Set(ext=9)
same => n,Goto(sigue,1)
same => n(asignac10),Set(ext=10)
same => n,Goto(sigue,1)
same => n(asignac11),Set(ext=11)
same => n,Goto(sigue,1)
```

```
same => n(asignac12),Set(ext=12)
same => n,Goto(sigue,1)

same => n(asignac13),Set(ext=22)
same => n,Playback(custom/mensaje)
same => n(stt),Gosub(google_stt,s,1)
same => n,GotoIf($[${confidence} < 0.9]?mal:bien)
same => n(mal),Playback(custom/badquality)
same => n,Goto(stt)
same => n(bien),Set(mensj=sms=${utterance}%)
same => n,AGI(anotamensaje.sh,${mensj})
same => n,Goto(sigue,1)


same => n(asignac14),Set(ext=14)
same => n,Goto(sigue,1)
same => n(asignac15),Set(ext=15)
same => n,Goto(sigue,1)
same => n(asignac16),Set(ext=16)
same => n,Goto(sigue,1)
same => n(asignac17),Set(ext=17)
same => n,Goto(sigue,1)
same => n(asignac18),Set(ext=18)
same => n,Goto(sigue,1)
same => n(asignac19),Set(ext=19)
same => n,Goto(sigue,1)
same => n(asignac20),Set(ext=20)
same => n,Goto(sigue,1)
same => n(asignac21),Set(ext=21)
same => n,Goto(sigue,1)

exten =>null,1,NoOP(Respuesta vacía)
same => n,Playback(custom/silencio)
same => n,Goto(s,futura)

;------DEFINIR FECHA-------
exten => sigue,1,NoOp(Definir fecha y hora)
same => n(date),Playback(custom/date)
same => n,Gosub(google_stt,s,1)
same => n,Set(fecha=${utterance})
same => n,AGI(checkdate.sh,${fecha})
same => n,Set(fecha=${FILE(/tmp/checkdate.txt,0,8)})
same => n,NoOp(TextoFichero=${fecha})
same => n,GotoIf($["${fecha}" = ""]?nook:hora)
same => n(nook),Playback(custom/formatInv)
same => n,Goto(date)
```

```
;--------DEFINIR HORA------
same => n(hora),Playback(custom/time)
same => n,Gosub(google_stt,s,1)
same => n,Set(horario=${utterance})
same => n,AGI(checktime.sh,${horario})
same => n,Set(horario=${FILE(/tmp/checktime.txt,0,4)})
same => n,NoOp(TextoFichero: ${horario})
same => n,GotoIf($["${horario}" = ""]?nookH:sigue2,1)
same => n(nookH),Playback(custom/formatInv)
same => n,Goto(hora)

;---------CREAR CALLFILE--------
exten => sigue2,1,NoOp(Creando callfiles)
same => n,AGI(creating_callfile.sh,${fecha},${horario},${ext})
same => n,Playback(custom/exito)
same => n,Goto(continuar_fin-ivr,s,1)

;*********************************************************
```

Figura 10.29.- Fichero "*extensions.conf*", contexto [futura-ivr].

```
[continuar_fin-ivr]
exten => s,1,Background(custom/continue-end)
same => n(stt),Gosub(google_stt,s,1)
same => n,Set(textoGoogle=${utterance})
same => n,GotoIf($["${textoGoogle}" = "uno"]?1,1:2,1)

exten => 1,1,NoOp(Continuar)
same => n,Goto(home,40,bienvenida)

exten => 2,1,NoOp(Finalizar)
same => n,Playback(custom/bye)
same => n,Hangup()

;*********************************************************
```

Figura 10.30.- Fichero "*extensions.conf*", contexto [continuar_fin-ivr].

```
[google_stt]
exten => s,1,agi(speech-recog.agi,es,2,#) ;Script de Zaf, idioma español, 2
segundos de timeout , y # tecla de interrupción/finalización.
same => n,Verbose(1,Lo que dijiste fue: ${utterance}) ;Verbose muestra el
contenido de la variable utterance en la consola de Asterisk.
same => n,Verbose(1,La probabilidad de ser correcto es: ${confidence}) ;lo
mismo pero con la variable confidence.
same => n,Verbose(1,La cantidad de caracteres que tiene utterance es:
${LEN(${utterance})}) ;Muestra la cantidad de caracteres que tiene
```

```
utterance.
same => n,Return

;*****************************************************
[call_interna]
exten => 555,1,NoOp(Esto es una llamada local para crear callfiles)
same => n,Answer()
same => n,Hangup()

;*****************************************************
[google_stt_test]
exten => 30,1,Answer(); contestar automáticamente la llamada.
same => n,agi(speech-recog.agi,es,3,#)
same => n,Verbose(1,Lo que dijiste fue: ${utterance})
same => n,Verbose(1,La probabilidad de ser correcto es: ${confidence})
same => n,Verbose(1,La cantidad de caracteres que tiene utterance es:
${LEN(${utterance})})
same => n,Set(text= Lo que dijiste fue: )
same => n,AGI(telegram.sh,${text}${utterance})
same => n,Set(text1= La probabilidad de ser correcto es: )
same => n,AGI(telegram.sh,${text1}${confidence})
same => n,NoOP(${utterance})
same => n,Hangup() ;Se termina la llamada.

;*****************************************************
```

Figura 10.31.- Fichero "*extensions.conf*", otros contextos.

10.3.2.- Programación de los *Scripts* AGI

A continuación, se muestra el código de los diferentes *scripts* que se han empleado.

- *Script* "request.sh":

```
#!/bin/bash

echo $1 >> /tmp/fichero.txt
argumento=$1
argum_parseado=(${argumento// /@})

echo "ARGUMENTO" > /tmp/ficheroTest.txt
echo $argum_parseado >> /tmp/ficheroTest.txt

curl http://192.168.1.131/$argum_parseado
```

Figura 10.32.- *Script* "request.sh".

- *Script* "checkdate.sh":

```
#!/bin/bash
cd /var/lib/asterisk/agi-bin/

#Recibe la fecha como entrada en el formato (DD de MM del AAAA)
fecha=$1

year=${fecha: -4}

#Reemplaza la coincidencia
fecha=${fecha/enero/01}
fecha=${fecha/febrero/02}
fecha=${fecha/marzo/03}
fecha=${fecha/abril/04}
fecha=${fecha/mayo/05}
fecha=${fecha/junio/06}
fecha=${fecha/julio/07}
fecha=${fecha/agosto/08}
fecha=${fecha/septiembre/09}
fecha=${fecha/octubre/10}
fecha=${fecha/noviembre/11}
fecha=${fecha/diciembre/12}

echo "FECHA---" > /tmp/salida.txt
echo $fecha >> /tmp/salida.txt

#Reemplazo global(//) en la cadena fecha de los espacios por espacio, luego
se accede al 1er elem del array
fechaArray=(${fecha// / })
dia=${fechaArray[0]}
mes=${fechaArray[2]}

echo "MES---" >> /tmp/salida.txt
echo $mes >> /tmp/salida.txt

#Devuelve el tamaño de la variable dia
size=${#dia}
if [ $size = 1 ]
then
dia=0${dia}
fi

fecha=$year-$mes-$dia
fecha1=$year$mes$dia

#date -d $fecha>/dev/null 2>&1 && echo "si" || echo "no"
```

```
rm /tmp/checkdate.txt
touch /tmp/checkdate.txt
chmod 777 /tmp/checkdate.txt

if date -d $fecha >/dev/null 2>&1
then
 echo -n $fecha1 > /tmp/checkdate.txt
else
 echo -n > /tmp/checkdate.txt
fi
```

Figura 10.33.- *Script* "checkdate.sh".

- *Script* "checktime.sh":

```
#!/bin/bash
cd /var/lib/asterisk/agi-bin/

#Recibe la hora como entrada en el formato (hh:mm / hh y mm)
horario=$1

#TRATAMIENTO DE LA HORA
horario=$(echo $horario | sed 's/ //g') #Borra los espacios en blanco que
puede traer de la traducción de Google

if [[ "$horario" == *":"* ]]; then

  horas=$(echo $horario | cut -d: -f1)
  minutos=$(echo $horario | cut -d: -f2)

elif [[ "$horario" == *"y"* ]];  then

  horas=$(echo $horario | cut -d'y' -f1)
  minutos=$(echo $horario | cut -d'y' -f2)

fi
#Se lleva a formato de dos dígitos
if [ ${#horas} = 1 ]
then
 horas=0${horas}
fi

if [ ${#minutos} = 1 ]
then
 minutos=0$minutos
fi
```

```
#Se concatena horas y minutos para tener formato compatible
horario=$horas$minutos
rm /tmp/checktime.txt
touch /tmp/checktime.txt
chmod 777 /tmp/checktime.txt

if date -d $horario >/dev/null 2>&1
then
 echo -n $horario > /tmp/checktime.txt
else
 echo -n > /tmp/checktime.txt
fi
```

Figura 10.34.- *Script* "checktime.sh".

- *Script* "anotamensaje.sh":

```
#!/bin/bash

touch /tmp/anotamensaje.txt
chmod 777 /tmp/anotamensaje.txt

echo $1 > /tmp/anotamensaje.txt
```

Figura 10.35.- *Script* "anotamensaje.sh".

- *Script* "creating_callfile.sh":

```
#!/bin/bash
cd /var/lib/asterisk/agi-bin/

#Recibe la fecha en el formato AAAAMMDD y el horario en el formato hhmm
fecha=$1
horario=$2
exten=$3

touch $fecha-$horario.call
chmod 777 $fecha-$horario.call
echo Channel: Local/555@call_interna >> $fecha-$horario.call
echo Callerid: smarthome >> $fecha-$horario.call
echo WaitTime: 30 >> $fecha-$horario.call
echo MaxRetries: 3 >> $fecha-$horario.call
echo RetryTime: 120 >> $fecha-$horario.call
echo Context: acciones >> $fecha-$horario.call
echo Extension: $exten >> $fecha-$horario.call
```

```
echo Archive: yes >> $fecha-$horario.call
touch -t $fecha$horario.00 $fecha-$horario.call
mv $fecha-$horario.call /var/spool/asterisk/outgoing/
```

Figura 10.36.- *Script* "creating_callfile.sh".

- *Script* "telegram.sh":

```
#!/bin/bash

token="188...73:AAH...9D....PGSIv..." # Token del bot de Telegram
chatId="-1001.......6"  #Id del chat Asterisk+Arduino

url="https://api.telegram.org/bot$token/sendMessage"

curl -s -X POST $url -d chat_id=$chatId -d text="$*"
```

Figura 10.37.- *Script* "telegram.sh".

- *Script* "notif.php":

```
<?php
/*Arduino invoca este script php alojado en apache pasando un # como
parámetro de la petición http y este script invoca directamente el script
telegram.sh y se le pasa como parámetro el texto que se desee enviar*/

// Recibe desde Arduino un número correspondiente a la acción y la
temperatura
$var = $_GET['var'];
$temp = $_GET['temp'];

//echo "La variable es: " . $var;

switch ($var) {
    case 0: //Alarma
        $msj="INTRUSO!! ALARMA ACTIVADA EN CASA";
        break;

    case 1: //Aire On
        $msj="Se ha encendido el AIRE_ACONDICIONADO: La temperatura ha
SUPERADO los ". $temp . "ºC.";
        break;
    case 2: //Aire Off
        $msj="Se ha apagado el AIRE_ACONDICIONADO: La temperatura está por
```

```
DEBAJO de ". $temp . "ºC.";
        break;

    case 3: //Cale On
        $msj="Se ha encendido la CALEFACCIÓN: La temperatura está por
DEBAJO de ". $temp . "ºC.";
        break;

    case 4: //Cale Off
        $msj="Se ha apagado la CALEFACCIÓN: La temperatura ha SUPERADO los
". $temp . "ºC.";
        break;

    case 5: //Mensaje enviado
        $msj="Se ha enviado el mensaje.";
        break;

    case 6: //Dispositivo On
        $msj="Dispositivo ENCENDIDO";
        break;

    case 7: //Dispositivo Off
        $msj="Dispositivo APAGADO";
        break;

    case 8: //OPEN
        $msj="Puerta ABIERTA";
        break;

    case 9: //CLOSE
        $msj="Puerta CERRADA";
        break;
}

exec("/var/lib/asterisk/agi-bin/telegram.sh $msj");

?>
```

Figura 10.38.- *Script* " notif.php ".

11.- REFERENCIAS

[1] L. Hernandez y M. Ospina, «Scheme and Creation of a Prototype for the Supervision of Lights and Electronic Devices with a PBX, Using a WLAN Solution Based on IoT», en *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, jun. 2019, pp. 1-6. doi: 10.1109/ColComCon.2019.8809159.

[2] M. Spahic, A. Secerbegovic, V. Mesic, H. Hadzic, A. Hasanbasic, y O. Jahic, «Smart home IVR-based system with south Slavic language integration», en *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, sep. 2020, pp. 442-445. doi: 10.23919/MIPRO48935.2020.9245187.

[3] M. B. Andrés, *Internet de las Cosas*. Editorial Reus, 2021.

[4] H. Tschofenig, J. Arkko, D. Thaler, y D. R. McPherson, «Architectural Considerations in Smart Object Networking», Internet Engineering Task Force, Request for Comments RFC 7452, mar. 2015. doi: 10.17487/RFC7452.

[5] Ó. Torrente Artero, *Arduino: curso práctico de formación*. México, D.F.: Alfaomega, 2013.

[6] «Osoyoo Mega2560 Board — Fully compatible with Arduino Mega2560 Rev.3 « osoyoo.com». https://osoyoo.com/2017/08/30/osoyoo-mega2560-board-fully-compatible-with-arduino-mega2560-rev-3/ (accedido 20 de junio de 2022).

[7] «Introduction of OSOYOO MEGA-IoT Shield « osoyoo.com». https://osoyoo.com/2019/10/18/introduction-of-osoyoo-mega-iot-shield/ (accedido 20 de junio de 2022).

[8] «OSOYOO Smart Home IoT Learning Kit for Arduino Mega2560 « osoyoo.com». https://osoyoo.com/2019/10/18/osoyoo-smart-home-iot-learning-kit-with-mega2560-introduction/ (accedido 20 de junio de 2022).

[9] «KS0085 Keyestudio Smart Home Kit for Arduino - Keyestudio Wiki». https://wiki.keyestudio.com/KS0085_Keyestudio_Smart_Home_Kit_for_Arduino (accedido 20 de junio de 2022).

[10] «Arduino - Home». https://www.arduino.cc/ (accedido 20 de junio de 2022).

[11] E. Schooler *et al.*, «RFC 3261. SIP: Session Initiation Protocol», Internet Engineering Task Force, Request for Comments RFC 3261, jul. 2002. doi: 10.17487/RFC3261.

[12] J. G. López, *VoIP y Asterisk: redescubriendo la telefonía*. Grupo Editorial RA-MA.

[13] A. Escudero--Pascual y L. Berthilson, «VoIP para el desarrollo: Una guía para crear una infraestructura de voz en regiones en desarrollo». 2006.

[14] R. Frederick, S. L. Casner, V. Jacobson, y H. Schulzrinne, «RTP: A Transport Protocol for Real-Time Applications», Internet Engineering Task Force, Request for Comments RFC 1889, ene. 1996. doi: 10.17487/RFC1889.

[15] H. Schulzrinne, S. L. Casner, R. Frederick, y V. Jacobson, «RTP: A Transport Protocol for Real-Time Applications», Internet Engineering Task Force, Request for Comments RFC 3550, jul. 2003. doi: 10.17487/RFC3550.

[16] «PJSIP Configuration Sections and Relationships - Asterisk Project - Asterisk Project Wiki». https://wiki.asterisk.org/wiki/display/AST/PJSIP+Configuration+Sections+and+Relationships (accedido 20 de junio de 2022).

[17] L. Madsen, J. V. Meggelen, y R. Bryant, *Asterisk: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[18] F. E. Goncalves, *Configuration Guide for Asterisk PBX*. Booksurge, 2007.

[19] «Asterisk 13 Dialplan Applications - Asterisk Project - Asterisk Project Wiki». https://wiki.asterisk.org/wiki/display/AST/Asterisk+13+Dialplan+Applications (accedido 20 de junio de 2022).

[20] «Asterisk 13 Application_AGI - Asterisk Project - Asterisk Project Wiki». https://wiki.asterisk.org/wiki/display/AST/Asterisk+13+Application_AGI (accedido 20 de junio de 2022).

[21] «Asterisk Call Files - Asterisk Project - Asterisk Project Wiki». https://wiki.asterisk.org/wiki/display/AST/Asterisk+Call+Files (accedido 20 de junio de 2022).

[22] E. M. Talón, *Apache*. Ministerio de Educación, 2012.

[23] Ramey, Chet, «Bash Reference Manual». diciembre de 2020. Accedido: 20 de junio de 2022. [En línea]. Disponible en: https://www.gnu.org/software/bash/manual/bash.pdf

[24] «Speech-to-Text: reconocimiento de voz automático | Cloud Speech-to-Text», *Google Cloud*. https://cloud.google.com/speech-to-text?hl=es (accedido 20 de junio de 2022).

[25] I. Potamitis, K. Georgila, N. Fakotakis, y G. Kokkinakis, «An Integrated System for Smart-Home Control of Appliances Based on Remote Speech Interaction», p. 4, 2003.

[26] V. Deka, D. S. K. Sarma, S. Sarma, y R. Sandilya, «BUILDING IVR FRAMEWORK THROUGH ASTERISK FOR CONTROLLING HOME APPLIANCES», p. 10.

[27] M. C. Dias, D. C. Lucena, y E. P. Santos, «O USO DO ASTERISK PARA O CONTROLE REMOTO DE SISTEMAS DE AUTOMAÇÃO», p. 7, 2014.

[28] D. De Freitas Melo, E. De Souza Lage, A. V. Rocha, y B. De Jesus Cardoso, «Improving the consumption and water heating efficiency in smart buildings», en *2017 13th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*, nov. 2017, pp. 1-6. doi: 10.1109/CEWIT.2017.8263304.

[29] D. A. R. Mosquera y S. M. P. Rueda, «Diseño e implementación de un sistema domótico para la seguridad del hogar controlado vía central Asterisk e Interfaz de Hardware Arduino.», *undefined*, 2015, Accedido: 20 de junio de 2022. [En línea]. Disponible en: https://www.semanticscholar.org/paper/Dise%C3%B1o-e-

implementaci%C3%B3n-de-un-sistema-dom%C3%B3tico-para-Mosquera-Rueda/14b8ffc4e3bb3fbe10f0f9856e89702e65cceb67

[30] E. R. Barahona Padilla y D. G. Huilcapi Ruiz, «Diseño e Implementación de un Sistema de Control Domótico Supervisado por un Teléfono Móvil Mediante la Utilización de Asterisk.», nov. 2015, Accedido: 20 de junio de 2022. [En línea]. Disponible en: http://dspace.espoch.edu.ec/handle/123456789/5051

[31] Á. F. Veintimilla Ocaña y C. A. Yunga Sánchez, «Diseño e implementación de un prototipo de control domótico de bajo costo activado por voz para personas con discapacidad motriz.», may 2018, Accedido: 20 de junio de 2022. [En línea]. Disponible en: http://dspace.espoch.edu.ec/handle/123456789/9149

[32] D. G. Tomala Cuenca, «Sistema domótico controlado por voz para personas con discapacidad en extremidades superiores, utilizandi tarjeta raspberry Pi.», ene. 2018, Accedido: 20 de junio de 2022. [En línea]. Disponible en: http://dspace.ups.edu.ec/handle/123456789/15141

[33] L. Zafiris, *zaf/asterisk-speech-recog*. 2022. Accedido: 20 de junio de 2022. [En línea]. Disponible en: https://github.com/zaf/asterisk-speech-recog

[34] «Software». https://www.arduino.cc/en/software (accedido 20 de junio de 2022).

[35] *Librería WiFiEsp-master de Arduino.* [En línea]. Disponible en: https://osoyoo.com/driver/WiFiEsp-master.zip

[36] «RingBuf - Arduino Reference». https://www.arduino.cc/reference/en/libraries/ringbuf/ (accedido 20 de junio de 2022).

[37] «NoDelay - Arduino Reference». https://www.arduino.cc/reference/en/libraries/nodelay/ (accedido 20 de junio de 2022).

[38] «Servo - Arduino Reference». https://www.arduino.cc/reference/en/libraries/servo/ (accedido 20 de junio de 2022).

[39] «LiquidCrystal I2C - Arduino Reference». https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/ (accedido 20 de junio de 2022).

[40] *Librería DHT11 de Arduino*. [En línea]. Disponible en: https://osoyoo.com/wp-content/uploads/samplecode/DHT.zip

[41] «Index of /releases/20.04.3». https://old-releases.ubuntu.com/releases/20.04.3/ (accedido 20 de junio de 2022).

[42] J. Mutai, «Install Asterisk 18 LTS on Ubuntu 22.04|20.04|18.04 | ComputingForGeeks», 14 de agosto de 2020. https://computingforgeeks.com/how-to-install-asterisk-pbx-on-ubuntu/ (accedido 20 de junio de 2022).

[43] D. A, «Asterisk, instalación y configuración básica en Ubuntu 18.04», *Ubunlog*, 22 de agosto de 2018. https://ubunlog.com/asterisk-instalacion-ubuntu-1804/ (accedido 20 de junio de 2022).

[44] «Download Zoiper 5, a free VoIP softphone :: Zoiper». https://www.zoiper.com/en/voip-softphone/download/current (accedido 20 de junio de 2022).

[45] «Cómo instalar el servidor web Apache en Ubuntu 20.04 | DigitalOcean». https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-20-04-es (accedido 20 de junio de 2022).

[46] «Cómo instalar la pila Linux, Apache, MySQL y PHP (LAMP) en Ubuntu 20.04 | DigitalOcean». https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04-es (accedido 20 de junio de 2022).

[47] «Telegram Web». https://web.telegram.org/z/ (accedido 20 de junio de 2022).